

Contents

[.NET documentation](#)

[Get started](#)

[Hello World](#)

[Get started tutorials](#)

[.NET Live TV](#)

[How to install](#)

[Overview](#)

[Install on Windows](#)

[Install on macOS](#)

[Install on Linux](#)

[Overview](#)

[Ubuntu](#)

[Alpine](#)

[CentOS](#)

[CentOS Stream](#)

[Debian](#)

[Fedora](#)

[OpenSUSE](#)

[Red Hat Enterprise Linux](#)

[SLES](#)

[Install with Snap](#)

[Install script & binaries](#)

[Remove outdated runtimes and SDKs](#)

[Manage .NET templates](#)

[macOS Notarization issues](#)

[Troubleshoot .NET Package mix ups on Linux](#)

[How to check .NET versions](#)

[Install localized IntelliSense](#)

[Overview](#)

[Introduction to .NET](#)

[.NET implementations](#)

[.NET class libraries](#)

[.NET Standard overview](#)

[Releases, patches, and support](#)

[Ecma standards](#)

[.NET glossary](#)

[Tutorials](#)

[.NET 6 template changes](#)

[Use Visual Studio](#)

[Create a console app](#)

[Debug an app](#)

[Publish an app](#)

[Create a library](#)

[Unit test a library](#)

[Install and use a package](#)

[Create and publish a package](#)

[Use Visual Studio Code](#)

[Create a console app](#)

[Debug an app](#)

[Publish an app](#)

[Create a library](#)

[Unit test a library](#)

[Install and use a package](#)

[Create and publish a package](#)

[Use Visual Studio for Mac](#)

[Create a console app](#)

[Debug an app](#)

[Publish an app](#)

[Create a library](#)

[Unit test a library](#)

[Install and use a package](#)

[More tutorials](#)

[What's new in .NET](#)

[.NET 6](#)

[What's new](#)

[Breaking changes](#)

[.NET 5](#)

[What's new](#)

[Breaking changes](#)

[.NET Core 3.1](#)

[What's new](#)

[Breaking changes](#)

[.NET Core 3.0](#)

[What's new](#)

[Breaking changes](#)

[.NET Core 2.2](#)

[.NET Core 2.1](#)

[What's new](#)

[Breaking changes](#)

[.NET Core 2.0](#)

[.NET Standard](#)

[Tools and diagnostics](#)

[Overview](#)

[.NET SDK](#)

[Overview](#)

[Environment variables](#)

[dotnet-install scripts](#)

[global.json overview](#)

[Telemetry](#)

[Error messages](#)

[List of all SDK errors](#)

[NETSDK1004](#)

[NETSDK1005 and NETSDK1047](#)

[NETSDK1013](#)

[NETSDK1022](#)

[NETSDK1045](#)

[NETSDK1059](#)

[NETSDK1064](#)

[NETSDK1071](#)

[NETSDK1073](#)

[NETSDK1079](#)

[NETSDK1080](#)

[NETSDK1082](#)

[NETSDK1100](#)

[NETSDK1112](#)

[NETSDK1130](#)

[NETSDK1135](#)

[NETSDK1136](#)

[NETSDK1137](#)

[NETSDK1138](#)

[NETSDK1141](#)

[NETSDK1145](#)

[NETSDK1147](#)

[NETSDK1149](#)

[NETSDK1174](#)

[NETSDK1182](#)

.NET CLI

[Overview](#)

[dotnet](#)

[dotnet add/list/remove package](#)

[dotnet add package](#)

[dotnet list package](#)

[dotnet remove package](#)

[dotnet add/list/remove reference](#)

[dotnet add reference](#)

dotnet list reference
dotnet remove reference
dotnet build
dotnet build-server
dotnet clean
dotnet dev-certs
dotnet format
dotnet help
dotnet migrate
dotnet msbuild
dotnet new
dotnet new <TEMPLATE>
dotnet new list
dotnet new search
dotnet new install
dotnet new uninstall
dotnet new update
.NET default templates
Custom templates
dotnet nuget
dotnet nuget delete
dotnet nuget locals
dotnet nuget push
dotnet nuget add source
dotnet nuget disable source
dotnet nuget enable source
dotnet nuget list source
dotnet nuget remove source
dotnet nuget update source
dotnet nuget verify
dotnet nuget trust
dotnet nuget sign

[dotnet pack](#)
[dotnet publish](#)
[dotnet restore](#)
[dotnet run](#)
[dotnet sdk check](#)
[dotnet sln](#)
[dotnet store](#)
[dotnet test](#)
[dotnet tool](#)

[dotnet tool install](#)
[dotnet tool list](#)
[dotnet tool restore](#)
[dotnet tool run](#)
[dotnet tool search](#)
[dotnet tool uninstall](#)
[dotnet tool update](#)

[dotnet vstest](#)
[dotnet watch](#)
[dotnet workload](#)

[dotnet workload install](#)
[dotnet workload list](#)
[dotnet workload repair](#)
[dotnet workload restore](#)
[dotnet workload search](#)
[dotnet workload uninstall](#)
[dotnet workload update](#)

[Elevated access](#)

[Enable Tab completion](#)

[Develop libraries with the CLI](#)

[Create templates for the CLI](#)

[1 - Create an item template](#)
[2 - Create a project template](#)

3 - Create a template package

SYSLIB diagnostics

Obsoletions

[Overview](#)

[SYSLIB0001](#)

[SYSLIB0002](#)

[SYSLIB0003](#)

[SYSLIB0004](#)

[SYSLIB0005](#)

[SYSLIB0006](#)

[SYSLIB0007](#)

[SYSLIB0008](#)

[SYSLIB0009](#)

[SYSLIB0010](#)

[SYSLIB0011](#)

[SYSLIB0012](#)

[SYSLIB0013](#)

[SYSLIB0014](#)

[SYSLIB0015](#)

[SYSLIB0016](#)

[SYSLIB0017](#)

[SYSLIB0018](#)

[SYSLIB0019](#)

[SYSLIB0020](#)

[SYSLIB0021](#)

[SYSLIB0022](#)

[SYSLIB0023](#)

[SYSLIB0024](#)

[SYSLIB0025](#)

[SYSLIB0026](#)

[SYSLIB0027](#)

[SYSLIB0028](#)

[SYSLIB0029](#)
[SYSLIB0030](#)
[SYSLIB0031](#)
[SYSLIB0032](#)
[SYSLIB0033](#)
[SYSLIB0034](#)
[SYSLIB0035](#)
[SYSLIB0036](#)
[SYSLIB0037](#)
[SYSLIB0038](#)
[SYSLIB0039](#)
[SYSLIB0040](#)
[SYSLIB0041](#)
[SYSLIB0042](#)
[SYSLIB0043](#)
[SYSLIB0047](#)

Source-generated code

[Overview](#)

[SYSLIB1001](#)
[SYSLIB1002](#)
[SYSLIB1003](#)
[SYSLIB1005](#)
[SYSLIB1006](#)
[SYSLIB1007](#)
[SYSLIB1008](#)
[SYSLIB1009](#)
[SYSLIB1010](#)
[SYSLIB1011](#)
[SYSLIB1012](#)
[SYSLIB1013](#)
[SYSLIB1014](#)
[SYSLIB1015](#)

[SYSLIB1016](#)

[SYSLIB1017](#)

[SYSLIB1018](#)

[SYSLIB1019](#)

[SYSLIB1020](#)

[SYSLIB1021](#)

[SYSLIB1022](#)

[SYSLIB1023](#)

[SYSLIB1030](#)

[SYSLIB1031](#)

[SYSLIB1032](#)

[SYSLIB1033](#)

[SYSLIB1035](#)

[SYSLIB1036](#)

[SYSLIB1037](#)

[SYSLIB1038](#)

Integrated development environments (IDEs)

[Visual Studio](#)

[Visual Studio for Mac](#)

[Visual Studio Code](#)

MSBuild and project files

[Project SDKs](#)

[Overview](#)

[Reference](#)

[Microsoft.NET.Sdk](#)

[Microsoft.NET.Sdk.Web](#)

[Microsoft.NET.Sdk.Razor](#)

[Microsoft.NET.Sdk/Desktop](#)

[Target frameworks](#)

[Dependency management](#)

Global and local tools

[Manage tools](#)

Troubleshoot tools

Create tools for the CLI

- 1 - Create a tool
- 2 - Use a global tool
- 3 - Use a local tool

Additional tools

Overview

- .NET uninstall tool
- .NET install tool for extension authors
- Generate self-signed certificates
- WCF web service reference provider
- WCF service utility
- WCF service XML serializer
- XML serializer generator

Diagnostics and instrumentation

Overview

Managed debuggers

Diagnostic port

Dumps

Overview

Linux dumps

SOS debugger extension

Logging and tracing

Overview

Well-known event providers

Event Source

Overview

Getting started

Instrumentation

Collection

Activity IDs

DiagnosticSource and DiagnosticListener

- [Getting started](#)
- [EventPipe](#)
- [Metrics](#)
 - [Overview](#)
 - [Instrumentation](#)
 - [Collection](#)
 - [EventCounters](#)
 - [Overview](#)
 - [Well-known counters](#)
 - [Tutorial: Measure performance with EventCounters](#)
 - [Compare metric APIs](#)
- [Distributed tracing](#)
 - [Overview](#)
 - [Concepts](#)
 - [Instrumentation](#)
 - [Collection](#)
- [Symbols](#)
- [Microsoft.Diagnostics.NETCore.Client library](#)
 - [Overview and examples](#)
 - [API reference](#)
- [Runtime events](#)
 - [Overview](#)
 - [Contention events](#)
 - [Exception events](#)
 - [Garbage collection events](#)
 - [Interop events](#)
 - [Loader and binder events](#)
 - [Method events](#)
 - [ThreadPool events](#)
 - [Type-system events](#)
- [Collect diagnostics in containers](#)
- [.NET CLI global tools](#)

[dotnet-counters](#)

[dotnet-coverage](#)

[dotnet-dump](#)

[dotnet-gcdump](#)

[dotnet-monitor](#)

[dotnet-trace](#)

[dotnet-stack](#)

[dotnet-symbol](#)

[dotnet-sos](#)

[dotnet-dsrouter](#)

[.NET diagnostics tutorials](#)

[Collect performance trace in Linux with PerfCollect](#)

[Debug a memory leak](#)

[Debug high CPU usage](#)

[Debug deadlock](#)

[Debug ThreadPool starvation](#)

[Debug a StackOverflow](#)

[Code analysis](#)

[Overview](#)

[Configuration](#)

[General options](#)

[Configuration files](#)

[How to suppress warnings](#)

[Code quality rules](#)

[Rule options](#)

[Predefined configurations](#)

[Code style rules](#)

[Rule options](#)

[Rule reference](#)

[Categories](#)

[Code quality rules](#)

[Overview](#)

Design rules

[Overview](#)

[CA1000](#)

[CA1001](#)

[CA1002](#)

[CA1003](#)

[CA1005](#)

[CA1008](#)

[CA1010](#)

[CA1012](#)

[CA1014](#)

[CA1016](#)

[CA1017](#)

[CA1018](#)

[CA1019](#)

[CA1021](#)

[CA1024](#)

[CA1027](#)

[CA1028](#)

[CA1030](#)

[CA1031](#)

[CA1032](#)

[CA1033](#)

[CA1034](#)

[CA1036](#)

[CA1040](#)

[CA1041](#)

[CA1043](#)

[CA1044](#)

[CA1045](#)

[CA1046](#)

[CA1047](#)

[CA1050](#)

[CA1051](#)

[CA1052](#)

[CA1053](#)

[CA1054](#)

[CA1055](#)

[CA1056](#)

[CA1058](#)

[CA1060](#)

[CA1061](#)

[CA1062](#)

[CA1063](#)

[CA1064](#)

[CA1065](#)

[CA1066](#)

[CA1067](#)

[CA1068](#)

[CA1069](#)

[CA1070](#)

[Documentation rules](#)

[Overview](#)

[CA1200](#)

[Globalization rules](#)

[Overview](#)

[CA1303](#)

[CA1304](#)

[CA1305](#)

[CA1307](#)

[CA1308](#)

[CA1309](#)

[CA1310](#)

[CA2101](#)

[Portability and interoperability rules](#)

[Overview](#)

[CA1401](#)

[CA1416](#)

[CA1417](#)

[CA1418](#)

[CA1419](#)

[Maintainability rules](#)

[Overview](#)

[CA1501](#)

[CA1502](#)

[CA1505](#)

[CA1506](#)

[CA1507](#)

[CA1508](#)

[CA1509](#)

[Naming rules](#)

[Overview](#)

[CA1700](#)

[CA1707](#)

[CA1708](#)

[CA1710](#)

[CA1711](#)

[CA1712](#)

[CA1713](#)

[CA1714](#)

[CA1715](#)

[CA1716](#)

[CA1717](#)

[CA1720](#)

[CA1721](#)

[CA1724](#)

[CA1725](#)

[CA1727](#)

[Performance rules](#)

[Overview](#)

[CA1802](#)

[CA1805](#)

[CA1806](#)

[CA1810](#)

[CA1812](#)

[CA1813](#)

[CA1814](#)

[CA1815](#)

[CA1819](#)

[CA1820](#)

[CA1821](#)

[CA1822](#)

[CA1823](#)

[CA1824](#)

[CA1825](#)

[CA1826](#)

[CA1827](#)

[CA1828](#)

[CA1829](#)

[CA1830](#)

[CA1831](#)

[CA1832](#)

[CA1833](#)

[CA1834](#)

[CA1835](#)

[CA1836](#)

[CA1837](#)

[CA1838](#)

[CA1839](#)

[CA1840](#)

[CA1841](#)

[CA1842](#)

[CA1843](#)

[CA1844](#)

[CA1845](#)

[CA1846](#)

[CA1847](#)

[CA1848](#)

[CA1849](#)

[CA1850](#)

[CA1851](#)

[CA1854](#)

SingleFile rules

[Overview](#)

[IL3000](#)

[IL3001](#)

[IL3002](#)

[IL3003](#)

Reliability rules

[Overview](#)

[CA2000](#)

[CA2002](#)

[CA2007](#)

[CA2008](#)

[CA2009](#)

[CA2011](#)

[CA2012](#)

[CA2013](#)

[CA2014](#)

[CA2015](#)

[CA2016](#)

[CA2017](#)

[CA2018](#)

[Security rules](#)

[Overview](#)

[CA2100](#)

[CA2109](#)

[CA2119](#)

[CA2153](#)

[CA2300](#)

[CA2301](#)

[CA2302](#)

[CA2305](#)

[CA2310](#)

[CA2311](#)

[CA2312](#)

[CA2315](#)

[CA2321](#)

[CA2322](#)

[CA2326](#)

[CA2327](#)

[CA2328](#)

[CA2329](#)

[CA2330](#)

[CA2350](#)

[CA2351](#)

[CA2352](#)

[CA2353](#)

[CA2354](#)

[CA2355](#)

[CA2356](#)

[CA2361](#)

CA2362

CA3001

CA3002

CA3003

CA3004

CA3005

CA3006

CA3007

CA3008

CA3009

CA3010

CA3011

CA3012

CA3061

CA3075

CA3076

CA3077

CA3147

CA5350

CA5351

CA5358

CA5359

CA5360

CA5361

CA5362

CA5363

CA5364

CA5365

CA5366

CA5367

CA5368

CA5369

CA5370

CA5371

CA5372

CA5373

CA5374

CA5375

CA5376

CA5377

CA5378

CA5379

CA5380

CA5381

CA5382

CA5383

CA5384

CA5385

CA5386

CA5387

CA5388

CA5389

CA5390

CA5391

CA5392

CA5393

CA5394

CA5395

CA5396

CA5397

CA5398

CA5399

CA5400

CA5401

[CA5402](#)

[CA5403](#)

[CA5404](#)

[CA5405](#)

[Usage rules](#)

[Overview](#)

[CA1801](#)

[CA1816](#)

[CA2200](#)

[CA2201](#)

[CA2207](#)

[CA2208](#)

[CA2211](#)

[CA2213](#)

[CA2214](#)

[CA2215](#)

[CA2216](#)

[CA2217](#)

[CA2218](#)

[CA2219](#)

[CA2224](#)

[CA2225](#)

[CA2226](#)

[CA2227](#)

[CA2229](#)

[CA2231](#)

[CA2234](#)

[CA2235](#)

[CA2237](#)

[CA2241](#)

[CA2242](#)

[CA2243](#)

[CA2244](#)

[CA2245](#)

[CA2246](#)

[CA2247](#)

[CA2248](#)

[CA2249](#)

[CA2250](#)

[CA2251](#)

[CA2252](#)

[CA2253](#)

[CA2254](#)

[CA2255](#)

[CA2256](#)

[CA2257](#)

[CA2258](#)

Code style rules

[Overview](#)

[Language rules](#)

[Overview](#)

[this and Me preferences](#)

[Use language keywords for types](#)

[Modifier preferences](#)

[Parentheses preferences](#)

[Expression-level preferences](#)

[Namespace declaration preferences](#)

[Null-checking preferences](#)

[var preferences](#)

[Expression-bodied members](#)

[Pattern matching preferences](#)

[Code block preferences](#)

[using directive preferences](#)

[File header preferences](#)

[Unnecessary code rules](#)

[Overview](#)

[IDE0001](#)

[IDE0002](#)

[IDE0004](#)

[IDE0005](#)

[IDE0035](#)

[IDE0051](#)

[IDE0052](#)

[IDE0058](#)

[IDE0059](#)

[IDE0060](#)

[IDE0079](#)

[IDE0080](#)

[IDE0081](#)

[IDE0100](#)

[IDE0110](#)

[IDE0140](#)

[Miscellaneous rules](#)

[Overview](#)

[IDE0076](#)

[IDE0077](#)

[Formatting rules](#)

[IDE0055](#)

[Naming rules](#)

[Platform compatibility analyzer](#)

[Portability analyzer](#)

[Package validation](#)

[Get started](#)

[Baseline package validator](#)

[Compatible framework in package validator](#)

[Compatible framework validator](#)

[Diagnostic IDs](#)

[Execution model](#)

[Common Language Runtime \(CLR\)](#)

[Managed execution process](#)

[Assemblies](#)

[Metadata and self-describing components](#)

[Dependency loading](#)

[Overview](#)

[Understand AssemblyLoadContext](#)

[Dependency loading details](#)

[Default dependency probing](#)

[Load managed assemblies](#)

[Load satellite assemblies](#)

[Load unmanaged libraries](#)

[Collect detailed assembly loading information](#)

[Tutorials](#)

[Create a .NET application with plugins](#)

[How to use and debug assembly unloadability](#)

[Versioning](#)

[Overview](#)

[.NET version selection](#)

[Configure .NET Runtime](#)

[Settings](#)

[Compilation settings](#)

[Debugging and profiling settings](#)

[Garbage collector settings](#)

[Globalization settings](#)

[Networking settings](#)

[Threading settings](#)

[Troubleshoot app launch failures](#)

[Deployment models](#)

[Overview](#)

[Deploy apps with Visual Studio](#)

[Publish apps with the CLI](#)

[Create a NuGet package with the CLI](#)

[Self-contained deployment runtime roll forward](#)

[Single file deployment and executable](#)

[ReadyToRun](#)

[Trim self-contained deployments](#)

[Overview and how-to](#)

[Intro to trim warnings](#)

[Trim incompatibilities](#)

[Options](#)

[Trimming libraries](#)

[Trim warnings](#)

[IL2001](#)

[IL2002](#)

[IL2003](#)

[IL2004](#)

[IL2005](#)

[IL2007](#)

[IL2008](#)

[IL2009](#)

[IL2010](#)

[IL2011](#)

[IL2012](#)

[IL2013](#)

[IL2014](#)

[IL2015](#)

[IL2016](#)

[IL2017](#)

[IL2018](#)

[IL2019](#)

[IL2022](#)

[IL2023](#)

[IL2024](#)

[IL2025](#)

[IL2026](#)

[IL2027](#)

[IL2028](#)

[IL2029](#)

[IL2030](#)

[IL2031](#)

[IL2032](#)

[IL2033](#)

[IL2034](#)

[IL2035](#)

[IL2036](#)

[IL2037](#)

[IL2038](#)

[IL2039](#)

[IL2040](#)

[IL2041](#)

[IL2042](#)

[IL2043](#)

[IL2044](#)

[IL2045](#)

[IL2046](#)

[IL2048](#)

[IL2049](#)

[IL2050](#)

[IL2051](#)

[IL2052](#)

[IL2053](#)

[IL2054](#)

[IL2055](#)

IL2056

IL2057

IL2058

IL2059

IL2060

IL2061

IL2062

IL2063

IL2064

IL2065

IL2066

IL2067

IL2068

IL2069

IL2070

IL2072

IL2073

IL2074

IL2075

IL2077

IL2078

IL2079

IL2080

IL2082

IL2083

IL2084

IL2085

IL2087

IL2088

IL2089

IL2090

IL2091

[IL2092](#)

[IL2093](#)

[IL2094](#)

[IL2095](#)

[IL2096](#)

[IL2097](#)

[IL2098](#)

[IL2099](#)

[IL2100](#)

[IL2101](#)

[IL2102](#)

[IL2103](#)

[IL2104](#)

[IL2105](#)

[IL2106](#)

[IL2107](#)

[IL2108](#)

[IL2109](#)

[IL2110](#)

[IL2111](#)

[IL2112](#)

[IL2113](#)

[IL2114](#)

[IL2115](#)

[IL2116](#)

Native AOT deployment model

Overview

AOT warnings

[IL3050](#)

[IL3051](#)

[IL3052](#)

[IL3053](#)

- [IL3054](#)
- [IL3055](#)
- [IL3056](#)
- [Runtime package store](#)
- [Runtime Identifier \(RID\) catalog](#)
- [Resource manifest names](#)
- [Docker](#)
 - [Introduction to .NET and Docker](#)
 - [Containerize a .NET app](#)
 - [Container tools in Visual Studio](#)
- [DevOps](#)
 - [GitHub Actions and .NET](#)
 - [.NET CLI and Continuous Integration](#)
 - [Official .NET GitHub Actions](#)
- [Tutorials](#)
 - [Create a GitHub Action with .NET](#)
- [Quickstarts](#)
 - [Create a build GitHub workflow](#)
 - [Create a test GitHub workflow](#)
 - [Create a publish GitHub workflow](#)
 - [Create a CodeQL GitHub workflow](#)
- [Fundamental coding components](#)
 - [Base types overview](#)
 - [Common type system & common language specification](#)
 - [Common type system](#)
 - [Language independence](#)
 - [Type conversion](#)
 - [Type conversion tables](#)
 - [Choose between anonymous and tuple types](#)
 - [Framework libraries](#)
 - [Class library overview](#)
 - [Generic types](#)

[Overview](#)

[Intro to generic types](#)

[Generic collections](#)

[Generic delegates for manipulating arrays and lists](#)

[Generic math](#)

[Generic interfaces](#)

[Covariance and contravariance](#)

[Collections and data structures](#)

[Overview](#)

[Select a collection class](#)

[Commonly used collection types](#)

[When to use generic collections](#)

[Comparisons and sorts within collections](#)

[Sorted collection types](#)

[Hashtable and Dictionary types](#)

[Thread-safe collections](#)

[Delegates and lambdas](#)

[Events](#)

[Overview](#)

[Raise and consume events](#)

[Handle multiple events using event properties](#)

[Observer design pattern](#)

[Overview](#)

[Best practices](#)

[How to: Implement a provider](#)

[How to: Implement an observer](#)

[Exceptions](#)

[Overview](#)

[Exception class and properties](#)

[How-tos](#)

[Use the try-catch block to catch exceptions](#)

[Use specific exceptions in a catch block](#)

- [Explicitly throw exceptions](#)
- [Create user-defined exceptions](#)
- [Create user-defined exceptions with localized exception messages](#)
- [Use finally blocks](#)
- [Use user-filtered exception handlers](#)
- [Handle COM interop exceptions](#)
- [Best practices](#)
- [Numeric types](#)
 - [Dates, times, and time zones](#)
- [Attributes](#)
 - [Overview](#)
 - [Apply attributes](#)
 - [Write custom attributes](#)
 - [Retrieve information stored in attributes](#)
- [Runtime libraries](#)
 - [Overview](#)
 - [Format numbers, dates, other types](#)
 - [Overview](#)
 - [Standard numeric format strings](#)
 - [Custom numeric format strings](#)
 - [Standard date and time format strings](#)
 - [Custom date and time format strings](#)
 - [Standard TimeSpan format strings](#)
 - [Custom TimeSpan format strings](#)
 - [Enumeration format strings](#)
 - [Composite formatting](#)
 - [How-tos](#)
 - [Pad a number with leading zeros](#)
 - [Extract the day of the week from a date](#)
 - [Use custom numeric format providers](#)
 - [Round-trip date and time values](#)
 - [Display milliseconds in date and time values](#)

[Display dates in non-Gregorian calendars](#)

[Work with strings](#)

[Character encoding](#)

[How to use character encoding classes](#)

[Best practices](#)

[Comparing strings](#)

[Displaying and persisting formatted data](#)

[Behavior changes in .NET 5+ \(Windows\)](#)

[Basic string operations](#)

[Overview](#)

[Create new strings](#)

[Trim and remove characters](#)

[Pad strings](#)

[Comparison methods](#)

[Change case](#)

[Separate parts of a string](#)

[Use the StringBuilder class](#)

[How to: Perform basic string manipulations](#)

[Parse \(convert\) strings](#)

[Overview](#)

[Parse numeric strings](#)

[Parse date and time strings](#)

[Parse other strings](#)

[Regular expressions](#)

[Overview](#)

[Language reference](#)

[Overview](#)

[Character escapes](#)

[Character classes](#)

[Anchors](#)

[Grouping constructs](#)

[Quantifiers](#)

- [Backreference constructs](#)
- [Alternation constructs](#)
- [Substitutions](#)
- [Regular expression options](#)
- [Miscellaneous constructs](#)
- [Best practices for regular expressions](#)
- [Regular expression object model](#)
- [Regular expression behavior](#)
 - [Overview](#)
 - [Backtracking](#)
 - [Compilation and reuse](#)
 - [Thread safety](#)
- [Examples](#)
 - [Scan for HREFs](#)
 - [Change date formats](#)
 - [Extract a protocol and port number from a URL](#)
 - [Strip invalid characters from a string](#)
 - [Verify that strings are in valid email format](#)
- [Serialization](#)
 - [Overview](#)
 - [JSON serialization](#)
 - [Overview](#)
 - [Reflection vs. source generation](#)
 - [How to serialize and deserialize JSON](#)
 - [Control serialization behavior](#)
 - [Instantiate `JsonSerializerOptions`](#)
 - [Enable case-insensitive matching](#)
 - [Customize property names and values](#)
 - [Ignore properties](#)
 - [Allow invalid JSON](#)
 - [Handle overflow JSON, use `JsonElement` or `JsonNode`](#)
 - [Preserve references, handle circular references](#)

- [Deserialize to immutable types, non-public accessors](#)
- [Polymorphic serialization](#)
- [Use DOM, Utf8JsonReader, Utf8JsonWriter](#)
- [Migrate from Newtonsoft.Json](#)
- [Supported collection types](#)
- [Advanced
 - \[Customize character encoding\]\(#\)
 - \[Use source generation\]\(#\)
 - \[Write custom converters\]\(#\)](#)
- [Binary serialization](#)
 - [Overview](#)
 - [BinaryFormatter security guide](#)
 - [BinaryFormatter event source](#)
 - [Serialization concepts](#)
 - [Basic serialization](#)
 - [Selective serialization](#)
 - [Custom serialization](#)
 - [Steps in the serialization process](#)
 - [Version-tolerant serialization](#)
 - [Serialization guidelines](#)
 - [How to: Chunk serialized data](#)
 - [How to: Determine if a .NET Standard object is serializable](#)
- [XML and SOAP serialization](#)
 - [Overview](#)
 - [XML serialization in depth](#)
 - [Examples](#)
 - [The XML Schema Definition tool](#)
 - [Control XML serialization using attributes](#)
 - [Attributes that control XML serialization](#)
 - [XML serialization with XML Web Services](#)
 - [Attributes that control encoded SOAP serialization](#)
 - [How-tos](#)

[Serialize an object](#)

[Deserialize an object](#)

[Use the XML Schema Definition tool to generate classes and XML schema documents](#)

[Control serialization of derived classes](#)

[Specify an alternate element name for an XML stream](#)

[Qualify XML element and XML attribute names](#)

[Serialize an object as a SOAP-encoded XML stream](#)

[Override encoded SOAP XML serialization](#)

[XML serialization elements](#)

[system.xml.serialization](#)

[dateTimeSerialization](#)

[schemaImporterExtensions](#)

[add element for schemaImporterExtensions](#)

[xmlSerializer](#)

[Tools](#)

[XML Serializer Generator tool \(Sgen.exe\)](#)

[XML Schema Definition tool \(Xsd.exe\)](#)

[System.CommandLine](#)

[Overview](#)

[Get started tutorial](#)

[Command-line syntax](#)

[Define commands](#)

[Model binding](#)

[Tab completion](#)

[Dependency injection](#)

[Customize help](#)

[Handle termination](#)

[Use middleware](#)

[File and stream I/O](#)

[Overview](#)

[File path formats on Windows systems](#)

[Common I/O tasks](#)

- [How to: Copy Directories](#)
- [How to: Enumerate Directories and Files](#)
- [How to: Read and Write to a Newly Created Data File](#)
- [How to: Open and Append to a Log File](#)
- [How to: Write Text to a File](#)
- [How to: Read Text from a File](#)
- [How to: Read Characters from a String](#)
- [How to: Write Characters to a String](#)
- [How to: Add or Remove Access Control List Entries](#)
- [How to: Compress and Extract Files](#)
- [Composing Streams](#)
 - [How to: Convert Between .NET Framework Streams and Windows Runtime Streams](#)
- [Asynchronous file I/O](#)
 - [Handle I/O errors](#)
 - [Isolated storage](#)
 - [Types of Isolation](#)
 - [How to: Obtain Stores for Isolated Storage](#)
 - [How to: Enumerate Stores for Isolated Storage](#)
 - [How to: Delete Stores in Isolated Storage](#)
 - [How to: Anticipate Out-of-Space Conditions with Isolated Storage](#)
 - [How to: Create Files and Directories in Isolated Storage](#)
 - [How to: Find Existing Files and Directories in Isolated Storage](#)
 - [How to: Read and Write to Files in Isolated Storage](#)
 - [How to: Delete Files and Directories in Isolated Storage](#)
 - [Pipes](#)
 - [How to: Use Anonymous Pipes for Local Interprocess Communication](#)
 - [How to: Use Named Pipes for Network Interprocess Communication](#)
 - [Pipelines](#)
 - [Work with buffers](#)
 - [Memory-mapped files](#)
 - [The System.Console class](#)
 - [Dependency injection](#)

[Overview](#)

[Use dependency injection](#)

[Dependency injection guidelines](#)

[Configuration](#)

[Overview](#)

[Configuration providers](#)

[Implement a custom configuration provider](#)

[Options pattern](#)

[Options pattern guidance for library authors](#)

[Logging](#)

[Overview](#)

[Logging providers](#)

[Compile-time logging source generation](#)

[Implement a custom logging provider](#)

[High-performance logging](#)

[Console log formatting](#)

[HostBuilder \(generic host\)](#)

[Networking](#)

[Network programming](#)

[Network availability](#)

[IPv6 overview](#)

[HTTP](#)

[HTTP support](#)

[HTTP client guidelines](#)

[Make HTTP requests](#)

[IHttpClientFactory](#)

[HTTP/3 with .NET](#)

[Rate limit an HTTP handler](#)

[Sockets](#)

[Sockets support](#)

[Use Sockets to send and receive data](#)

[TCP](#)

- [TCP support](#)
- [Use TcpClient and TcpListener](#)
- [File globbing](#)
- [Primitives library](#)
- [Globalization and localization](#)
 - [Overview](#)
 - [Globalization](#)
 - [Globalization and ICU](#)
 - [Localizability review](#)
 - [Localization](#)
 - [Culture-insensitive string operations](#)
 - [Overview](#)
 - [String comparisons](#)
 - [Case changes](#)
 - [String operations in collections](#)
 - [String operations in arrays](#)
 - [Best practices for developing world-ready apps](#)
- [Resources in .NET apps](#)
 - [Overview](#)
 - [Create resource files](#)
 - [Overview](#)
 - [Work with .resx files programmatically](#)
 - [Create satellite assemblies](#)
 - [Package and deploy resources](#)
 - [Retrieve resources](#)
- [Worker Services](#)
 - [Overview](#)
 - [Create a Queue Service](#)
 - [Use scoped services with a BackgroundService](#)
 - [Create a Windows Service using BackgroundService](#)
 - [Implement the IHostedService interface](#)
 - [Deploy a Worker Service to Azure](#)

[Caching](#)

[Channels](#)

[Data access](#)

[LINQ](#)

[XML documents and data](#)

[Microsoft.Data.Sqlite](#)

[Entity Framework Core](#)

[Parallel processing, concurrency, and async](#)

[Asynchronous programming patterns](#)

[Parallel programming](#)

[Overview](#)

[Task Parallel Library \(TPL\)](#)

[Data parallelism](#)

[How to: Write a Simple Parallel.For Loop](#)

[How to: Write a Simple Parallel.ForEach Loop](#)

[How to: Write a Parallel.For Loop with Thread-Local Variables](#)

[How to: Write a Parallel.ForEach Loop with Partition-Local Variables](#)

[How to: Cancel a Parallel.For or ForEach Loop](#)

[How to: Handle Exceptions in Parallel Loops](#)

[How to: Speed Up Small Loop Bodies](#)

[How to: Iterate File Directories with the Parallel Class](#)

[Task-based asynchronous programming](#)

[Chaining Tasks by Using Continuation Tasks](#)

[Attached and Detached Child Tasks](#)

[Task Cancellation](#)

[Exception Handling](#)

[How to: Use Parallel.Invoke to Execute Parallel Operations](#)

[How to: Return a Value from a Task](#)

[How to: Cancel a Task and Its Children](#)

[How to: Create Pre-Computed Tasks](#)

[How to: Traverse a Binary Tree with Parallel Tasks](#)

[How to: Unwrap a Nested Task](#)

[How to: Prevent a Child Task from Attaching to its Parent](#)

Dataflow

[How to: Write Messages to and Read Messages from a Dataflow Block](#)

[How to: Implement a Producer-Consumer Dataflow Pattern](#)

[How to: Perform Action When a Dataflow Block Receives Data](#)

[Walkthrough: Creating a Dataflow Pipeline](#)

[How to: Unlink Dataflow Blocks](#)

[Walkthrough: Using Dataflow in a Windows Forms Application](#)

[How to: Cancel a Dataflow Block](#)

[Walkthrough: Creating a Custom Dataflow Block Type](#)

[How to: Use JoinBlock to Read Data From Multiple Sources](#)

[How to: Specify the Degree of Parallelism in a Dataflow Block](#)

[How to: Specify a Task Scheduler in a Dataflow Block](#)

[Walkthrough: Using BatchBlock and BatchedJoinBlock to Improve Efficiency](#)

Use TPL with Other Asynchronous Patterns

[TPL and Traditional .NET Asynchronous Programming](#)

[How to: Wrap EAP Patterns in a Task](#)

Potential Pitfalls in Data and Task Parallelism

Parallel LINQ (PLINQ)

[Introduction to PLINQ](#)

[Understanding Speedup in PLINQ](#)

[Order Preservation in PLINQ](#)

[Merge Options in PLINQ](#)

[Potential Pitfalls with PLINQ](#)

[How to: Create and Execute a Simple PLINQ Query](#)

[How to: Control Ordering in a PLINQ Query](#)

[How to: Combine Parallel and Sequential LINQ Queries](#)

[How to: Handle Exceptions in a PLINQ Query](#)

[How to: Cancel a PLINQ Query](#)

[How to: Write a Custom PLINQ Aggregate Function](#)

[How to: Specify the Execution Mode in PLINQ](#)

[How to: Specify Merge Options in PLINQ](#)

- [How to: Iterate File Directories with PLINQ](#)
- [How to: Measure PLINQ Query Performance](#)
- [PLINQ Data Sample](#)
- [Data structures for parallel programming](#)
- [Parallel diagnostic tools](#)
- [Custom partitioners for PLINQ and TPL](#)
- [Overview](#)
- [How to: Implement Dynamic Partitions](#)
- [How to: Implement a Partitioner for Static Partitioning](#)
- [Lambda expressions in PLINQ and TPL](#)
- [Further reading](#)
- [Threading](#)
- [Testing](#)
 - [Overview](#)
 - [Unit testing best practices](#)
 - [xUnit](#)
 - [C# unit testing](#)
 - [F# unit testing](#)
 - [VB unit testing](#)
 - [Organize a project and test with xUnit](#)
 - [NUnit](#)
 - [C# unit testing](#)
 - [F# unit testing](#)
 - [VB unit testing](#)
 - [MSTest](#)
 - [C# unit testing](#)
 - [F# unit testing](#)
 - [VB unit testing](#)
 - [Run selective unit tests](#)
 - [Order unit tests](#)
 - [Unit test code coverage](#)
 - [Unit test published output](#)

Live unit test .NET projects with Visual Studio

Security

Advanced topics

Performance

Memory management

What is "managed code"?

Automatic memory management

Clean up unmanaged resources

Overview

Implement a Dispose method

Implement a DisposeAsync method

Use objects that implement IDisposable

Garbage collection

Overview

Fundamentals

Workstation and server GC

Background GC

The large object heap

Garbage collection and performance

Induced collections

Latency modes

Optimization for shared web hosting

Garbage collection notifications

Application domain resource monitoring

Weak references

Memory and span-related types

Overview

Memory<T> and Span<T> usage guidelines

SIMD-enabled types

Native interoperability

Overview

P/Invoke

- [Overview](#)
- [Cross-platform P/Invoke](#)
- [Source generation](#)
- [Type marshalling](#)
 - [Overview](#)
 - [Charsets and marshalling](#)
 - [Disabled marshalling](#)
 - [Customize structure marshalling](#)
 - [Customize parameter marshalling](#)
 - [Source generation](#)
- [Interop guidance](#)
 - [Expose .NET components to COM](#)
 - [Host .NET from native code](#)
 - [COM interop](#)
 - [Overview](#)
 - [COM wrappers](#)
 - [Overview](#)
 - [Runtime-callable wrapper](#)
 - [COM-callable wrapper](#)
 - [Tutorial - Use the ComWrappers API](#)
 - [Qualifying .NET types for COM interop](#)
 - [Apply interop attributes](#)
 - [Exceptions](#)
- [.NET distribution packaging](#)
- [Open-source library guidance](#)
- [Framework design guidelines](#)
 - [Overview](#)
 - [Naming guidelines](#)
 - [Capitalization conventions](#)
 - [General naming conventions](#)
 - [Names of assemblies and DLLs](#)
 - [Names of namespaces](#)

Names of classes, structs, and interfaces

Names of type members

Naming parameters

Naming resources

Type design guidelines

Choose between class and struct

Abstract class design

Static class design

Interface design

Struct design

Enum design

Nested types

Member design guidelines

Member overloading

Property design

Constructor design

Event design

Field design

Extension methods

Operator overloads

Parameter design

Design for extensibility

Unsealed classes

Protected members

Events and callbacks

Virtual members

Abstractions (abstract types and interfaces)

Base classes for implementing abstractions

Sealing

Exception design guidelines

Exception throwing

Use standard exception types

[Exceptions and performance](#)

[Usage guidelines](#)

[Arrays](#)

[Attributes](#)

[Collections](#)

[Serialization](#)

[System.Xml usage](#)

[Equality operators](#)

[Common design patterns](#)

[Dependency properties](#)

[Migration guide](#)

[Overview](#)

[General Information](#)

[About .NET](#)

[Versioning info for .NET SDK, MSBuild, and Visual Studio](#)

[Choose between .NET 5 and .NET Framework for server apps](#)

[.NET Upgrade Assistant tool](#)

[Overview](#)

[Windows Presentation Foundation](#)

[Windows Forms](#)

[Universal Windows Platform](#)

[ASP.NET Core](#)

[Telemetry](#)

[Breaking changes](#)

[Pre-Migration](#)

[Assess the portability of your project](#)

[Unsupported Dependencies](#)

[Use the Windows Compatibility Pack](#)

[Unavailable technologies](#)

[Unsupported APIs](#)

[Needed changes before porting code](#)

[Migration](#)

Create a porting plan

Approaches

Project structure

Application Porting Guides

Windows Forms

Windows Presentation Foundation

Port C++/CLI projects

Get started with .NET

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article teaches you how to create and run a "Hello World!" app with [.NET](#).

Create an application

First, download and install the [.NET SDK](#) on your computer.

Next, open a terminal such as **PowerShell**, **Command Prompt**, or **bash**.

Type the following commands:

```
dotnet new console -o sample1  
cd sample1  
dotnet run
```

You should see the following output:

```
Hello World!
```

Congratulations! You've created a simple .NET application.

Next steps

Get started on developing .NET applications by following a [step-by-step tutorial](#) or by watching [.NET 101 videos](#) on YouTube.

Tutorials for getting started with .NET

9/20/2022 • 2 minutes to read • [Edit Online](#)

The following step-by-step tutorials run on Windows, Linux, or macOS, except as noted.

Tutorials for creating apps

- Create a console app
 - [using Visual Studio Code](#)
 - [using Visual Studio \(Windows\)](#)
 - [using Visual Studio for Mac \(macOS\)](#)
- Create a web app
 - [with server-side web UI](#)
 - [with client-side web UI](#)
- [Create a web API](#)
- [Create a remote procedure call web app](#)
- [Create a real-time web app](#)
- [Create a serverless function in the cloud](#)
- [Create a mobile app for Android and iOS \(Windows\)](#)
- Create a Windows desktop app
 - [WPF](#)
 - [Windows Forms](#)
 - [Universal Windows Platform \(UWP\)](#)
- [Create a game using Unity](#)
- [Create a Windows service](#)

Tutorials for creating class libraries

- Create a class library
 - [using Visual Studio Code](#)
 - [using Visual Studio \(Windows\)](#)
 - [using Visual Studio for Mac \(macOS\)](#)

Resources for learning .NET languages

- [Get started with C#](#)
- [Get started with F#](#)
- [Get started with Visual Basic](#)

Other get-started resources

The following resources are for getting started with developing .NET apps but aren't step-by-step tutorials:

- [Internet of Things \(IoT\)](#)
- [Machine learning](#)

Next steps

To learn more about .NET, see [Introduction to .NET](#).

Install .NET on Windows

9/20/2022 • 11 minutes to read • [Edit Online](#)

In this article, you'll learn how to install .NET on Windows. .NET is made up of the runtime and the SDK. The runtime is used to run a .NET app and may or may not be included with the app. The SDK is used to create .NET apps and libraries. The .NET runtime is always installed with the SDK.

The latest version of .NET is 6.

[DOWNLOAD
.NET](#)

Install with Windows Package Manager (winget)

You can install and manage .NET through the Windows Package Manager service, using the `winget` tool. For more information about how to install and use `winget`, see [Use the winget tool](#).

If you're installing .NET system-wide, install with administrative privileges.

Install the SDK

The .NET SDK allows you to develop apps with .NET. If you install the .NET SDK, you don't need to install the corresponding runtimes. To install the .NET SDK, run the following command:

```
winget install Microsoft.DotNet.SDK.6
```

Install the runtime

For Windows, there are three .NET runtimes you can install. You should install both the .NET Desktop Runtime and the ASP.NET Core Runtime to ensure that you're compatible with all types of .NET apps.

- .NET Desktop Runtime

This runtime includes the base .NET runtime, and supports Windows Presentation Foundation (WPF) and Windows Forms apps that are built with .NET. This isn't the same as .NET Framework, which comes with Windows.

```
winget install Microsoft.DotNet/DesktopRuntime.6
```

- ASP.NET Core Runtime

This runtime includes the base .NET runtime, and runs web server apps. The ASP.NET Core Runtime allows you to run apps that were made with .NET that didn't provide the runtime. The following commands install the ASP.NET Core Runtime, which is the most compatible runtime for .NET. In your terminal, run the following commands:

```
winget install Microsoft.DotNet/AspNetCore.6
```

- .NET Runtime

This is the base runtime, and contains just the components needed to run a console app. Typically, you'd install the other runtimes.

```
winget install Microsoft.DotNet.Runtime.6
```

You can install preview versions of the runtimes by substituting the version number, such as `6`, with the word `Preview`. The following example installs the preview release of the .NET Desktop Runtime:

```
winget install Microsoft.DotNet.DesktopRuntime.Preview
```

Install alongside Visual Studio Code

Visual Studio Code is a powerful and lightweight source code editor that runs on your desktop. Visual Studio Code is available for Windows, macOS, and Linux.

While Visual Studio Code doesn't come with an automated .NET Core installer like Visual Studio does, adding .NET Core support is simple.

1. [Download and install Visual Studio Code](#).
2. [Download and install the .NET SDK](#).
3. [Install the C# extension from the Visual Studio Code marketplace](#).

Install with Windows Installer

The [download page](#) for .NET provides Windows Installer executables.

When you use the Windows installers to install .NET, you can customize the installation path by setting the `DOTNETHOME_X64` and `DOTNETHOME_X86` parameters:

```
dotnet-sdk-3.1.301-win-x64.exe DOTNETHOME_X64="F:\dotnet\x64" DOTNETHOME_X86="F:\dotnet\x86"
```

If you want to install .NET silently, such as in a production environment or to support continuous integration, use the following switches:

- `/install`
Installs .NET.
- `/quiet`
Prevents any UI and prompts from displaying.
- `/norestart`
Suppresses any attempts to restart.

```
dotnet-sdk-3.1.301-win-x64.exe /install /quiet /norestart
```

For more information, see [Standard Installer Command-Line Options](#).

TIP

The installer returns an exit code of 0 for success and an exit code of 3010 to indicate that a restart is required. Any other value is generally an error code.

Install with PowerShell automation

The [dotnet-install scripts](#) are used for CI automation and non-admin installs of the runtime. You can download the script from the [dotnet-install script reference page](#).

The script defaults to installing the latest [long term support \(LTS\)](#) version, which is .NET 6. You can choose a specific release by specifying the `-Channel` switch. Include the `-Runtime` switch to install a runtime. Otherwise, the script installs the SDK.

```
dotnet-install.ps1 -Channel 6.0 -Runtime aspnetcore
```

Install the SDK by omitting the `-Runtime` switch. The `-Channel` switch is set in this example to `Current`, which installs the latest supported version.

```
dotnet-install.ps1 -Channel Current
```

Install with Visual Studio

If you're using Visual Studio to develop .NET apps, the following table describes the minimum required version of Visual Studio based on the target .NET SDK version.

.NET SDK VERSION	VISUAL STUDIO VERSION
6.0	Visual Studio 2022 version 17.0 or higher.
5.0	Visual Studio 2019 version 16.8 or higher.
3.1	Visual Studio 2019 version 16.4 or higher.
3.0	Visual Studio 2019 version 16.3 or higher.
2.2	Visual Studio 2017 version 15.9 or higher.
2.1	Visual Studio 2017 version 15.7 or higher.

If you already have Visual Studio installed, you can check your version with the following steps.

1. Open Visual Studio.
2. Select **Help > About Microsoft Visual Studio**.
3. Read the version number from the **About** dialog.

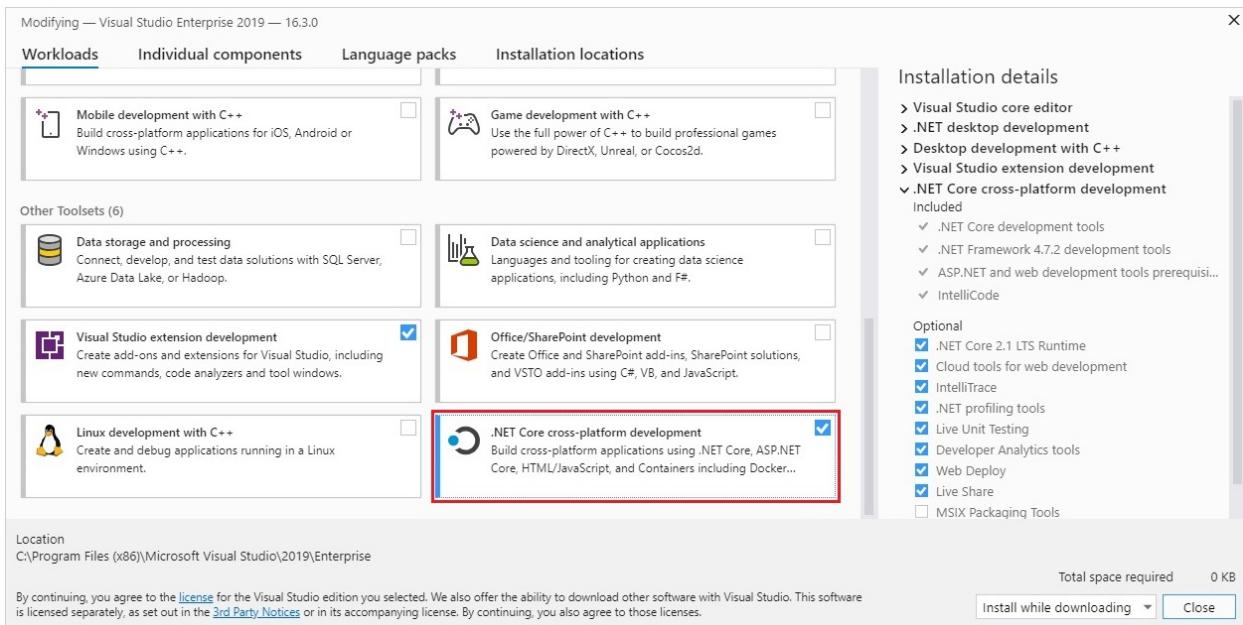
Visual Studio can install the latest .NET SDK and runtime.



Select a workload

When installing or modifying Visual Studio, select one or more of the following workloads, depending on the kind of application you're building:

- The **.NET Core cross-platform development** workload in the **Other Toolsets** section.
- The **ASP.NET and web development** workload in the **Web & Cloud** section.
- The **Azure development** workload in the **Web & Cloud** section.
- The **.NET desktop development** workload in the **Desktop & Mobile** section.



Supported releases

The following table is a list of currently supported .NET releases and the versions of Windows they're supported on. These versions remain supported until either the version of [.NET reaches end-of-support](#) or the version of [Windows reaches end-of-life](#).

Windows 10 versions end-of-service dates are segmented by edition. Only **Home**, **Pro**, **Pro Education**, and **Pro for Workstations** editions are considered in the following table. Check the [Windows lifecycle fact sheet](#) for specific details.

TIP

A symbol represents the minimum version.

OPERATING SYSTEM	.NET CORE 3.1	.NET 6
Windows 11	✓	✓
Windows Server 2022	✓	✓
Windows 10 Version 21H1	✓	✓
Windows 10 / Windows Server, Version 20H2	✓	✓
Windows 10 / Windows Server, Version 2004	✓	✓
Windows 10 / Windows Server, Version 1909	✓	✓
Windows 10 / Windows Server, Version 1903	✓	✓
Windows 10, Version 1809	✓	✓

OPERATING SYSTEM	.NET CORE 3.1	.NET 6
Windows 10, Version 1803	✓	✓
Windows 10, Version 1709	✓	✓
Windows 10, Version 1607	✓	✓
Windows 8.1	✓	✓
Windows 7 SP1 ESU	✓	✓
Windows Server 2019 Windows Server 2016 Windows Server 2012 R2 Windows Server 2012	✓	✓
Windows Server Core 2012 R2	✓	✓
Windows Server Core 2012	✓	✓
Nano Server, Version 1809+	✓	✓
Nano Server, Version 1803	✓	✗

For more information about .NET 6 supported operating systems, distributions, and lifecycle policy, see [.NET 6 Supported OS Versions](#).

Unsupported releases

The following versions of .NET are ✗ no longer supported:

- .NET 5
- .NET Core 3.0
- .NET Core 2.2
- .NET Core 2.1
- .NET Core 2.0

Runtime information

The runtime is used to run apps created with .NET. When an app author publishes an app, they can include the runtime with their app. If they don't include the runtime, it's up to the user to install the runtime.

There are three different runtimes you can install on Windows:

- *ASP.NET Core runtime*
Runs ASP.NET Core apps. Includes the .NET runtime.
- *Desktop runtime*
Runs .NET WPF and Windows Forms desktop apps for Windows. Includes the .NET runtime.
- *.NET runtime*
This runtime is the simplest runtime and doesn't include any other runtime. It's highly recommended that you install both *ASP.NET Core runtime* and *Desktop runtime* for the best compatibility with .NET apps.

[DOWNLOAD .NET](#)
[RUNTIME](#)

SDK information

The SDK is used to build and publish .NET apps and libraries. Installing the SDK includes all three [runtimes](#): ASP.NET Core, Desktop, and .NET.

[DOWNLOAD .NET](#)
[SDK](#)

Arm-based Windows PCs

The following sections describe things you should consider when installing .NET on an Arm-based Windows PC.

What's supported

The following table describes which versions of .NET are supported on an Arm-based Windows PC:

.NET VERSION	ARCHITECTURE	SDK	RUNTIME	PATH CONFLICT
6.0	Arm64	Yes	Yes	No
6.0	x64	Yes	Yes	No
5.0	Arm64	Yes	Yes	Yes
5.0	x64	No	Yes	Yes
3.1	Arm64	No	No	N/A
3.1	x64	No	Yes	Yes

The x64 and Arm64 versions of the .NET 6 SDK exist independently from each other. If a new version is released, each install needs to be upgraded.

Path differences

On an Arm-based Windows PC, all Arm64 versions of .NET are installed to the normal *C:\Program Files\dotnet* folder. However, when you install the x64 version of .NET 6 SDK, it's installed to the *C:\Program Files\dotnet\x64* folder.

Path conflicts

The x64 .NET 6 SDK installs to its own directory, as described in the previous section. This allows the Arm64 and x64 versions of the .NET 6 SDK to exist on the same machine. However, any x64 SDK prior to 6.0 isn't supported and installs to the same location as the Arm64 version, the *C:\Program Files\dotnet* folder. If you want to install an unsupported x64 SDK, you'll need to first uninstall the Arm64 version. The opposite is also true, you'll need to uninstall the unsupported x64 SDK to install the Arm64 version.

Path variables

Environment variables that add .NET to system path, such as the `PATH` variable, may need to be changed if you have both the x64 and Arm64 versions of the .NET 6 SDK installed. Additionally, some tools rely on the `DOTNET_ROOT` environment variable, which would also need to be updated to point to the appropriate .NET 6 SDK installation folder.

Dependencies

- [.NET 6](#)
- [.NET Core 3.1](#)

The following Windows versions are supported with .NET 6:

NOTE

A  symbol represents the minimum version.

OS	VERSION	ARCHITECTURES
Windows 11	21H2	x64, Arm64
Windows 10 Client	1607+	x64, x86, Arm64
Windows Client	7 SP1+, 8.1	x64, x86
Windows Server	2012+	x64, x86
Windows Server Core	2012+	x64, x86
Nano Server	1809+	x64

For more information about .NET 6 supported operating systems, distributions, and lifecycle policy, see [.NET 6 Supported OS Versions](#).

Windows 7 / Vista / 8.1 / Server 2008 R2 / Server 2012 R2

More dependencies are required if you're installing the .NET SDK or runtime on the following Windows versions:

OPERATING SYSTEM	PREREQUISITES
Windows 7 SP1 ESU	<ul style="list-style-type: none">- Microsoft Visual C++ 2015-2019 Redistributable 64-bit / 32-bit- KB3063858 64-bit / 32-bit- Microsoft Root Certificate Authority 2011 (.NET Core 2.1 offline installer only)
Windows Vista SP 2	Microsoft Visual C++ 2015-2019 Redistributable 64-bit / 32-bit
Windows 8.1	Microsoft Visual C++ 2015-2019 Redistributable 64-bit / 32-bit
Windows Server 2008 R2	Microsoft Visual C++ 2015-2019 Redistributable 64-bit / 32-bit
Windows Server 2012	Microsoft Visual C++ 2015-2019 Redistributable 64-bit / 32-bit
Windows Server 2012 R2	Microsoft Visual C++ 2015-2019 Redistributable 64-bit / 32-bit

The previous requirements are also required if you receive an error related to either of the following dlls:

- *api-ms-win-crt-runtime-l1-1-0.dll*
- *api-ms-win-cor-timezone-l1-1-0.dll*
- *hostfxr.dll*

Docker

Containers provide a lightweight way to isolate your application from the rest of the host system. Containers on the same machine share just the kernel and use resources given to your application.

.NET can run in a Docker container. Official .NET Docker images are published to the Microsoft Container Registry (MCR) and are discoverable at the [Microsoft .NET Docker Hub repository](#). Each repository contains images for different combinations of the .NET (SDK or Runtime) and OS that you can use.

Microsoft provides images that are tailored for specific scenarios. For example, the [ASP.NET Core repository](#) provides images that are built for running ASP.NET Core apps in production.

For more information about using .NET in a Docker container, see [Introduction to .NET and Docker](#) and [Samples](#).

Troubleshooting

After installing the .NET SDK, you may run into problems trying to run .NET CLI commands. This section collects those common problems and provides solutions.

- [It was not possible to find any installed .NET Core SDKs](#)

It was not possible to find any installed .NET Core SDKs

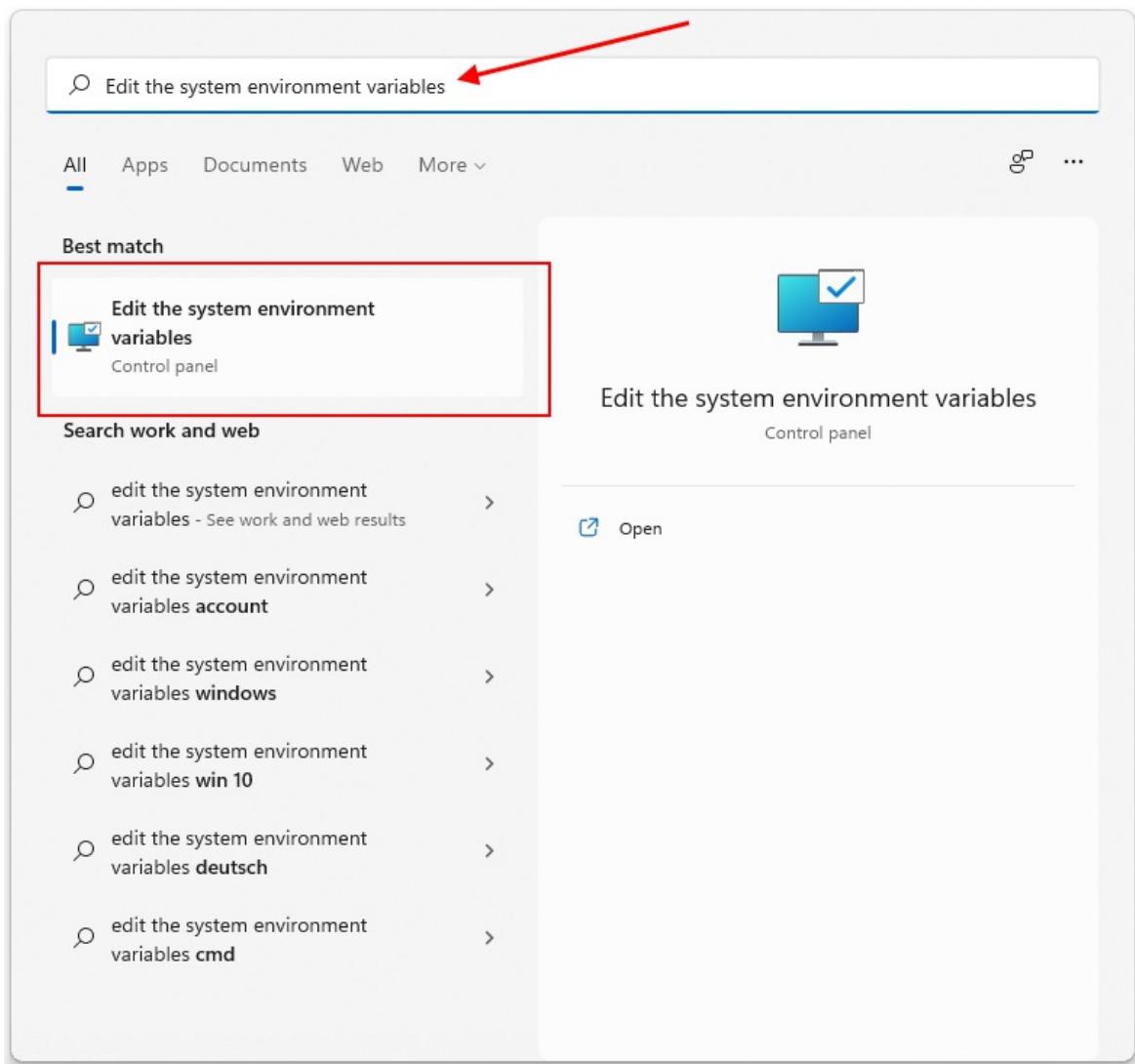
Most likely you've installed both the x86 (32-bit) and x64 (64-bit) versions of the .NET SDK. This is causing a conflict because when you run the `dotnet` command it's resolving to the x86 version when it should resolve to the x64 version. This is usually fixed by adjusting the `%PATH%` variable to resolve the x64 version first.

1. Verify that you have both versions installed by running the `where.exe dotnet` command. If you do, you should see an entry for both the *Program Files* and *Program Files (x86)* folders. If the *Program Files (x86)* folder is first as indicated by the following example, it's incorrect and you should continue on to the next step.

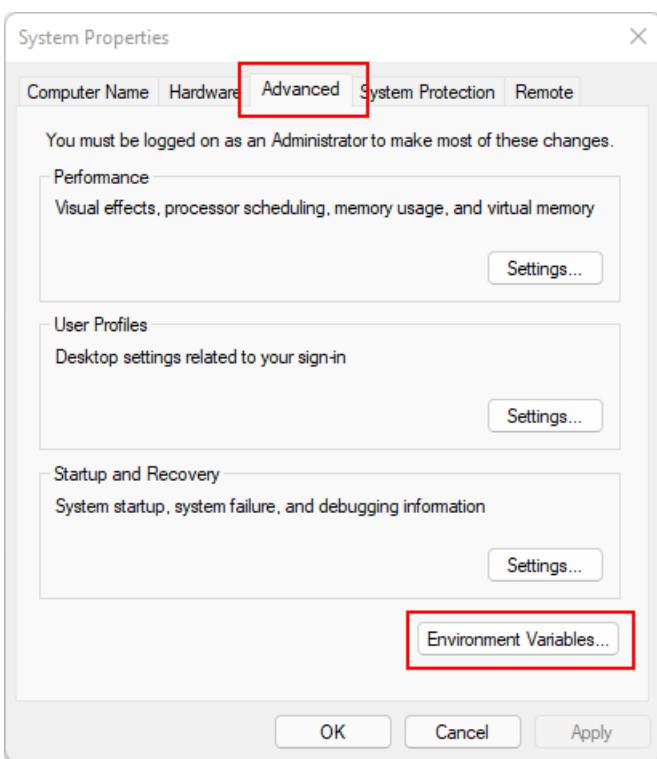
```
> where.exe dotnet
C:\Program Files (x86)\dotnet\dotnet.exe
C:\Program Files\dotnet\dotnet.exe
```

If it's correct and the *Program Files* is first, you don't have the problem this section is discussing and you should create a [.NET help request issue on GitHub](#)

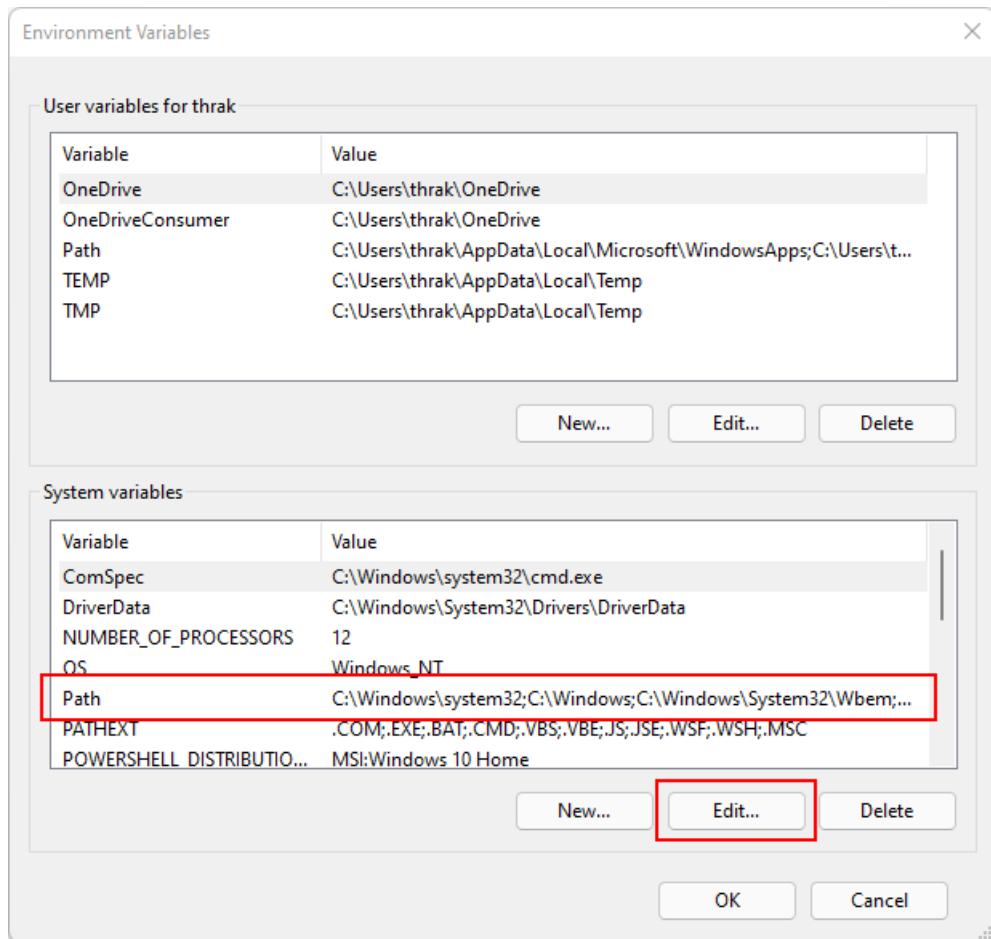
2. Press the Windows button and type "Edit the system environment variables" into search. Select **Edit the system environment variables**.



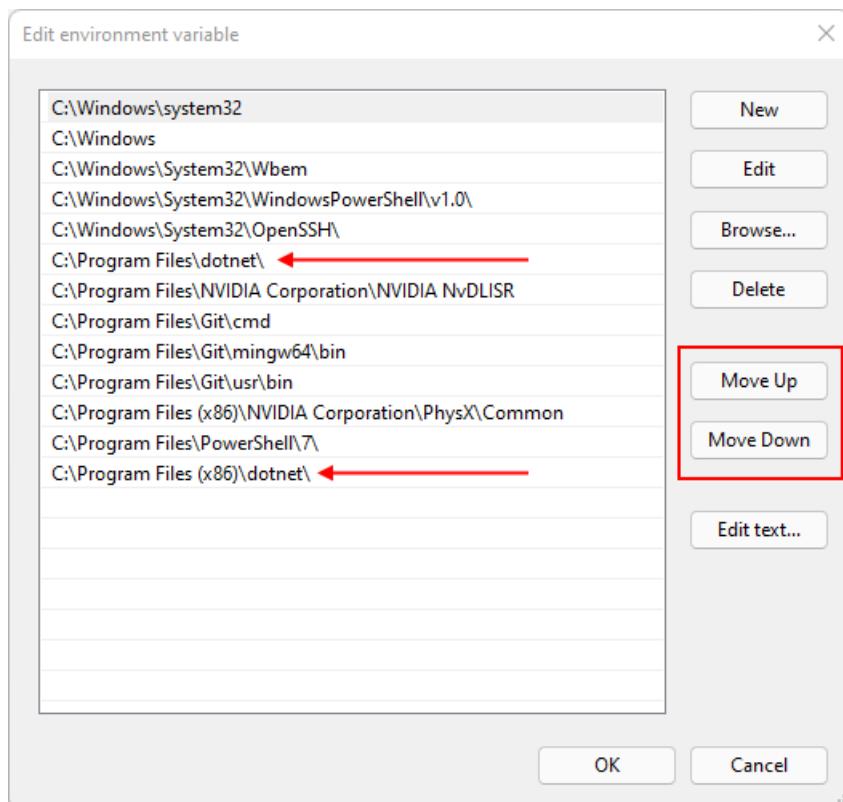
3. The System Properties window opens up to the Advanced Tab. Select Environment Variables.



4. On the Environment Variables window, under the System variables group, select the *Path** row and then select the Edit button.



5. Use the Move Up and Move Down buttons to move the C:\Program Files\dotnet\ entry above C:\Program Files (x86)\dotnet\.



Next steps

- [How to check if .NET is already installed.](#)
- [Tutorial: Hello World tutorial.](#)

- [Tutorial: Create a new app with Visual Studio Code.](#)
- [Tutorial: Containerize a .NET Core app.](#)

Install .NET on macOS

9/20/2022 • 7 minutes to read • [Edit Online](#)

In this article, you'll learn how to install .NET on macOS. .NET is made up of the runtime and the SDK. The runtime is used to run a .NET app and may or may not be included with the app. The SDK is used to create .NET apps and libraries. The .NET runtime is always installed with the SDK.

The latest version of .NET is 6.0.

[DOWNLOAD .NET
CORE](#)

Supported releases

The following table is a list of currently supported .NET releases and the versions of macOS they're supported on. These versions remain supported until the version of .NET reaches [end-of-support](#).

- A ✓ indicates that the version of .NET is still supported.
- A ✗ indicates that the version of .NET isn't supported.

OPERATING SYSTEM	.NET CORE 3.1	.NET 6
macOS 12.0 "Monterey"	✓ 3.1	✓ 6.0
macOS 11.0 "Big Sur"	✓ 3.1	✓ 6.0
macOS 10.15 "Catalina"	✓ 3.1	✓ 6.0

For more information about the life cycle of .NET releases, see [.NET and .NET Core Support Policy](#).

Unsupported releases

The following versions of .NET are ✗ no longer supported:

- .NET 5
- .NET Core 3.0
- .NET Core 2.2
- .NET Core 2.1
- .NET Core 2.0

Runtime information

The runtime is used to run apps created with .NET. When an app author publishes an app, they can include the runtime with their app. If they don't include the runtime, it's up to the user to install the runtime.

There are two different runtimes you can install on macOS:

- *ASP.NET Core runtime*
Runs ASP.NET Core apps. Includes the .NET runtime.
- *.NET runtime*
This runtime is the simplest runtime and doesn't include any other runtime. It's highly recommended that

you install *ASP.NET Core runtime* for the best compatibility with .NET apps.

[DOWNLOAD .NET](#)

[RUNTIME](#)

SDK information

The SDK is used to build and publish .NET apps and libraries. Installing the SDK includes both [runtimes](#): ASP.NET Core and .NET.

Notarization

Beginning with macOS Catalina (version 10.15), all software built after June 1, 2019 that is distributed with Developer ID, must be notarized. This requirement applies to the .NET runtime, .NET SDK, and software created with .NET.

The runtime and SDK installers for .NET have been notarized since February 18, 2020. Prior released versions aren't notarized. If you run a non-notarized app, you'll see an error similar to the following image:



For more information about how enforced-notarization affects .NET (and your .NET apps), see [Working with macOS Catalina Notarization](#).

libgdiplus

.NET applications that use the *System.Drawing.Common* assembly require libgdiplus to be installed.

An easy way to obtain libgdiplus is by using the [Homebrew \("brew"\)](#) package manager for macOS. After installing *brew*, install libgdiplus by executing the following commands at a Terminal (command) prompt:

```
brew update  
brew install mono-libgdiplus
```

Install with an installer

macOS has standalone installers that can be used to install the .NET 6 SDK:

- [x64 and Arm64 CPUs](#)

Download and manually install

As an alternative to the macOS installers for .NET, you can download and manually install the SDK and runtime. Manual installation is usually performed as part of continuous integration testing. For a developer or user, it's generally better to use an [installer](#).

First, download a **binary** release for either the SDK or the runtime from one of the following sites. If you install the .NET SDK, you will not need to install the corresponding runtime:

- ✓ [.NET 6 downloads](#)
- ✓ [.NET Core 3.1 downloads](#)
- [All .NET downloads](#)

Next, extract the downloaded file and use the `export` command to set `DOTNET_ROOT` to the extracted folder's location and then ensure .NET is in PATH. This should make the .NET CLI commands available at the terminal.

Alternatively, after downloading the .NET binary, the following commands may be run from the directory where the file is saved to extract the runtime. This will also make the .NET CLI commands available at the terminal and set the required environment variables. **Remember to change the `DOTNET_FILE` value to the name of the downloaded binary:**

```
DOTNET_FILE=dotnet-sdk-6.0.100-osx-x64.tar.gz
export DOTNET_ROOT=$(pwd)/dotnet

mkdir -p "$DOTNET_ROOT" && tar zxf "$DOTNET_FILE" -C "$DOTNET_ROOT"

export PATH=$PATH:$DOTNET_ROOT
```

TIP

The preceding `export` commands only make the .NET CLI commands available for the terminal session in which it was run.

You can edit your shell profile to permanently add the commands. There are a number of different shells available for Linux and each has a different profile. For example:

- **Bash Shell:** `~/.bash_profile`, `~/.bashrc`
- **Korn Shell:** `~/.kshrc` or `.profile`
- **Z Shell:** `~/.zshrc` or `.zprofile`

Edit the appropriate source file for your shell and add `:$HOME/dotnet` to the end of the existing `PATH` statement. If no `PATH` statement is included, add a new line with `export PATH=$PATH:$HOME/dotnet`.

Also, add `export DOTNET_ROOT=$HOME/dotnet` to the end of the file.

This approach lets you install different versions into separate locations and choose explicitly which one to use by which application.

Arm-based Macs

The following sections describe things you should consider when installing .NET on an Arm-based Mac.

What's supported

The following table describes which versions of .NET are supported on an Arm-based Mac:

.NET VERSION	ARCHITECTURE	SDK	RUNTIME	PATH CONFLICT
6.0	Arm64	Yes	Yes	No
6.0	x64	Yes	Yes	No
3.1	Arm64	No	No	N/A
3.1	x64	No	Yes	Yes

The x64 and Arm64 versions of the .NET 6 SDK exist independently from each other. If a new version is released, each install needs to be upgraded.

Path differences

On an Arm-based Mac, all Arm64 versions of .NET are installed to the normal `/usr/local/share/dotnet`/folder. However, when you install the **x64** version of .NET 6 SDK, it's installed to the `/usr/local/share/dotnet/x64/dotnet`/folder.

Path conflicts

The **x64** .NET 6 SDK installs to its own directory, as described in the previous section. This allows the Arm64 and x64 versions of the .NET 6 SDK to exist on the same machine. However, any **x64** SDK prior to 6.0 isn't supported and installs to the same location as the Arm64 version, the `/usr/local/share/dotnet`/folder. If you want to install an unsupported x64 SDK, you'll need to first uninstall the Arm64 version. The opposite is also true, you'll need to uninstall the unsupported x64 SDK to install the Arm64 version.

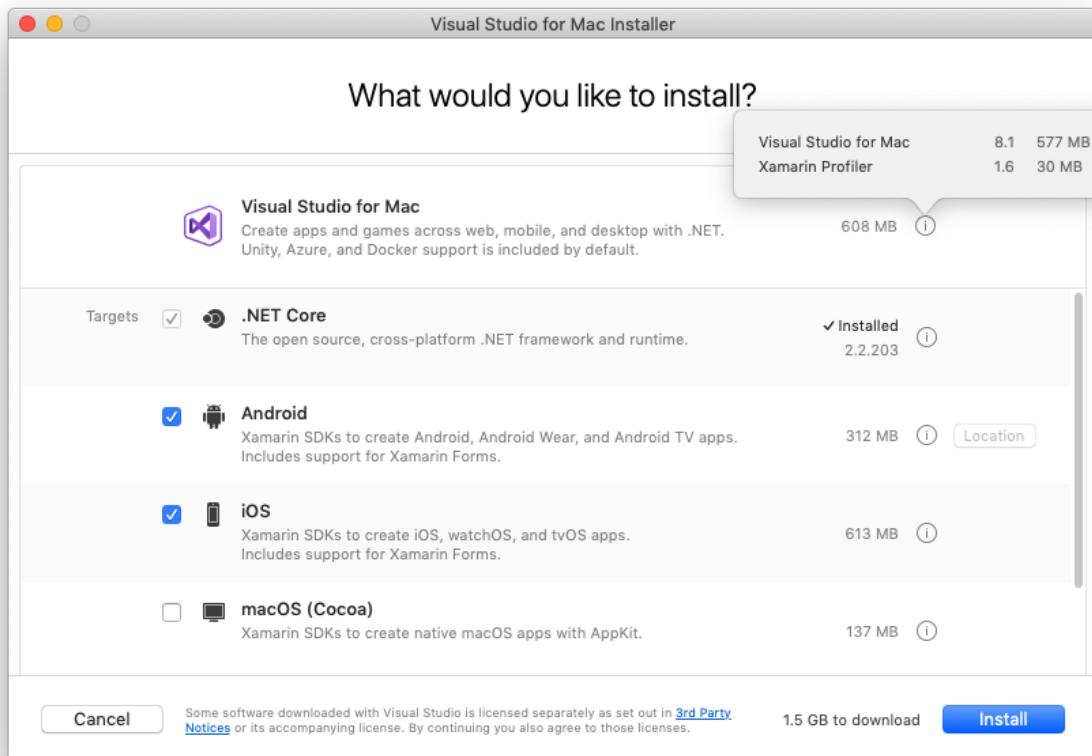
Path variables

Environment variables that add .NET to system path, such as the `PATH` variable, may need to be changed if you have both the x64 and Arm64 versions of the .NET 6 SDK installed. Additionally, some tools rely on the `DOTNET_ROOT` environment variable, which would also need to be updated to point to the appropriate .NET 6 SDK installation folder.

Install with Visual Studio for Mac

Visual Studio for Mac installs the .NET SDK when the **.NET** workload is selected. To get started with .NET development on macOS, see [Install Visual Studio 2019 for Mac](#).

.NET SDK VERSION	VISUAL STUDIO VERSION
6.0	Visual Studio 2022 for Mac Preview 3 17.0 or higher.
3.1	Visual Studio 2019 for Mac version 8.4 or higher.



Install alongside Visual Studio Code

Visual Studio Code is a powerful and lightweight source code editor that runs on your desktop. Visual Studio Code is available for Windows, macOS, and Linux.

While Visual Studio Code doesn't come with an automated .NET installer like Visual Studio does, adding .NET support is simple.

1. [Download and install Visual Studio Code.](#)
2. [Download and install the .NET SDK.](#)
3. [Install the C# extension from the Visual Studio Code marketplace.](#)

Install with bash automation

The [dotnet-install scripts](#) are used for automation and non-admin installs of the runtime. You can download the script from the [dotnet-install script reference page](#).

The script defaults to installing the latest [long term support \(LTS\)](#) version, which is .NET 6.0. You can choose a specific release by specifying the `current` switch. Include the `runtime` switch to install a runtime. Otherwise, the script installs the [SDK](#).

```
./dotnet-install.sh --channel 6.0 --runtime aspnetcore
```

NOTE

The previous command installs the ASP.NET Core runtime for maximum compatibility. The ASP.NET Core runtime also includes the standard .NET runtime.

Docker

Containers provide a lightweight way to isolate your application from the rest of the host system. Containers on the same machine share just the kernel and use resources given to your application.

.NET can run in a Docker container. Official .NET Docker images are published to the Microsoft Container Registry (MCR) and are discoverable at the [Microsoft .NET Docker Hub repository](#). Each repository contains images for different combinations of the .NET (SDK or Runtime) and OS that you can use.

Microsoft provides images that are tailored for specific scenarios. For example, the [ASP.NET Core repository](#) provides images that are built for running ASP.NET Core apps in production.

For more information about using .NET in a Docker container, see [Introduction to .NET and Docker](#) and [Samples](#).

Next steps

- [How to check if .NET is already installed](#).
- [Working with macOS Catalina notarization](#).
- [Tutorial: Get started on macOS](#).
- [Tutorial: Create a new app with Visual Studio Code](#).
- [Tutorial: Containerize a .NET app](#).

Install .NET on Linux

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article details how to install .NET on various Linux distributions, manually and via a package manager. Typically, stable .NET versions are available in a package manager, and Preview versions are not.

Manual installation

You can install .NET manually in the following ways:

- [Download tarballs](#)
- [Scripted install](#)
- [Manual binary extraction](#)

You may need to install [.NET dependencies](#) if you install .NET manually.

Official package archives

.NET is available in the [official package archives](#) for various Linux distributions, including the following ones:

- [Alpine Linux](#)
- [Arch Linux](#)
- [Arch Linux User Repository](#)
- [Fedora](#)
- [Red Hat Enterprise Linux](#)
- [Ubuntu](#)

Microsoft collaborates with partners to ensure .NET works well on their Linux distributions. Support is provided by those distributions. You can still [open issues at dotnet/core](#) if you run into problems.

Microsoft packages

.NET is also available via [packages.microsoft.com](#).

- [CentOS](#)
- [Debian](#)
- [Fedora](#)
- [openSUSE](#)
- [SLES](#)
- [Ubuntu](#)

These packages are [supported by Microsoft](#).

You're encouraged to install .NET from the official archive for your distribution if it's available there, even if it's also available at [packages.microsoft.com](#).

Other distributions

Installation information is also provided for other distributions.

- [Alpine](#)

- [Containers](#)
- [Snap](#)

Next steps

- [How to check if .NET is already installed.](#)
- [Tutorial: Create a new app with Visual Studio Code.](#)
- [Tutorial: Containerize a .NET app.](#)

Install the .NET SDK or the .NET Runtime on Ubuntu

9/20/2022 • 11 minutes to read • [Edit Online](#)

.NET is supported on Ubuntu. This article describes how to install .NET on Ubuntu. When an [Ubuntu version](#) falls out of support, .NET is no longer supported with that version.

Install the SDK (which includes the runtime) if you want to develop .NET apps. Or, if you only need to run apps, install the Runtime. If you're installing the Runtime, we suggest you install the [ASP.NET Core Runtime](#) as it includes both .NET and ASP.NET Core runtimes.

If you've already installed the SDK or Runtime, use the `dotnet --list-sdks` and `dotnet --list-runtimes` commands to see which versions are installed. For more information, see [How to check that .NET is already installed](#).

IMPORTANT

Package manager installs are only supported on the **x64** architecture. Other architectures, such as **Arm**, must install .NET by some other means such as with Snap, an installer script, or through a manual binary installation.

For more information on installing .NET **without a package manager**, see one of the following articles:

- [Alternatively install .NET with Snap.](#)
- [Alternatively install .NET with `install-dotnet` script.](#)
- [Manually install .NET](#)

Supported distributions

The following table is a list of currently supported .NET releases and the versions of Ubuntu they're supported on.

UBUNTU	.NET
22.04 (LTS)	6+
20.04 (LTS)	3.1, 6
18.04 (LTS)	3.1, 6
16.04 (LTS)	3.1, 6

The following versions of .NET are **X** no longer supported:

- .NET 5
- .NET Core 3.0
- .NET Core 2.2
- .NET Core 2.1
- .NET Core 2.0

Install preview versions

Preview and release candidate versions of .NET aren't available in package managers. You can install previews and release candidates of .NET in one of the following ways:

- [Snap package](#)
- [Scripted install with *install-dotnet.sh*](#)
- [Manual binary extraction](#)

Remove preview versions

When using a package manager to manage your installation of .NET, you may run into a conflict if you've previously installed a preview release. The package manager may interpret the non-preview release as an earlier version of .NET. To install the non-preview release, first uninstall the preview versions. For more information about uninstalling .NET, see [How to remove the .NET Runtime and SDK](#).

22.04

WARNING

If you've previously installed .NET 6 from packages.microsoft.com, see the [Advisory on installing .NET 6 on Ubuntu 22.04](#).

.NET 6 is included in the Ubuntu 22.04 package manager feeds.

Install the SDK

The .NET SDK allows you to develop apps with .NET. If you install the .NET SDK, you don't need to install the corresponding runtime. To install the .NET SDK, run the following commands:

```
sudo apt-get update && \
sudo apt-get install -y dotnet
```

Install the runtime

The ASP.NET Core Runtime allows you to run apps that were made with .NET that didn't provide the runtime. The following commands install the ASP.NET Core Runtime, which is the most compatible runtime for .NET. In your terminal, run the following commands:

```
sudo apt-get update && \
sudo apt-get install -y aspnetcore-runtime-6.0
```

As an alternative to the ASP.NET Core Runtime, you can install the .NET Runtime, which doesn't include ASP.NET Core support: replace `aspnetcore-runtime-6.0` in the previous command with `dotnet-runtime-6.0`:

```
sudo apt-get install -y dotnet-runtime-6.0
```

22.04 (Microsoft package feed)

NOTE

Warning: .NET 6 is included in Ubuntu 22.04. See the [Advisory on installing .NET 6 on Ubuntu 22.04](#) if you want to use .NET packages from packages.microsoft.com.

Installing with APT can be done with a few commands. Before you install .NET, run the following commands to add the Microsoft package signing key to your list of trusted keys and add the package repository.

Open a terminal and run the following commands:

```
wget https://packages.microsoft.com/config/ubuntu/22.04/packages-microsoft-prod.deb -O packages-microsoft-prod.deb  
sudo dpkg -i packages-microsoft-prod.deb  
rm packages-microsoft-prod.deb
```

Install the SDK

The .NET SDK allows you to develop apps with .NET. If you install the .NET SDK, you don't need to install the corresponding runtime. To install the .NET SDK, run the following commands:

```
sudo apt-get update && \  
sudo apt-get install -y dotnet-sdk-6.0
```

IMPORTANT

If you receive an error message similar to **Unable to locate package dotnet-sdk-6.0**, see the [APT troubleshooting](#) section.

Install the runtime

The ASP.NET Core Runtime allows you to run apps that were made with .NET that didn't provide the runtime. The following commands install the ASP.NET Core Runtime, which is the most compatible runtime for .NET. In your terminal, run the following commands:

```
sudo apt-get update && \  
sudo apt-get install -y aspnetcore-runtime-6.0
```

IMPORTANT

If you receive an error message similar to **Unable to locate package aspnetcore-runtime-6.0**, see the [APT troubleshooting](#) section.

As an alternative to the ASP.NET Core Runtime, you can install the .NET Runtime, which doesn't include ASP.NET Core support: replace `aspnetcore-runtime-6.0` in the previous command with `dotnet-runtime-6.0`:

```
sudo apt-get install -y dotnet-runtime-6.0
```

NOTE

Ubuntu 22.04 includes OpenSSL 3 as the baseline version. .NET 6 supports OpenSSL 3 while earlier .NET versions don't. Microsoft doesn't test or support using OpenSSL 1.x on Ubuntu 22.04. For more information, see [.NET 6 Security Improvements](#).

20.04

Installing with APT can be done with a few commands. Before you install .NET, run the following commands to add the Microsoft package signing key to your list of trusted keys and add the package repository.

Open a terminal and run the following commands:

```
wget https://packages.microsoft.com/config/ubuntu/20.04/packages-microsoft-prod.deb -O packages-microsoft-prod.deb  
sudo dpkg -i packages-microsoft-prod.deb  
rm packages-microsoft-prod.deb
```

Install the SDK

The .NET SDK allows you to develop apps with .NET. If you install the .NET SDK, you don't need to install the corresponding runtime. To install the .NET SDK, run the following commands:

```
sudo apt-get update && \  
sudo apt-get install -y dotnet-sdk-6.0
```

IMPORTANT

If you receive an error message similar to **Unable to locate package dotnet-sdk-6.0**, see the [APT troubleshooting](#) section.

Install the runtime

The ASP.NET Core Runtime allows you to run apps that were made with .NET that didn't provide the runtime. The following commands install the ASP.NET Core Runtime, which is the most compatible runtime for .NET. In your terminal, run the following commands:

```
sudo apt-get update && \  
sudo apt-get install -y aspnetcore-runtime-6.0
```

IMPORTANT

If you receive an error message similar to **Unable to locate package aspnetcore-runtime-6.0**, see the [APT troubleshooting](#) section.

As an alternative to the ASP.NET Core Runtime, you can install the .NET Runtime, which doesn't include ASP.NET Core support: replace `aspnetcore-runtime-6.0` in the previous command with `dotnet-runtime-6.0`:

```
sudo apt-get install -y dotnet-runtime-6.0
```

18.04

Installing with APT can be done with a few commands. Before you install .NET, run the following commands to add the Microsoft package signing key to your list of trusted keys and add the package repository.

Open a terminal and run the following commands:

```
wget https://packages.microsoft.com/config/ubuntu/18.04/packages-microsoft-prod.deb -O packages-microsoft-prod.deb  
sudo dpkg -i packages-microsoft-prod.deb  
rm packages-microsoft-prod.deb
```

Install the SDK

The .NET SDK allows you to develop apps with .NET. If you install the .NET SDK, you don't need to install the corresponding runtime. To install the .NET SDK, run the following commands:

```
sudo apt-get update && \  
sudo apt-get install -y dotnet-sdk-6.0
```

IMPORTANT

If you receive an error message similar to **Unable to locate package dotnet-sdk-6.0**, see the [APT troubleshooting](#) section.

Install the runtime

The ASP.NET Core Runtime allows you to run apps that were made with .NET that didn't provide the runtime. The following commands install the ASP.NET Core Runtime, which is the most compatible runtime for .NET. In your terminal, run the following commands:

```
sudo apt-get update && \  
sudo apt-get install -y aspnetcore-runtime-6.0
```

IMPORTANT

If you receive an error message similar to **Unable to locate package aspnetcore-runtime-6.0**, see the [APT troubleshooting](#) section.

As an alternative to the ASP.NET Core Runtime, you can install the .NET Runtime, which doesn't include ASP.NET Core support: replace `aspnetcore-runtime-6.0` in the previous command with `dotnet-runtime-6.0`:

```
sudo apt-get install -y dotnet-runtime-6.0
```

16.04

Installing with APT can be done with a few commands. Before you install .NET, run the following commands to add the Microsoft package signing key to your list of trusted keys and add the package repository.

Open a terminal and run the following commands:

```
wget https://packages.microsoft.com/config/ubuntu/16.04/packages-microsoft-prod.deb -O packages-microsoft-prod.deb  
sudo dpkg -i packages-microsoft-prod.deb  
rm packages-microsoft-prod.deb
```

Install the SDK

The .NET SDK allows you to develop apps with .NET. If you install the .NET SDK, you don't need to install the corresponding runtime. To install the .NET SDK, run the following commands:

```
sudo apt-get update && \  
sudo apt-get install -y dotnet-sdk-6.0
```

IMPORTANT

If you receive an error message similar to **Unable to locate package dotnet-sdk-6.0**, see the [APT troubleshooting](#) section.

Install the runtime

The ASP.NET Core Runtime allows you to run apps that were made with .NET that didn't provide the runtime. The following commands install the ASP.NET Core Runtime, which is the most compatible runtime for .NET. In your terminal, run the following commands:

```
sudo apt-get update && \  
sudo apt-get install -y aspnetcore-runtime-6.0
```

IMPORTANT

If you receive an error message similar to **Unable to locate package aspnetcore-runtime-6.0**, see the [APT troubleshooting](#) section.

As an alternative to the ASP.NET Core Runtime, you can install the .NET Runtime, which doesn't include ASP.NET Core support: replace `aspnetcore-runtime-6.0` in the previous command with `dotnet-runtime-6.0`:

```
sudo apt-get install -y dotnet-runtime-6.0
```

How to install other versions

All versions of .NET are available for download at <https://dotnet.microsoft.com/download/dotnet>, but require [manual installation](#). You can try and use the package manager to install a different version of .NET. However, the requested version may not be available.

The packages added to package manager feeds are named in a hackable format, for example:

```
{product}-{type}-{version} .
```

- **product**

The type of .NET product to install. Valid options are:

- dotnet
- aspnetcore

- **type**

Chooses the SDK or the runtime. Valid options are:

- sdk
- runtime

- **version**

The version of the SDK or runtime to install. This article will always give the instructions for the latest supported version. Valid options are any released version, such as:

- 5.0
- 3.1
- 3.0
- 2.1

It's possible the SDK/runtime you're trying to download is not available for your Linux distribution. For a list of supported distributions, see [Install .NET on Linux](#).

Examples

- Install the ASP.NET Core 5.0 runtime: `aspnetcore-runtime-5.0`
- Install the .NET Core 2.1 runtime: `dotnet-runtime-2.1`
- Install the .NET 5 SDK: `dotnet-sdk-5.0`
- Install the .NET Core 3.1 SDK: `dotnet-sdk-3.1`

Package missing

If the package-version combination doesn't work, it's not available. For example, there isn't an ASP.NET Core SDK, the SDK components are included with the .NET SDK. The value `aspnetcore-sdk-2.2` is incorrect and should be `dotnet-sdk-2.2`. For a list of Linux distributions supported by .NET, see [.NET dependencies and requirements](#).

Use APT to update .NET

When a new patch release is available for .NET, you can simply upgrade it through APT with the following commands:

```
sudo apt-get update  
sudo apt-get upgrade
```

If you've upgraded your Linux distribution since installing .NET, you may need to reconfigure the Microsoft package repository. Run the installation instructions for your current distribution version to upgrade to the appropriate package repository for .NET updates.

APT troubleshooting

This section provides information on common errors you may get while using APT to install .NET.

Unable to find package

IMPORTANT

Package manager installs are only supported on the **x64** architecture. Other architectures, such as **Arm**, must install .NET by some other means such as with Snap, an installer script, or through a manual binary installation.

For more information on installing .NET **without a package manager**, see one of the following articles:

- [Alternatively install .NET with Snap](#).
- [Alternatively install .NET with `install-dotnet` script](#).

- Manually install .NET

Unable to locate \ Some packages could not be installed

NOTE

This information only applies when .NET is installed from the Microsoft package feed.

If you receive an error message similar to **Unable to locate package {dotnet-package}** or **Some packages could not be installed**, run the following commands.

There are two placeholders in the following set of commands.

- `{dotnet-package}`

This represents the .NET package you're installing, such as `aspnetcore-runtime-3.1`. This is used in the following `sudo apt-get install` command.

- `{os-version}`

This represents the distribution version you're on. This is used in the `wget` command below. The distribution version is the numerical value, such as `20.04` on Ubuntu or `10` on Debian.

First, try purging the package list:

```
sudo dpkg --purge packages-microsoft-prod && sudo dpkg -i packages-microsoft-prod.deb  
sudo apt-get update
```

Then, try to install .NET again. If that doesn't work, you can run a manual install with the following commands:

```
sudo apt-get install -y gpg  
wget -O - https://packages.microsoft.com/keys/microsoft.asc | gpg --dearmor -o microsoft.asc.gpg  
sudo mv microsoft.asc.gpg /etc/apt/trusted.gpg.d/  
wget https://packages.microsoft.com/config/ubuntu/{os-version}/prod.list  
sudo mv prod.list /etc/apt/sources.list.d/microsoft-prod.list  
sudo chown root:root /etc/apt/trusted.gpg.d/microsoft.asc.gpg  
sudo chown root:root /etc/apt/sources.list.d/microsoft-prod.list  
sudo apt-get update && \  
sudo apt-get install -y {dotnet-package}
```

Failed to fetch

While installing the .NET package, you may see an error similar to

`Failed to fetch ... File has unexpected size ... Mirror sync in progress?`. This error could mean that the package feed for .NET is being upgraded with newer package versions, and that you should try again later. During an upgrade, the package feed shouldn't be unavailable for more than 30 minutes. If you continually receive this error for more than 30 minutes, please file an issue at <https://github.com/dotnet/core/issues>.

Dependencies

When you install with a package manager, these libraries are installed for you. But, if you manually install .NET or you publish a self-contained app, you'll need to make sure these libraries are installed:

- libc6
- libgcc1
- libgssapi-krb5-2
- libicu52 (for 14.x)
- libicu55 (for 16.x)

- libicu60 (for 18.x)
- libicu66 (for 20.x)
- libssl1.0.0 (for 14.x, 16.x)
- libssl1.1 (for 18.x, 20.x)
- libstdc++6
- zlib1g

For .NET apps that use the *System.Drawing.Common* assembly, you also need the following dependency:

- libgdiplus (version 6.0.1 or later)

WARNING

You can install a recent version of *libgdiplus* by adding the Mono repository to your system. For more information, see <https://www.mono-project.com/download/stable/>.

Next steps

- [How to enable TAB completion for the .NET CLI](#)
- [Tutorial: Create a console application with .NET SDK using Visual Studio Code](#)

Install the .NET SDK or the .NET Runtime on Alpine

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article describes how to install .NET on Alpine. When an Alpine version falls out of support, .NET is no longer supported with that version. However, these instructions may help you to get .NET running on those versions, even though it isn't supported.

Install the SDK (which includes the runtime) if you want to develop .NET apps. Or, if you only need to run apps, install the Runtime. If you're installing the Runtime, we suggest you install the **ASP.NET Core Runtime** as it includes both .NET and ASP.NET Core runtimes.

If you've already installed the SDK or Runtime, use the `dotnet --list-sdks` and `dotnet --list-runtimes` commands to see which versions are installed. For more information, see [How to check that .NET is already installed](#).

Install

Installers aren't available for Alpine Linux. You must install .NET in one of the following ways:

- Scripted install with [*install-dotnet.sh*](#)
- Manual binary extraction

Supported distributions

The following table is a list of currently supported .NET releases and the versions of Alpine they're supported on. These versions remain supported until either the version of [.NET reaches end-of-support](#) or the version of [Alpine reaches end-of-life](#).

- A ✓ indicates that the version of Alpine or .NET is still supported.
- A ✗ indicates that the version of Alpine or .NET isn't supported on that Alpine release.
- When both a version of Alpine and a version of .NET have ✓, that OS and .NET combination is supported.

ALPINE	.NET CORE 3.1	.NET 6
✓ 3.15	✓ 3.1	✓ 6.0
✓ 3.14	✓ 3.1	✓ 6.0
✓ 3.13	✓ 3.1	✓ 6.0
✓ 3.12	✓ 3.1	✓ 6.0
✗ 3.11	✓ 3.1	✗ 6.0
✗ 3.10	✓ 3.1	✗ 6.0
✗ 3.9	✓ 3.1	✗ 6.0
✗ 3.8	✓ 3.1	✗ 6.0

The following versions of .NET are ✗ no longer supported:

- .NET 5
- .NET Core 3.0
- .NET Core 2.2
- .NET Core 2.1
- .NET Core 2.0

Dependencies

.NET on Alpine Linux requires the following dependencies installed:

- icu-libs
- krb5-libs
- libgcc
- libgdiplus (if the .NET app requires the *System.Drawing.Common* assembly)
- libintl
- libssl1.1 (Alpine v3.9 or greater)
- libssl1.0 (Alpine v3.8 or lower)
- libstdc++
- zlib

To install the needed requirements, run the following command:

```
apk add bash icu-libs krb5-libs libgcc libintl libssl1.1 libstdc++ zlib
```

To install **libgdiplus**, you may need to specify a repository:

```
apk add libgdiplus --repository https://dl-3.alpinelinux.org/alpine/edge/testing/
```

Next steps

- [How to enable TAB completion for the .NET CLI](#)
- [Tutorial: Create a console application with .NET SDK using Visual Studio Code](#)

Install the .NET SDK or the .NET Runtime on CentOS

9/20/2022 • 7 minutes to read • [Edit Online](#)

.NET is supported on CentOS. This article describes how to install .NET on CentOS. If you need to install .NET On CentOS Stream, see [Install the .NET SDK or the .NET Runtime on RHEL and CentOS Stream](#).

Install the SDK (which includes the runtime) if you want to develop .NET apps. Or, if you only need to run apps, install the Runtime. If you're installing the Runtime, we suggest you install the **ASP.NET Core Runtime** as it includes both .NET and ASP.NET Core runtimes.

If you've already installed the SDK or Runtime, use the `dotnet --list-sdks` and `dotnet --list-runtimes` commands to see which versions are installed. For more information, see [How to check that .NET is already installed](#).

IMPORTANT

Package manager installs are only supported on the **x64** architecture. Other architectures, such as **Arm**, must install .NET by some other means such as with Snap, an installer script, or through a manual binary installation.

For more information on installing .NET **without a package manager**, see one of the following articles:

- [Alternatively install .NET with Snap](#).
- [Alternatively install .NET with `install-dotnet` script](#).
- [Manually install .NET](#)

Supported distributions

The following table is a list of currently supported .NET releases on both CentOS 7 and CentOS 8. These versions remain supported until either the version of [.NET reaches end-of-support](#) or the version of CentOS is no longer supported.

- A ✓ indicates that the version of CentOS or .NET is still supported.
- A ✗ indicates that the version of CentOS or .NET isn't supported on that CentOS release.
- When both a version of CentOS and a version of .NET have ✓, that OS and .NET combination is supported.

CENTOS	.NET CORE 3.1	.NET 6
✓ 7	✓ 3.1	✓ 6.0
✗ 8*	✓ 3.1	✗ 6.0

WARNING

*CentOS 8 reached an early End Of Life (EOL) on December 31st, 2021. For more information, see the official [CentOS Linux EOL page](#). Because of this, .NET 6 won't be supported on CentOS Linux 8.

The following versions of .NET are ✗ no longer supported:

- .NET 5
- .NET Core 3.0
- .NET Core 2.2
- .NET Core 2.1
- .NET Core 2.0

IMPORTANT

Package manager installs are only supported on the **x64** architecture. Other architectures, such as **Arm**, must install .NET by some other means such as with Snap, an installer script, or through a manual binary installation.

For more information on installing .NET **without a package manager**, see one of the following articles:

- [Alternatively install .NET with Snap.](#)
- [Alternatively install .NET with `install-dotnet` script.](#)
- [Manually install .NET](#)

Install preview versions

Preview and release candidate versions of .NET aren't available in package managers. You can install previews and release candidates of .NET in one of the following ways:

- [Snap package](#)
- [Scripted install with `install-dotnet.sh`](#)
- [Manual binary extraction](#)

Remove preview versions

When using a package manager to manage your installation of .NET, you may run into a conflict if you've previously installed a preview release. The package manager may interpret the non-preview release as an earlier version of .NET. To install the non-preview release, first uninstall the preview versions. For more information about uninstalling .NET, see [How to remove the .NET Runtime and SDK](#).

CentOS 7 ✓

Before you install .NET, run the following commands to add the Microsoft package signing key to your list of trusted keys and add the Microsoft package repository. Open a terminal and run the following commands:

```
sudo rpm -Uvh https://packages.microsoft.com/config/centos/7/packages-microsoft-prod.rpm
```

Install the SDK

The .NET SDK allows you to develop apps with .NET. If you install the .NET SDK, you don't need to install the corresponding runtime. To install the .NET SDK, run the following command:

```
sudo yum install dotnet-sdk-6.0
```

Install the runtime

The ASP.NET Core Runtime allows you to run apps that were made with .NET that didn't provide the runtime. The following command installs the ASP.NET Core Runtime, which is the most compatible runtime for .NET. In your terminal, run the following command:

```
sudo yum install aspnetcore-runtime-6.0
```

As an alternative to the ASP.NET Core Runtime, you can install the .NET Runtime, which doesn't include ASP.NET Core support: replace `aspnetcore-runtime-6.0` in the previous command with `dotnet-runtime-6.0`:

```
sudo yum install dotnet-runtime-6.0
```

CentOS 8 ✓

WARNING

*CentOS 8 will reach an early End Of Life (EOL) on December 31st, 2021. For more information, see the official [CentOS Linux EOL page](#). Because of this, .NET 6 won't be supported on CentOS Linux 8.

.NET 5 is available in the default package repositories for CentOS 8.

Install the SDK

The .NET SDK allows you to develop apps with .NET. If you install the .NET SDK, you don't need to install the corresponding runtime. To install the .NET SDK, run the following command:

```
sudo dnf install dotnet-sdk-5.0
```

Install the runtime

The ASP.NET Core Runtime allows you to run apps that were made with .NET that didn't provide the runtime. The following command install the ASP.NET Core Runtime, which is the most compatible runtime for .NET. In your terminal, run the following command:

```
sudo dnf install aspnetcore-runtime-5.0
```

As an alternative to the ASP.NET Core Runtime, you can install the .NET Runtime, which doesn't include ASP.NET Core support: replace `aspnetcore-runtime-5.0` in the previous command with `dotnet-runtime-5.0`:

```
sudo dnf install dotnet-runtime-5.0
```

How to install other versions

All versions of .NET are available for download at <https://dotnet.microsoft.com/download/dotnet>, but require [manual installation](#). You can try and use the package manager to install a different version of .NET. However, the requested version may not be available.

The packages added to package manager feeds are named in a hackable format, for example:

```
{product}-{type}-{version} .
```

- **product**

The type of .NET product to install. Valid options are:

- dotnet
- aspnetcore

- **type**

Chooses the SDK or the runtime. Valid options are:

- sdk
- runtime

- **version**

The version of the SDK or runtime to install. This article will always give the instructions for the latest supported version. Valid options are any released version, such as:

- 5.0
- 3.1
- 3.0
- 2.1

It's possible the SDK/runtime you're trying to download is not available for your Linux distribution. For a list of supported distributions, see [Install .NET on Linux](#).

Examples

- Install the ASP.NET Core 5.0 runtime: `aspnetcore-runtime-5.0`
- Install the .NET Core 2.1 runtime: `dotnet-runtime-2.1`
- Install the .NET 5 SDK: `dotnet-sdk-5.0`
- Install the .NET Core 3.1 SDK: `dotnet-sdk-3.1`

Package missing

If the package-version combination doesn't work, it's not available. For example, there isn't an ASP.NET Core SDK, the SDK components are included with the .NET SDK. The value `aspnetcore-sdk-2.2` is incorrect and should be `dotnet-sdk-2.2`. For a list of Linux distributions supported by .NET, see [.NET dependencies and requirements](#).

Troubleshoot the package manager

This section provides information on common errors you may get while using the package manager to install .NET.

Unable to find package

IMPORTANT

Package manager installs are only supported on the `x64` architecture. Other architectures, such as `Arm`, must install .NET by some other means such as with Snap, an installer script, or through a manual binary installation.

For more information on installing .NET without a package manager, see one of the following articles:

- [Alternatively install .NET with Snap](#).
- [Alternatively install .NET with `install-dotnet` script](#).
- [Manually install .NET](#)

Failed to fetch

While installing the .NET package, you may see an error similar to

`signature verification failed for file 'repomd.xml' from repository 'packages-microsoft-com-prod'`. Generally speaking, this error means that the package feed for .NET is being upgraded with newer package versions, and that you should try again later. During an upgrade, the package feed should not be unavailable for more than 2 hours. If you continually receive this error for more than 2 hours, please file an issue at <https://github.com/dotnet/core/issues>.

Errors related to missing `fxr`, `libhostfxr.so`, or `FrameworkList.xml`

For more information about solving these problems, see [Troubleshoot `fxr`](#), [libhostfxr.so](#), and [FrameworkList.xml errors](#).

Dependencies

When you install with a package manager, these libraries are installed for you. But, if you manually install .NET Core or you publish a self-contained app, you'll need to make sure these libraries are installed:

- krb5-libs
- libicu
- openssl-libs
- zlib

If the target runtime environment's OpenSSL version is 1.1 or newer, you'll need to install `compat-openssl10`.

For more information about the dependencies, see [Self-contained Linux apps](#).

For .NET Core apps that use the `System.Drawing.Common` assembly, you'll also need the following dependency:

- `libgdiplus` (version 6.0.1 or later)

WARNING

You can install a recent version of `libgdiplus` by adding the Mono repository to your system. For more information, see <https://www.mono-project.com/download/stable/>.

Next steps

- [How to enable TAB completion for the .NET CLI](#)
- [Tutorial: Create a console application with .NET SDK using Visual Studio Code](#)

Install the .NET SDK or the .NET Runtime on RHEL and CentOS Stream

9/20/2022 • 8 minutes to read • [Edit Online](#)

.NET is supported on Red Hat Enterprise Linux (RHEL). This article describes how to install .NET on RHEL and CentOS Stream.

Install the SDK (which includes the runtime) if you want to develop .NET apps. Or, if you only need to run apps, install the Runtime. If you're installing the Runtime, we suggest you install the **ASP.NET Core Runtime** as it includes both .NET and ASP.NET Core runtimes.

If you've already installed the SDK or Runtime, use the `dotnet --list-sdks` and `dotnet --list-runtimes` commands to see which versions are installed. For more information, see [How to check that .NET is already installed](#).

Register your Red Hat subscription

To install .NET from Red Hat on RHEL, you first need to register using the Red Hat Subscription Manager. If this hasn't been done on your system, or if you're unsure, see the [Red Hat Product Documentation for .NET](#).

IMPORTANT

This doesn't apply to CentOS Stream.

Supported distributions

The following table is a list of currently supported .NET releases on both RHEL and CentOS Stream. These versions remain supported until either the version of [.NET reaches end-of-support](#) or the Linux distribution is no longer supported.

- A ✓ indicates that the version of RHEL or .NET is still supported.
- A ✗ indicates that the version of RHEL or .NET isn't supported on that RHEL release.
- When both a version of Linux distribution and a version of .NET have ✓, that OS and .NET combination is supported.

DISTRIBUTION	.NET CORE 3.1	.NET 6
✓ RHEL 8	✓ 3.1	✓ 6.0
✓ RHEL 7	✓ 3.1	✓ 6.0
✓ CentOS Stream 9	✗ 3.1	✓ 6.0
✓ CentOS Stream 8	✓ 3.1	✓ 6.0

The following versions of .NET are ✗ no longer supported:

- .NET 5
- .NET Core 3.0

- .NET Core 2.2
- .NET Core 2.1
- .NET Core 2.0

Install preview versions

Preview and release candidate versions of .NET aren't available in package managers. You can install previews and release candidates of .NET in one of the following ways:

- [Snap package](#)
- [Scripted install with *install-dotnet.sh*](#)
- [Manual binary extraction](#)

Remove preview versions

When using a package manager to manage your installation of .NET, you may run into a conflict if you've previously installed a preview release. The package manager may interpret the non-preview release as an earlier version of .NET. To install the non-preview release, first uninstall the preview versions. For more information about uninstalling .NET, see [How to remove the .NET Runtime and SDK](#).

RHEL 8 ✓

.NET is included in the AppStream repositories for RHEL 8.

Install the SDK

The .NET SDK allows you to develop apps with .NET. If you install the .NET SDK, you don't need to install the corresponding runtime. To install the .NET SDK, run the following command:

```
sudo dnf install dotnet-sdk-6.0
```

Install the runtime

The ASP.NET Core Runtime allows you to run apps that were made with .NET that didn't provide the runtime. The following command install the ASP.NET Core Runtime, which is the most compatible runtime for .NET. In your terminal, run the following command:

```
sudo dnf install aspnetcore-runtime-6.0
```

As an alternative to the ASP.NET Core Runtime, you can install the .NET Runtime, which doesn't include ASP.NET Core support: replace `aspnetcore-runtime-6.0` in the previous command with `dotnet-runtime-6.0`:

```
sudo dnf install dotnet-runtime-6.0
```

RHEL 7 ✓ .NET 6.0

The following command installs the `scl-utils` package:

```
sudo yum install scl-utils
```

Install the SDK

The .NET SDK allows you to develop apps with .NET. If you install the .NET SDK, you don't need to install the

corresponding runtime. To install .NET SDK, run the following commands:

```
subscription-manager repos --enable=rhel-7-server-dotnet-rpms  
yum install rh-dotnet60 -y  
scl enable rh-dotnet60 bash
```

Red Hat does not recommend permanently enabling `rh-dotnet60` because it may affect other programs. If you want to enable `rh-dotnet` permanently, add the following line to your `~/.bashrc` file.

```
source scl_source enable rh-dotnet60
```

Install the runtime

The .NET Runtime allows you to run apps that were made with .NET that didn't include the runtime. The commands below install the ASP.NET Core Runtime, which is the most compatible runtime for .NET Core. In your terminal, run the following commands.

```
subscription-manager repos --enable=rhel-7-server-dotnet-rpms  
yum install rh-dotnet60-aspnetcore-runtime-6.0 -y  
scl enable rh-dotnet60 bash
```

Red Hat does not recommend permanently enabling `rh-dotnet60` because it may affect other programs. If you want to enable `rh-dotnet60` permanently, add the following line to your `~/.bashrc` file.

```
source scl_source enable rh-dotnet60
```

As an alternative to the ASP.NET Core Runtime, you can install the .NET Runtime that doesn't include ASP.NET Core support: replace `rh-dotnet60-aspnetcore-runtime-6.0` in the preceding command with `rh-dotnet60-dotnet-runtime-6.0`.

RHEL 7 ✓ .NET 5.0

The following command installs the `scl-utils` package:

```
sudo yum install scl-utils
```

Install the SDK

The .NET SDK allows you to develop apps with .NET. If you install the .NET SDK, you don't need to install the corresponding runtime. To install .NET SDK, run the following commands:

```
subscription-manager repos --enable=rhel-7-server-dotnet-rpms  
yum install rh-dotnet50 -y  
scl enable rh-dotnet50 bash
```

Red Hat does not recommend permanently enabling `rh-dotnet50` because it may affect other programs. If you want to enable `rh-dotnet` permanently, add the following line to your `~/.bashrc` file.

```
source scl_source enable rh-dotnet50
```

Install the runtime

The .NET Runtime allows you to run apps that were made with .NET that didn't include the runtime. The

commands below install the ASP.NET Core Runtime, which is the most compatible runtime for .NET Core. In your terminal, run the following commands.

```
subscription-manager repos --enable=rhel-7-server-dotnet-rpms
yum install rh-dotnet50-aspnetcore-runtime-5.0 -y
scl enable rh-dotnet50 bash
```

Red Hat does not recommend permanently enabling `rh-dotnet50` because it may affect other programs. If you want to enable `rh-dotnet50` permanently, add the following line to your `~/.bashrc` file.

```
source scl_source enable rh-dotnet50
```

As an alternative to the ASP.NET Core Runtime, you can install the .NET Runtime that doesn't include ASP.NET Core support: replace `rh-dotnet50-aspnetcore-runtime-5.0` in the commands above with `rh-dotnet50-dotnet-runtime-5.0`.

RHEL 7 ✓ .NET Core 3.1

Before you install .NET, run the following commands to add the Microsoft package signing key to your list of trusted keys and add the Microsoft package repository. Open a terminal and run the following commands:

The following command installs the `scl-utils` package:

```
sudo yum install scl-utils
```

Install the SDK

.NET SDK allows you to develop apps with .NET Core. If you install .NET SDK, you don't need to install the corresponding runtime. To install .NET SDK, run the following commands:

```
subscription-manager repos --enable=rhel-7-server-dotnet-rpms
yum install rh-dotnet31 -y
scl enable rh-dotnet31 bash
```

Red Hat does not recommend permanently enabling `rh-dotnet31` because it may affect other programs. For example, `rh-dotnet31` includes a version of `libcurl` that differs from the base RHEL version. This may lead to issues in programs that do not expect a different version of `libcurl`. If you want to enable `rh-dotnet` permanently, add the following line to your `~/.bashrc` file.

```
source scl_source enable rh-dotnet31
```

Install the runtime

The .NET Core Runtime allows you to run apps that were made with .NET Core that didn't include the runtime. The commands below install the ASP.NET Core Runtime, which is the most compatible runtime for .NET Core. In your terminal, run the following commands.

```
subscription-manager repos --enable=rhel-7-server-dotnet-rpms
yum install rh-dotnet31-aspnetcore-runtime-3.1 -y
scl enable rh-dotnet31 bash
```

Red Hat does not recommend permanently enabling `rh-dotnet31` because it may affect other programs. For example, `rh-dotnet31` includes a version of `libcurl` that differs from the base RHEL version. This may lead to

issues in programs that do not expect a different version of `libcurl`. If you want to enable `rh-dotnet31` permanently, add the following line to your `~/.bashrc` file.

```
source scl_source enable rh-dotnet31
```

As an alternative to the ASP.NET Core Runtime, you can install the .NET Core Runtime that doesn't include ASP.NET Core support: replace `rh-dotnet31-aspnetcore-runtime-3.1` in the commands above with `rh-dotnet31-dotnet-runtime-3.1`.

CentOS Stream 9 ✓

.NET is included in the AppStream repositories for CentOS Stream 9. However, .NET Core 3.1 and .NET 5 have been removed from CentOS Stream 9 and you should use .NET 6. For more information, see the blog post [Using .NET with OpenSSL in CentOS Stream 9 | Omair Majid](#).

Install the SDK

The .NET SDK allows you to develop apps with .NET. If you install the .NET SDK, you don't need to install the corresponding runtime. To install the .NET SDK, run the following command:

```
sudo dnf install dotnet-sdk-6.0
```

Install the runtime

The ASP.NET Core Runtime allows you to run apps that were made with .NET that didn't provide the runtime. The following command install the ASP.NET Core Runtime, which is the most compatible runtime for .NET. In your terminal, run the following command:

```
sudo dnf install aspnetcore-runtime-6.0
```

As an alternative to the ASP.NET Core Runtime, you can install the .NET Runtime, which doesn't include ASP.NET Core support: replace `aspnetcore-runtime-6.0` in the previous command with `dotnet-runtime-6.0`:

```
sudo dnf install dotnet-runtime-6.0
```

CentOS Stream 8 ✓

.NET is included in the AppStream repositories for CentOS Stream 8.

Install the SDK

The .NET SDK allows you to develop apps with .NET. If you install the .NET SDK, you don't need to install the corresponding runtime. To install the .NET SDK, run the following command:

```
sudo dnf install dotnet-sdk-6.0
```

Install the runtime

The ASP.NET Core Runtime allows you to run apps that were made with .NET that didn't provide the runtime. The following command install the ASP.NET Core Runtime, which is the most compatible runtime for .NET. In your terminal, run the following command:

```
sudo dnf install aspnetcore-runtime-6.0
```

As an alternative to the ASP.NET Core Runtime, you can install the .NET Runtime, which doesn't include ASP.NET Core support: replace `aspnetcore-runtime-6.0` in the previous command with `dotnet-runtime-6.0`:

```
sudo dnf install dotnet-runtime-6.0
```

Dependencies

When you install with a package manager, these libraries are installed for you. But, if you manually install .NET Core or you publish a self-contained app, you'll need to make sure these libraries are installed:

- `krb5-libs`
- `libicu`
- `openssl-libs`
- `zlib`

If the target runtime environment's OpenSSL version is 1.1 or newer, you'll need to install `compat-openssl10`.

For more information about the dependencies, see [Self-contained Linux apps](#).

For .NET Core apps that use the `System.Drawing.Common` assembly, you'll also need the following dependency:

- [libgdiplus \(version 6.0.1 or later\)](#)

WARNING

You can install a recent version of `libgdiplus` by adding the Mono repository to your system. For more information, see <https://www.mono-project.com/download/stable/>.

How to install other versions

Consult the [Red Hat documentation for .NET](#) on the steps required to install other releases of .NET.

Troubleshoot the package manager

This section provides information on common errors you may get while using the package manager to install .NET or .NET Core.

Errors related to missing `fxr`, `libhostfxr.so`, or `FrameworkList.xml`

For more information about solving these problems, see [Troubleshoot `fxr`, `libhostfxr.so`, and `FrameworkList.xml` errors](#).

Next steps

- [How to enable TAB completion for the .NET CLI](#)
- [Tutorial: Create a console application with .NET SDK using Visual Studio Code](#)

Install the .NET SDK or the .NET Runtime on Debian

9/20/2022 • 10 minutes to read • [Edit Online](#)

This article describes how to install .NET on Debian. When a Debian version falls out of support, .NET is no longer supported with that version. However, these instructions may help you to get .NET running on those versions, even though it isn't supported.

Install the SDK (which includes the runtime) if you want to develop .NET apps. Or, if you only need to run apps, install the Runtime. If you're installing the Runtime, we suggest you install the **ASP.NET Core Runtime** as it includes both .NET and ASP.NET Core runtimes.

If you've already installed the SDK or Runtime, use the `dotnet --list-sdks` and `dotnet --list-runtimes` commands to see which versions are installed. For more information, see [How to check that .NET is already installed](#).

IMPORTANT

Package manager installs are only supported on the **x64** architecture. Other architectures, such as **Arm**, must install .NET by some other means such as with Snap, an installer script, or through a manual binary installation.

For more information on installing .NET **without a package manager**, see one of the following articles:

- [Alternatively install .NET with Snap.](#)
- [Alternatively install .NET with `install-dotnet` script.](#)
- [Manually install .NET](#)

Supported distributions

The following table is a list of currently supported .NET releases and the versions of Debian they're supported on. These versions remain supported until either the version of [.NET reaches end-of-support](#) or the version of [Debian reaches end-of-life](#).

- A ✓ indicates that the version of Debian or .NET is still supported.
- A ✗ indicates that the version of Debian or .NET isn't supported on that Debian release.
- When both a version of Debian and a version of .NET have ✓, that OS and .NET combination is supported.

DEBIAN	.NET CORE 3.1	.NET 6
✓ 11	✓ 3.1	✓ 6.0
✓ 10	✓ 3.1	✓ 6.0
✓ 9	✓ 3.1	✓ 6.0
✗ 8	✗ 3.1	✗ 6.0

The following versions of .NET are ✗ no longer supported:

- .NET 5

- .NET Core 3.0
- .NET Core 2.2
- .NET Core 2.1
- .NET Core 2.0

Install preview versions

Preview and release candidate versions of .NET aren't available in package managers. You can install previews and release candidates of .NET in one of the following ways:

- [Snap package](#)
- [Scripted install with *install-dotnet.sh*](#)
- [Manual binary extraction](#)

Remove preview versions

When using a package manager to manage your installation of .NET, you may run into a conflict if you've previously installed a preview release. The package manager may interpret the non-preview release as an earlier version of .NET. To install the non-preview release, first uninstall the preview versions. For more information about uninstalling .NET, see [How to remove the .NET Runtime and SDK](#).

Debian 11 ✓

Installing with APT can be done with a few commands. Before you install .NET, run the following commands to add the Microsoft package signing key to your list of trusted keys and add the package repository.

Open a terminal and run the following commands:

```
wget https://packages.microsoft.com/config/debian/11/packages-microsoft-prod.deb -O packages-microsoft-prod.deb
sudo dpkg -i packages-microsoft-prod.deb
rm packages-microsoft-prod.deb
```

Install the SDK

The .NET SDK allows you to develop apps with .NET. If you install the .NET SDK, you don't need to install the corresponding runtime. To install the .NET SDK, run the following commands:

```
sudo apt-get update && \
sudo apt-get install -y dotnet-sdk-6.0
```

IMPORTANT

If you receive an error message similar to **Unable to locate package dotnet-sdk-6.0**, see the [APT troubleshooting](#) section.

Install the runtime

The ASP.NET Core Runtime allows you to run apps that were made with .NET that didn't provide the runtime. The following commands install the ASP.NET Core Runtime, which is the most compatible runtime for .NET. In your terminal, run the following commands:

```
sudo apt-get update && \
sudo apt-get install -y aspnetcore-runtime-6.0
```

IMPORTANT

If you receive an error message similar to **Unable to locate package aspnetcore-runtime-6.0**, see the [APT troubleshooting](#) section.

As an alternative to the ASP.NET Core Runtime, you can install the .NET Runtime, which doesn't include ASP.NET Core support: replace `aspnetcore-runtime-6.0` in the previous command with `dotnet-runtime-6.0`:

```
sudo apt-get install -y dotnet-runtime-6.0
```

Debian 10 ✓

Installing with APT can be done with a few commands. Before you install .NET, run the following commands to add the Microsoft package signing key to your list of trusted keys and add the package repository.

Open a terminal and run the following commands:

```
wget https://packages.microsoft.com/config/debian/10/packages-microsoft-prod.deb -O packages-microsoft-
prod.deb
sudo dpkg -i packages-microsoft-prod.deb
rm packages-microsoft-prod.deb
```

Install the SDK

The .NET SDK allows you to develop apps with .NET. If you install the .NET SDK, you don't need to install the corresponding runtime. To install the .NET SDK, run the following commands:

```
sudo apt-get update && \
sudo apt-get install -y dotnet-sdk-6.0
```

IMPORTANT

If you receive an error message similar to **Unable to locate package dotnet-sdk-6.0**, see the [APT troubleshooting](#) section.

Install the runtime

The ASP.NET Core Runtime allows you to run apps that were made with .NET that didn't provide the runtime. The following commands install the ASP.NET Core Runtime, which is the most compatible runtime for .NET. In your terminal, run the following commands:

```
sudo apt-get update && \
sudo apt-get install -y aspnetcore-runtime-6.0
```

IMPORTANT

If you receive an error message similar to **Unable to locate package aspnetcore-runtime-6.0**, see the [APT troubleshooting](#) section.

As an alternative to the ASP.NET Core Runtime, you can install the .NET Runtime, which doesn't include ASP.NET Core support: replace `aspnetcore-runtime-6.0` in the previous command with `dotnet-runtime-6.0`:

```
sudo apt-get install -y dotnet-runtime-6.0
```

Debian 9 ✓

Installing with APT can be done with a few commands. Before you install .NET, run the following commands to add the Microsoft package signing key to your list of trusted keys and add the package repository.

Open a terminal and run the following commands:

```
wget -O - https://packages.microsoft.com/keys/microsoft.asc | gpg --dearmor > microsoft.asc.gpg  
sudo mv microsoft.asc.gpg /etc/apt/trusted.gpg.d/  
wget https://packages.microsoft.com/config/debian/9/prod.list  
sudo mv prod.list /etc/apt/sources.list.d/microsoft-prod.list  
sudo chown root:root /etc/apt/trusted.gpg.d/microsoft.asc.gpg  
sudo chown root:root /etc/apt/sources.list.d/microsoft-prod.list
```

Install the SDK

The .NET SDK allows you to develop apps with .NET. If you install the .NET SDK, you don't need to install the corresponding runtime. To install the .NET SDK, run the following commands:

```
sudo apt-get update && \  
sudo apt-get install -y dotnet-sdk-6.0
```

IMPORTANT

If you receive an error message similar to **Unable to locate package dotnet-sdk-6.0**, see the [APT troubleshooting](#) section.

Install the runtime

The ASP.NET Core Runtime allows you to run apps that were made with .NET that didn't provide the runtime. The following commands install the ASP.NET Core Runtime, which is the most compatible runtime for .NET. In your terminal, run the following commands:

```
sudo apt-get update && \  
sudo apt-get install -y aspnetcore-runtime-6.0
```

IMPORTANT

If you receive an error message similar to **Unable to locate package aspnetcore-runtime-6.0**, see the [APT troubleshooting](#) section.

As an alternative to the ASP.NET Core Runtime, you can install the .NET Runtime, which doesn't include ASP.NET Core support: replace `aspnetcore-runtime-6.0` in the previous command with `dotnet-runtime-6.0`:

```
sudo apt-get install -y dotnet-runtime-6.0
```

Debian 8 X

X Please note that this version of Debian is no longer supported.

Installing with APT can be done with a few commands. Before you install .NET, run the following commands to add the Microsoft package signing key to your list of trusted keys and add the package repository.

Open a terminal and run the following commands:

```
wget -O - https://packages.microsoft.com/keys/microsoft.asc | gpg --dearmor > microsoft.asc.gpg
sudo mv microsoft.asc.gpg /etc/apt/trusted.gpg.d/
wget https://packages.microsoft.com/config/debian/8/prod.list
sudo mv prod.list /etc/apt/sources.list.d/microsoft-prod.list
sudo chown root:root /etc/apt/trusted.gpg.d/microsoft.asc.gpg
sudo chown root:root /etc/apt/sources.list.d/microsoft-prod.list
```

Install the SDK

The .NET Core SDK allows you to develop apps with .NET Core. If you install the .NET Core SDK, you don't need to install the corresponding runtime. To install the .NET Core SDK, run the following commands:

```
sudo apt-get update && \
sudo apt-get install -y dotnet-sdk-2.1
```

IMPORTANT

If you receive an error message similar to **Unable to locate package dotnet-sdk-2.1**, see the [APT troubleshooting](#) section.

Install the runtime

The .NET Core Runtime allows you to run apps that were made with .NET Core that didn't include the runtime. The following commands install the ASP.NET Core Runtime, which is the most compatible runtime for .NET Core. In your terminal, run the following commands.

```
sudo apt-get update && \
sudo apt-get install -y aspnetcore-runtime-2.1
```

IMPORTANT

If you receive an error message similar to **Unable to locate package aspnetcore-runtime-2.1**, see the [APT troubleshooting](#) section.

As an alternative to the ASP.NET Core Runtime, you can install the .NET Core Runtime, which doesn't include ASP.NET Core support: replace `aspnetcore-runtime-2.1` in the previous command with `dotnet-runtime-2.1`.

```
sudo apt-get install -y dotnet-runtime-2.1
```

How to install other versions

All versions of .NET are available for download at <https://dotnet.microsoft.com/download/dotnet>, but require [manual installation](#). You can try and use the package manager to install a different version of .NET. However, the requested version may not be available.

The packages added to package manager feeds are named in a hackable format, for example:

```
{product}-{type}-{version}.
```

- **product**

The type of .NET product to install. Valid options are:

- dotnet
- aspnetcore

- **type**

Chooses the SDK or the runtime. Valid options are:

- sdk
- runtime

- **version**

The version of the SDK or runtime to install. This article will always give the instructions for the latest supported version. Valid options are any released version, such as:

- 5.0
- 3.1
- 3.0
- 2.1

It's possible the SDK/runtime you're trying to download is not available for your Linux distribution. For a list of supported distributions, see [Install .NET on Linux](#).

Examples

- Install the ASP.NET Core 5.0 runtime: `aspnetcore-runtime-5.0`
- Install the .NET Core 2.1 runtime: `dotnet-runtime-2.1`
- Install the .NET 5 SDK: `dotnet-sdk-5.0`
- Install the .NET Core 3.1 SDK: `dotnet-sdk-3.1`

Package missing

If the package-version combination doesn't work, it's not available. For example, there isn't an ASP.NET Core SDK, the SDK components are included with the .NET SDK. The value `aspnetcore-sdk-2.2` is incorrect and should be `dotnet-sdk-2.2`. For a list of Linux distributions supported by .NET, see [.NET dependencies and requirements](#).

Use APT to update .NET

When a new patch release is available for .NET, you can simply upgrade it through APT with the following commands:

```
sudo apt-get update  
sudo apt-get upgrade
```

If you've upgraded your Linux distribution since installing .NET, you may need to reconfigure the Microsoft package repository. Run the installation instructions for your current distribution version to upgrade to the appropriate package repository for .NET updates.

APT troubleshooting

This section provides information on common errors you may get while using APT to install .NET.

Unable to find package

IMPORTANT

Package manager installs are only supported on the **x64** architecture. Other architectures, such as **Arm**, must install .NET by some other means such as with Snap, an installer script, or through a manual binary installation.

For more information on installing .NET **without a package manager**, see one of the following articles:

- [Alternatively install .NET with Snap.](#)
- [Alternatively install .NET with `install-dotnet` script.](#)
- [Manually install .NET](#)

Unable to locate \ Some packages could not be installed

If you receive an error message similar to **Unable to locate package {dotnet-package}** or **Some packages could not be installed**, run the following commands.

There are two placeholders in the following set of commands.

- `{dotnet-package}`
This represents the .NET package you're installing, such as `aspnetcore-runtime-3.1`. This is used in the following `sudo apt-get install` command.
- `{os-version}`
This represents the distribution version you're on. This is used in the `wget` command below. The distribution version is the numerical value, such as `20.04` on Ubuntu or `10` on Debian.

First, try purging the package list:

```
sudo dpkg --purge packages-microsoft-prod && sudo dpkg -i packages-microsoft-prod.deb  
sudo apt-get update
```

Then, try to install .NET again. If that doesn't work, you can run a manual install with the following commands:

```
sudo apt-get install -y gpg  
wget -O - https://packages.microsoft.com/keys/microsoft.asc | gpg --dearmor -o microsoft.asc.gpg  
sudo mv microsoft.asc.gpg /etc/apt/trusted.gpg.d/  
wget https://packages.microsoft.com/config/debian/{os-version}/prod.list  
sudo mv prod.list /etc/apt/sources.list.d/microsoft-prod.list  
sudo chown root:root /etc/apt/trusted.gpg.d/microsoft.asc.gpg  
sudo chown root:root /etc/apt/sources.list.d/microsoft-prod.list  
sudo apt-get update && \  
sudo apt-get install -y {dotnet-package}
```

Failed to fetch

While installing the .NET package, you may see an error similar to

`Failed to fetch ... File has unexpected size ... Mirror sync in progress?`. This error could mean that the package feed for .NET is being upgraded with newer package versions, and that you should try again later. During an upgrade, the package feed shouldn't be unavailable for more than 30 minutes. If you continually receive this error for more than 30 minutes, please file an issue at <https://github.com/dotnet/core/issues>.

Dependencies

When you install with a package manager, these libraries are installed for you. But, if you manually install .NET Core or you publish a self-contained app, you'll need to make sure these libraries are installed:

- libc6

- libgcc-s1
- libgssapi-krb5-2
- libicu52 (for 8.x)
- libicu57 (for 9.x)
- libicu63 (for 10.x)
- libicu67 (for 11.x)
- libssl1.0.0 (for 8.x)
- libssl1.1 (for 9.x-11.x)
- libstdc++6
- zlib1g

For .NET Core apps that use the *System.Drawing.Common* assembly, you also need the following dependency:

- libgdipplus (version 6.0.1 or later)

WARNING

You can install a recent version of *libgdipplus* by adding the Mono repository to your system. For more information, see <https://www.mono-project.com/download/stable/>.

Next steps

- [How to enable TAB completion for the .NET CLI](#)
- [Tutorial: Create a console application with .NET SDK using Visual Studio Code](#)

Install the .NET SDK or the .NET Runtime on Fedora

9/20/2022 • 7 minutes to read • [Edit Online](#)

.NET is supported on Fedora and this article describes how to install .NET on Fedora. When a Fedora version falls out of support, .NET is no longer supported with that version.

Install the SDK (which includes the runtime) if you want to develop .NET apps. Or, if you only need to run apps, install the Runtime. If you're installing the Runtime, we suggest you install the **ASP.NET Core Runtime** as it includes both .NET and ASP.NET Core runtimes.

If you've already installed the SDK or Runtime, use the `dotnet --list-sdks` and `dotnet --list-runtimes` commands to see which versions are installed. For more information, see [How to check that .NET is already installed](#).

For more information on installing .NET without a package manager, see one of the following articles:

- [Install the .NET SDK or the .NET Runtime with Snap](#).
- [Install the .NET SDK or the .NET Runtime with a script](#).
- [Install the .NET SDK or the .NET Runtime manually](#).

Install .NET 6

Install the SDK

The .NET SDK allows you to develop apps with .NET. If you install the .NET SDK, you don't need to install the corresponding runtime. To install the .NET SDK, run the following command:

```
sudo dnf install dotnet-sdk-6.0
```

Install the runtime

The ASP.NET Core Runtime allows you to run apps that were made with .NET that didn't provide the runtime. The following command install the ASP.NET Core Runtime, which is the most compatible runtime for .NET. In your terminal, run the following command:

```
sudo dnf install aspnetcore-runtime-6.0
```

As an alternative to the ASP.NET Core Runtime, you can install the .NET Runtime, which doesn't include ASP.NET Core support: replace `aspnetcore-runtime-6.0` in the previous command with `dotnet-runtime-6.0`:

```
sudo dnf install dotnet-runtime-6.0
```

Install .NET Core 3.1

Install the SDK

The .NET Core SDK allows you to develop apps with .NET Core. If you install the .NET Core SDK, you don't need to install the corresponding runtime. To install the .NET Core SDK, run the following command:

```
sudo dnf install dotnet-sdk-3.1
```

Install the runtime

The .NET Core Runtime allows you to run apps that were made with .NET Core that didn't include the runtime. The following command install the ASP.NET Core Runtime, which is the most compatible runtime for .NET Core. In your terminal, run the following command.

```
sudo dnf install aspnetcore-runtime-3.1
```

As an alternative to the ASP.NET Core Runtime, you can install the .NET Core Runtime, which doesn't include ASP.NET Core support: replace `aspnetcore-runtime-3.1` in the previous command with `dotnet-runtime-3.1`.

```
sudo dnf install dotnet-runtime-3.1
```

Supported distributions

The following table is a list of currently supported .NET releases and the versions of Fedora they're supported on. These versions remain supported until either the version of [.NET reaches end-of-support](#) or the version of [Fedora reaches end-of-life](#).

- A ✓ indicates that the version of Fedora or .NET is still supported.
- A ✗ indicates that the version of Fedora or .NET isn't supported on that Fedora release.
- When both a version of Fedora and a version of .NET have ✓, that OS and .NET combination is supported.

.NET VERSION	FEDORA 36	35 ✓	34 ✗	33 ✗	32 ✗	31 ✗	30 ✗	29 ✗	28 ✗	27 ✗
.NET 6	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗
.NET Core 3.1	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗

The following versions of .NET are ✗ no longer supported:

- .NET 5
- .NET Core 3.0
- .NET Core 2.2
- .NET Core 2.1
- .NET Core 2.0

Install preview versions

Preview and release candidate versions of .NET aren't available in package managers. You can install previews and release candidates of .NET in one of the following ways:

- [Snap package](#)
- [Scripted install with `install-dotnet.sh`](#)
- [Manual binary extraction](#)

Remove preview versions

When using a package manager to manage your installation of .NET, you may run into a conflict if you've previously installed a preview release. The package manager may interpret the non-preview release as an earlier

version of .NET. To install the non-preview release, first uninstall the preview versions. For more information about uninstalling .NET, see [How to remove the .NET Runtime and SDK](#).

Dependencies

When you install with a package manager, these libraries are installed for you. But, if you manually install .NET Core or you publish a self-contained app, you'll need to make sure these libraries are installed:

- krb5-libs
- libicu
- openssl-libs
- zlib

If the target runtime environment's OpenSSL version is 1.1 or newer, you'll need to install `compat-openssl10`.

For more information about the dependencies, see [Self-contained Linux apps](#).

For .NET Core apps that use the `System.Drawing.Common` assembly, you'll also need the following dependency:

- `libgdiplus` (version 6.0.1 or later)

WARNING

You can install a recent version of `libgdiplus` by adding the Mono repository to your system. For more information, see <https://www.monoproject.com/download/stable/>.

Install on older distributions

Older versions of Fedora don't contain .NET Core in the default package repositories. You can install .NET with `snap`, through the [`dotnet-install.sh` script](#), or use Microsoft's repository to install .NET:

1. First, add the Microsoft signing key to your list of trusted keys.

```
sudo rpm --import https://packages.microsoft.com/keys/microsoft.asc
```

2. Next, add the Microsoft package repository. The source of the repository is based on your version of Fedora.

FEDORA VERSION	PACKAGE REPOSITORY
33	https://packages.microsoft.com/config/fedora/33/prod.repo
32	https://packages.microsoft.com/config/fedora/32/prod.repo
31	https://packages.microsoft.com/config/fedora/31/prod.repo
30	https://packages.microsoft.com/config/fedora/30/prod.repo
29	https://packages.microsoft.com/config/fedora/29/prod.repo
28	https://packages.microsoft.com/config/fedora/28/prod.repo
27	https://packages.microsoft.com/config/fedora/27/prod.repo

```
sudo wget -O /etc/yum.repos.d/microsoft-prod.repo  
https://packages.microsoft.com/config/fedora/31/prod.repo
```

Install the SDK

The .NET Core SDK allows you to develop apps with .NET Core. If you install the .NET Core SDK, you don't need to install the corresponding runtime. To install the .NET Core SDK, run the following command:

```
sudo dnf install dotnet-sdk-3.1
```

Install the runtime

The .NET Core Runtime allows you to run apps that were made with .NET Core that didn't include the runtime. The following command install the ASP.NET Core Runtime, which is the most compatible runtime for .NET Core. In your terminal, run the following command.

```
sudo dnf install aspnetcore-runtime-3.1
```

As an alternative to the ASP.NET Core Runtime, you can install the .NET Core Runtime, which doesn't include ASP.NET Core support: replace `aspnetcore-runtime-3.1` in the previous command with `dotnet-runtime-3.1`.

```
sudo dnf install dotnet-runtime-3.1
```

How to install other versions

All versions of .NET are available for download at <https://dotnet.microsoft.com/download/dotnet>, but require [manual installation](#). You can try and use the package manager to install a different version of .NET. However, the requested version may not be available.

The packages added to package manager feeds are named in a hackable format, for example:

```
{product}-{type}-{version} .
```

- **product**

The type of .NET product to install. Valid options are:

- dotnet
- aspnetcore

- **type**

Chooses the SDK or the runtime. Valid options are:

- sdk
- runtime

- **version**

The version of the SDK or runtime to install. This article will always give the instructions for the latest supported version. Valid options are any released version, such as:

- 5.0
- 3.1
- 3.0
- 2.1

It's possible the SDK/runtime you're trying to download is not available for your Linux distribution. For a list of supported distributions, see [Install .NET on Linux](#).

Examples

- Install the ASP.NET Core 5.0 runtime: `aspnetcore-runtime-5.0`
- Install the .NET Core 2.1 runtime: `dotnet-runtime-2.1`
- Install the .NET 5 SDK: `dotnet-sdk-5.0`
- Install the .NET Core 3.1 SDK: `dotnet-sdk-3.1`

Package missing

If the package-version combination doesn't work, it's not available. For example, there isn't an ASP.NET Core SDK, the SDK components are included with the .NET SDK. The value `aspnetcore-sdk-2.2` is incorrect and should be `dotnet-sdk-2.2`. For a list of Linux distributions supported by .NET, see [.NET dependencies and requirements](#).

Troubleshoot the package manager

This section provides information on common errors you may get while using the package manager to install .NET or .NET Core.

Unable to find package

For more information on installing .NET without a package manager, see one of the following articles:

- [Install the .NET SDK or the .NET Runtime with Snap](#).
- [Install the .NET SDK or the .NET Runtime with a script](#).
- [Install the .NET SDK or the .NET Runtime manually](#).

Failed to fetch

While installing the .NET package, you may see an error similar to

`signature verification failed for file 'repomd.xml' from repository 'packages-microsoft-com-prod'`. Generally speaking, this error means that the package feed for .NET is being upgraded with newer package versions, and that you should try again later. During an upgrade, the package feed should not be unavailable for more than 2 hours. If you continually receive this error for more than 2 hours, please file an issue at <https://github.com/dotnet/core/issues>.

Errors related to missing `fxr`, `libhostfxr.so`, or `FrameworkList.xml`

For more information about solving these problems, see [Troubleshoot `fxr`, `libhostfxr.so`, and `FrameworkList.xml` errors](#).

Next steps

- [How to enable TAB completion for the .NET CLI](#)
- [Tutorial: Create a console application with .NET SDK using Visual Studio Code](#)

Install the .NET SDK or the .NET Runtime on openSUSE

9/20/2022 • 5 minutes to read • [Edit Online](#)

.NET is supported on openSUSE. This article describes how to install .NET on openSUSE.

Install the SDK (which includes the runtime) if you want to develop .NET apps. Or, if you only need to run apps, install the Runtime. If you're installing the Runtime, we suggest you install the **ASP.NET Core Runtime** as it includes both .NET and ASP.NET Core runtimes.

If you've already installed the SDK or Runtime, use the `dotnet --list-sdks` and `dotnet --list-runtimes` commands to see which versions are installed. For more information, see [How to check that .NET is already installed](#).

IMPORTANT

Package manager installs are only supported on the **x64** architecture. Other architectures, such as **Arm**, must install .NET by some other means such as with Snap, an installer script, or through a manual binary installation.

For more information on installing .NET without a package manager, see one of the following articles:

- [Alternatively install .NET with Snap](#).
- [Alternatively install .NET with `install-dotnet` script](#).
- [Manually install .NET](#)

Supported distributions

The following table is a list of currently supported .NET releases on openSUSE 15. These versions remain supported until either the version of [.NET reaches end-of-support](#) or the version of openSUSE is no longer supported.

- A ✓ indicates that the version of openSUSE or .NET is still supported.
- A ✗ indicates that the version of openSUSE or .NET isn't supported on that openSUSE release.
- When both a version of openSUSE and a version of .NET have ✓, that OS and .NET combination is supported.

OPEN SUSE	.NET CORE 3.1	.NET 6
✓ 15	✓ 3.1	✓ 6.0

The following versions of .NET are ✗ no longer supported:

- .NET 5
- .NET Core 3.0
- .NET Core 2.2
- .NET Core 2.1
- .NET Core 2.0

Install preview versions

Preview and release candidate versions of .NET aren't available in package managers. You can install previews and release candidates of .NET in one of the following ways:

- [Snap package](#)
- [Scripted install with *install-dotnet.sh*](#)
- [Manual binary extraction](#)

Remove preview versions

When using a package manager to manage your installation of .NET, you may run into a conflict if you've previously installed a preview release. The package manager may interpret the non-preview release as an earlier version of .NET. To install the non-preview release, first uninstall the preview versions. For more information about uninstalling .NET, see [How to remove the .NET Runtime and SDK](#).

openSUSE 15 ✓

Before you install .NET, run the following commands to add the Microsoft package signing key to your list of trusted keys and add the Microsoft package repository. Open a terminal and run the following commands:

```
sudo zypper install libicu
sudo rpm --import https://packages.microsoft.com/keys/microsoft.asc
wget https://packages.microsoft.com/config/opensuse/15/prod.repo
sudo mv prod.repo /etc/zypp/repos.d/microsoft-prod.repo
sudo chown root:root /etc/zypp/repos.d/microsoft-prod.repo
```

Install the SDK

The .NET SDK allows you to develop apps with .NET. If you install the .NET SDK, you don't need to install the corresponding runtime. To install the .NET SDK, run the following command:

```
sudo zypper install dotnet-sdk-6.0
```

Install the runtime

The ASP.NET Core Runtime allows you to run apps that were made with .NET that didn't provide the runtime. The following command installs the ASP.NET Core Runtime, which is the most compatible runtime for .NET. In your terminal, run the following command:

```
sudo zypper install aspnetcore-runtime-6.0
```

As an alternative to the ASP.NET Core Runtime, you can install the .NET Runtime, which doesn't include ASP.NET Core support: replace `aspnetcore-runtime-6.0` in the previous command with `dotnet-runtime-6.0`:

```
sudo zypper install dotnet-runtime-6.0
```

How to install other versions

All versions of .NET are available for download at <https://dotnet.microsoft.com/download/dotnet>, but require [manual installation](#). You can try and use the package manager to install a different version of .NET. However, the requested version may not be available.

The packages added to package manager feeds are named in a hackable format, for example:

```
{product}-{type}-{version}.
```

- **product**

The type of .NET product to install. Valid options are:

- dotnet
- aspnetcore

- **type**

Chooses the SDK or the runtime. Valid options are:

- sdk
- runtime

- **version**

The version of the SDK or runtime to install. This article will always give the instructions for the latest supported version. Valid options are any released version, such as:

- 5.0
- 3.1
- 3.0
- 2.1

It's possible the SDK/runtime you're trying to download is not available for your Linux distribution. For a list of supported distributions, see [Install .NET on Linux](#).

Examples

- Install the ASP.NET Core 5.0 runtime: `aspnetcore-runtime-5.0`
- Install the .NET Core 2.1 runtime: `dotnet-runtime-2.1`
- Install the .NET 5 SDK: `dotnet-sdk-5.0`
- Install the .NET Core 3.1 SDK: `dotnet-sdk-3.1`

Package missing

If the package-version combination doesn't work, it's not available. For example, there isn't an ASP.NET Core SDK, the SDK components are included with the .NET SDK. The value `aspnetcore-sdk-2.2` is incorrect and should be `dotnet-sdk-2.2`. For a list of Linux distributions supported by .NET, see [.NET dependencies and requirements](#).

Troubleshoot the package manager

This section provides information on common errors you may get while using the package manager to install .NET.

Unable to find package

IMPORTANT

Package manager installs are only supported on the **x64** architecture. Other architectures, such as **Arm**, must install .NET by some other means such as with Snap, an installer script, or through a manual binary installation.

For more information on installing .NET **without a package manager**, see one of the following articles:

- [Alternatively install .NET with Snap](#).
- [Alternatively install .NET with `install-dotnet` script](#).
- [Manually install .NET](#)

Failed to fetch

While installing the .NET package, you may see an error similar to

`signature verification failed for file 'repomd.xml' from repository 'packages-microsoft-com-prod'`. Generally

speaking, this error means that the package feed for .NET is being upgraded with newer package versions, and that you should try again later. During an upgrade, the package feed should not be unavailable for more than 2 hours. If you continually receive this error for more than 2 hours, please file an issue at <https://github.com/dotnet/core/issues>.

Dependencies

When you install with a package manager, these libraries are installed for you. But, if you manually install .NET or you publish a self-contained app, you'll need to make sure these libraries are installed:

- krb5
- libicu
- libopenssl1_0_0

If the target runtime environment's OpenSSL version is 1.1 or newer, you'll need to install **compat-openssl10**.

For more information about the dependencies, see [Self-contained Linux apps](#).

For .NET apps that use the *System.Drawing.Common* assembly, you'll also need the following dependency:

- [libgdiplus \(version 6.0.1 or later\)](#)

WARNING

You can install a recent version of *libgdiplus* by adding the Mono repository to your system. For more information, see <https://www.mono-project.com/download/stable/>.

Next steps

- [How to enable TAB completion for the .NET CLI](#)
- [Tutorial: Create a console application with .NET SDK using Visual Studio Code](#)

Install the .NET SDK or the .NET Runtime on RHEL and CentOS Stream

9/20/2022 • 8 minutes to read • [Edit Online](#)

.NET is supported on Red Hat Enterprise Linux (RHEL). This article describes how to install .NET on RHEL and CentOS Stream.

Install the SDK (which includes the runtime) if you want to develop .NET apps. Or, if you only need to run apps, install the Runtime. If you're installing the Runtime, we suggest you install the **ASP.NET Core Runtime** as it includes both .NET and ASP.NET Core runtimes.

If you've already installed the SDK or Runtime, use the `dotnet --list-sdks` and `dotnet --list-runtimes` commands to see which versions are installed. For more information, see [How to check that .NET is already installed](#).

Register your Red Hat subscription

To install .NET from Red Hat on RHEL, you first need to register using the Red Hat Subscription Manager. If this hasn't been done on your system, or if you're unsure, see the [Red Hat Product Documentation for .NET](#).

IMPORTANT

This doesn't apply to CentOS Stream.

Supported distributions

The following table is a list of currently supported .NET releases on both RHEL and CentOS Stream. These versions remain supported until either the version of [.NET reaches end-of-support](#) or the Linux distribution is no longer supported.

- A ✓ indicates that the version of RHEL or .NET is still supported.
- A ✗ indicates that the version of RHEL or .NET isn't supported on that RHEL release.
- When both a version of Linux distribution and a version of .NET have ✓, that OS and .NET combination is supported.

DISTRIBUTION	.NET CORE 3.1	.NET 6
✓ RHEL 8	✓ 3.1	✓ 6.0
✓ RHEL 7	✓ 3.1	✓ 6.0
✓ CentOS Stream 9	✗ 3.1	✓ 6.0
✓ CentOS Stream 8	✓ 3.1	✓ 6.0

The following versions of .NET are ✗ no longer supported:

- .NET 5
- .NET Core 3.0

- .NET Core 2.2
- .NET Core 2.1
- .NET Core 2.0

Install preview versions

Preview and release candidate versions of .NET aren't available in package managers. You can install previews and release candidates of .NET in one of the following ways:

- [Snap package](#)
- [Scripted install with *install-dotnet.sh*](#)
- [Manual binary extraction](#)

Remove preview versions

When using a package manager to manage your installation of .NET, you may run into a conflict if you've previously installed a preview release. The package manager may interpret the non-preview release as an earlier version of .NET. To install the non-preview release, first uninstall the preview versions. For more information about uninstalling .NET, see [How to remove the .NET Runtime and SDK](#).

RHEL 8 ✓

.NET is included in the AppStream repositories for RHEL 8.

Install the SDK

The .NET SDK allows you to develop apps with .NET. If you install the .NET SDK, you don't need to install the corresponding runtime. To install the .NET SDK, run the following command:

```
sudo dnf install dotnet-sdk-6.0
```

Install the runtime

The ASP.NET Core Runtime allows you to run apps that were made with .NET that didn't provide the runtime. The following command install the ASP.NET Core Runtime, which is the most compatible runtime for .NET. In your terminal, run the following command:

```
sudo dnf install aspnetcore-runtime-6.0
```

As an alternative to the ASP.NET Core Runtime, you can install the .NET Runtime, which doesn't include ASP.NET Core support: replace `aspnetcore-runtime-6.0` in the previous command with `dotnet-runtime-6.0`:

```
sudo dnf install dotnet-runtime-6.0
```

RHEL 7 ✓ .NET 6.0

The following command installs the `scl-utils` package:

```
sudo yum install scl-utils
```

Install the SDK

The .NET SDK allows you to develop apps with .NET. If you install the .NET SDK, you don't need to install the

corresponding runtime. To install .NET SDK, run the following commands:

```
subscription-manager repos --enable=rhel-7-server-dotnet-rpms  
yum install rh-dotnet60 -y  
scl enable rh-dotnet60 bash
```

Red Hat does not recommend permanently enabling `rh-dotnet60` because it may affect other programs. If you want to enable `rh-dotnet` permanently, add the following line to your `~/.bashrc` file.

```
source scl_source enable rh-dotnet60
```

Install the runtime

The .NET Runtime allows you to run apps that were made with .NET that didn't include the runtime. The commands below install the ASP.NET Core Runtime, which is the most compatible runtime for .NET Core. In your terminal, run the following commands.

```
subscription-manager repos --enable=rhel-7-server-dotnet-rpms  
yum install rh-dotnet60-aspnetcore-runtime-6.0 -y  
scl enable rh-dotnet60 bash
```

Red Hat does not recommend permanently enabling `rh-dotnet60` because it may affect other programs. If you want to enable `rh-dotnet60` permanently, add the following line to your `~/.bashrc` file.

```
source scl_source enable rh-dotnet60
```

As an alternative to the ASP.NET Core Runtime, you can install the .NET Runtime that doesn't include ASP.NET Core support: replace `rh-dotnet60-aspnetcore-runtime-6.0` in the preceding command with `rh-dotnet60-dotnet-runtime-6.0`.

RHEL 7 ✓ .NET 5.0

The following command installs the `scl-utils` package:

```
sudo yum install scl-utils
```

Install the SDK

The .NET SDK allows you to develop apps with .NET. If you install the .NET SDK, you don't need to install the corresponding runtime. To install .NET SDK, run the following commands:

```
subscription-manager repos --enable=rhel-7-server-dotnet-rpms  
yum install rh-dotnet50 -y  
scl enable rh-dotnet50 bash
```

Red Hat does not recommend permanently enabling `rh-dotnet50` because it may affect other programs. If you want to enable `rh-dotnet` permanently, add the following line to your `~/.bashrc` file.

```
source scl_source enable rh-dotnet50
```

Install the runtime

The .NET Runtime allows you to run apps that were made with .NET that didn't include the runtime. The

commands below install the ASP.NET Core Runtime, which is the most compatible runtime for .NET Core. In your terminal, run the following commands.

```
subscription-manager repos --enable=rhel-7-server-dotnet-rpms
yum install rh-dotnet50-aspnetcore-runtime-5.0 -y
scl enable rh-dotnet50 bash
```

Red Hat does not recommend permanently enabling `rh-dotnet50` because it may affect other programs. If you want to enable `rh-dotnet50` permanently, add the following line to your `~/.bashrc` file.

```
source scl_source enable rh-dotnet50
```

As an alternative to the ASP.NET Core Runtime, you can install the .NET Runtime that doesn't include ASP.NET Core support: replace `rh-dotnet50-aspnetcore-runtime-5.0` in the commands above with `rh-dotnet50-dotnet-runtime-5.0`.

RHEL 7 ✓ .NET Core 3.1

Before you install .NET, run the following commands to add the Microsoft package signing key to your list of trusted keys and add the Microsoft package repository. Open a terminal and run the following commands:

The following command installs the `scl-utils` package:

```
sudo yum install scl-utils
```

Install the SDK

.NET SDK allows you to develop apps with .NET Core. If you install .NET SDK, you don't need to install the corresponding runtime. To install .NET SDK, run the following commands:

```
subscription-manager repos --enable=rhel-7-server-dotnet-rpms
yum install rh-dotnet31 -y
scl enable rh-dotnet31 bash
```

Red Hat does not recommend permanently enabling `rh-dotnet31` because it may affect other programs. For example, `rh-dotnet31` includes a version of `libcurl` that differs from the base RHEL version. This may lead to issues in programs that do not expect a different version of `libcurl`. If you want to enable `rh-dotnet` permanently, add the following line to your `~/.bashrc` file.

```
source scl_source enable rh-dotnet31
```

Install the runtime

The .NET Core Runtime allows you to run apps that were made with .NET Core that didn't include the runtime. The commands below install the ASP.NET Core Runtime, which is the most compatible runtime for .NET Core. In your terminal, run the following commands.

```
subscription-manager repos --enable=rhel-7-server-dotnet-rpms
yum install rh-dotnet31-aspnetcore-runtime-3.1 -y
scl enable rh-dotnet31 bash
```

Red Hat does not recommend permanently enabling `rh-dotnet31` because it may affect other programs. For example, `rh-dotnet31` includes a version of `libcurl` that differs from the base RHEL version. This may lead to

issues in programs that do not expect a different version of `libcurl`. If you want to enable `rh-dotnet31` permanently, add the following line to your `~/.bashrc` file.

```
source scl_source enable rh-dotnet31
```

As an alternative to the ASP.NET Core Runtime, you can install the .NET Core Runtime that doesn't include ASP.NET Core support: replace `rh-dotnet31-aspnetcore-runtime-3.1` in the commands above with `rh-dotnet31-dotnet-runtime-3.1`.

CentOS Stream 9 ✓

.NET is included in the AppStream repositories for CentOS Stream 9. However, .NET Core 3.1 and .NET 5 have been removed from CentOS Stream 9 and you should use .NET 6. For more information, see the blog post [Using .NET with OpenSSL in CentOS Stream 9 | Omair Majid](#).

Install the SDK

The .NET SDK allows you to develop apps with .NET. If you install the .NET SDK, you don't need to install the corresponding runtime. To install the .NET SDK, run the following command:

```
sudo dnf install dotnet-sdk-6.0
```

Install the runtime

The ASP.NET Core Runtime allows you to run apps that were made with .NET that didn't provide the runtime. The following command install the ASP.NET Core Runtime, which is the most compatible runtime for .NET. In your terminal, run the following command:

```
sudo dnf install aspnetcore-runtime-6.0
```

As an alternative to the ASP.NET Core Runtime, you can install the .NET Runtime, which doesn't include ASP.NET Core support: replace `aspnetcore-runtime-6.0` in the previous command with `dotnet-runtime-6.0`:

```
sudo dnf install dotnet-runtime-6.0
```

CentOS Stream 8 ✓

.NET is included in the AppStream repositories for CentOS Stream 8.

Install the SDK

The .NET SDK allows you to develop apps with .NET. If you install the .NET SDK, you don't need to install the corresponding runtime. To install the .NET SDK, run the following command:

```
sudo dnf install dotnet-sdk-6.0
```

Install the runtime

The ASP.NET Core Runtime allows you to run apps that were made with .NET that didn't provide the runtime. The following command install the ASP.NET Core Runtime, which is the most compatible runtime for .NET. In your terminal, run the following command:

```
sudo dnf install aspnetcore-runtime-6.0
```

As an alternative to the ASP.NET Core Runtime, you can install the .NET Runtime, which doesn't include ASP.NET Core support: replace `aspnetcore-runtime-6.0` in the previous command with `dotnet-runtime-6.0`:

```
sudo dnf install dotnet-runtime-6.0
```

Dependencies

When you install with a package manager, these libraries are installed for you. But, if you manually install .NET Core or you publish a self-contained app, you'll need to make sure these libraries are installed:

- `krb5-libs`
- `libicu`
- `openssl-libs`
- `zlib`

If the target runtime environment's OpenSSL version is 1.1 or newer, you'll need to install `compat-openssl10`.

For more information about the dependencies, see [Self-contained Linux apps](#).

For .NET Core apps that use the `System.Drawing.Common` assembly, you'll also need the following dependency:

- [libgdiplus \(version 6.0.1 or later\)](#)

WARNING

You can install a recent version of `libgdiplus` by adding the Mono repository to your system. For more information, see <https://www.mono-project.com/download/stable/>.

How to install other versions

Consult the [Red Hat documentation for .NET](#) on the steps required to install other releases of .NET.

Troubleshoot the package manager

This section provides information on common errors you may get while using the package manager to install .NET or .NET Core.

Errors related to missing `fxr`, `libhostfxr.so`, or `FrameworkList.xml`

For more information about solving these problems, see [Troubleshoot `fxr`, `libhostfxr.so`, and `FrameworkList.xml` errors](#).

Next steps

- [How to enable TAB completion for the .NET CLI](#)
- [Tutorial: Create a console application with .NET SDK using Visual Studio Code](#)

Install the .NET SDK or the .NET Runtime on SLES

9/20/2022 • 5 minutes to read • [Edit Online](#)

.NET is supported on SLES. This article describes how to install .NET on SLES.

Install the SDK (which includes the runtime) if you want to develop .NET apps. Or, if you only need to run apps, install the Runtime. If you're installing the Runtime, we suggest you install the **ASP.NET Core Runtime** as it includes both .NET and ASP.NET Core runtimes.

If you've already installed the SDK or Runtime, use the `dotnet --list-sdks` and `dotnet --list-runtimes` commands to see which versions are installed. For more information, see [How to check that .NET is already installed](#).

Supported distributions

The following table is a list of currently supported .NET releases on both SLES 12 SP2 and SLES 15. These versions remain supported until either the version of [.NET reaches end-of-support](#) or the version of SLES is no longer supported.

- A ✓ indicates that the version of SLES or .NET is still supported.
- A ✗ indicates that the version of SLES or .NET isn't supported on that SLES release.
- When both a version of SLES and a version of .NET have ✓, that OS and .NET combination is supported.

SLES	.NET CORE 3.1	.NET 6
✓ 15	✓ 3.1	✓ 6.0
✓ 12 SP2	✓ 3.1	✓ 6.0

The following versions of .NET are ✗ no longer supported:

- .NET 5
- .NET Core 3.0
- .NET Core 2.2
- .NET Core 2.1
- .NET Core 2.0

Install preview versions

Preview and release candidate versions of .NET aren't available in package managers. You can install previews and release candidates of .NET in one of the following ways:

- [Snap package](#)
- [Scripted install with *install-dotnet.sh*](#)
- [Manual binary extraction](#)

Remove preview versions

When using a package manager to manage your installation of .NET, you may run into a conflict if you've previously installed a preview release. The package manager may interpret the non-preview release as an earlier

version of .NET. To install the non-preview release, first uninstall the preview versions. For more information about uninstalling .NET, see [How to remove the .NET Runtime and SDK](#).

SLES 15 ✓

Before you install .NET, run the following commands to add the Microsoft package signing key to your list of trusted keys and add the Microsoft package repository. Open a terminal and run the following commands:

```
sudo rpm -Uvh https://packages.microsoft.com/config/sles/15/packages-microsoft-prod.rpm
```

Currently, the SLES 15 Microsoft repository setup package installs the *microsoft-prod.repo* file to the wrong directory, preventing zypper from finding the .NET packages. To fix this problem, create a symlink in the correct directory.

```
sudo ln -s /etc/yum.repos.d/microsoft-prod.repo /etc/zypp/repos.d/microsoft-prod.repo
```

Install the SDK

The .NET SDK allows you to develop apps with .NET. If you install the .NET SDK, you don't need to install the corresponding runtime. To install the .NET SDK, run the following command:

```
sudo zypper install dotnet-sdk-6.0
```

Install the runtime

The ASP.NET Core Runtime allows you to run apps that were made with .NET that didn't provide the runtime. The following command installs the ASP.NET Core Runtime, which is the most compatible runtime for .NET. In your terminal, run the following command:

```
sudo zypper install aspnetcore-runtime-6.0
```

As an alternative to the ASP.NET Core Runtime, you can install the .NET Runtime, which doesn't include ASP.NET Core support: replace `aspnetcore-runtime-6.0` in the previous command with `dotnet-runtime-6.0`:

```
sudo zypper install dotnet-runtime-6.0
```

SLES 12 ✓

.NET requires SP2 as a minimum for the SLES 12 family.

Before you install .NET, run the following commands to add the Microsoft package signing key to your list of trusted keys and add the Microsoft package repository. Open a terminal and run the following commands:

```
sudo rpm -Uvh https://packages.microsoft.com/config/sles/12/packages-microsoft-prod.rpm
```

Install the SDK

The .NET SDK allows you to develop apps with .NET. If you install the .NET SDK, you don't need to install the corresponding runtime. To install the .NET SDK, run the following command:

```
sudo zypper install dotnet-sdk-6.0
```

Install the runtime

The ASP.NET Core Runtime allows you to run apps that were made with .NET that didn't provide the runtime. The following command installs the ASP.NET Core Runtime, which is the most compatible runtime for .NET. In your terminal, run the following command:

```
sudo zypper install aspnetcore-runtime-6.0
```

As an alternative to the ASP.NET Core Runtime, you can install the .NET Runtime, which doesn't include ASP.NET Core support: replace `aspnetcore-runtime-6.0` in the previous command with `dotnet-runtime-6.0`:

```
sudo zypper install dotnet-runtime-6.0
```

How to install other versions

All versions of .NET are available for download at <https://dotnet.microsoft.com/download/dotnet>, but require [manual installation](#). You can try and use the package manager to install a different version of .NET. However, the requested version may not be available.

The packages added to package manager feeds are named in a hackable format, for example:

```
{product}-{type}-{version}.
```

- **product**

The type of .NET product to install. Valid options are:

- dotnet
- aspnetcore

- **type**

Chooses the SDK or the runtime. Valid options are:

- sdk
- runtime

- **version**

The version of the SDK or runtime to install. This article will always give the instructions for the latest supported version. Valid options are any released version, such as:

- 5.0
- 3.1
- 3.0
- 2.1

It's possible the SDK/runtime you're trying to download is not available for your Linux distribution. For a list of supported distributions, see [Install .NET on Linux](#).

Examples

- Install the ASP.NET Core 5.0 runtime: `aspnetcore-runtime-5.0`
- Install the .NET Core 2.1 runtime: `dotnet-runtime-2.1`
- Install the .NET 5 SDK: `dotnet-sdk-5.0`
- Install the .NET Core 3.1 SDK: `dotnet-sdk-3.1`

Package missing

If the package-version combination doesn't work, it's not available. For example, there isn't an ASP.NET Core SDK, the SDK components are included with the .NET SDK. The value `aspnetcore-sdk-2.2` is incorrect and should be `dotnet-sdk-2.2`. For a list of Linux distributions supported by .NET, see [.NET dependencies and requirements](#).

Troubleshoot the package manager

This section provides information on common errors you may get while using the package manager to install .NET.

Failed to fetch

While installing the .NET package, you may see an error similar to

```
signature verification failed for file 'repomd.xml' from repository 'packages-microsoft-com-prod'
```

Generally speaking, this error means that the package feed for .NET is being upgraded with newer package versions, and that you should try again later. During an upgrade, the package feed should not be unavailable for more than 2 hours. If you continually receive this error for more than 2 hours, please file an issue at <https://github.com/dotnet/core/issues>.

Dependencies

When you install with a package manager, these libraries are installed for you. But, if you manually install .NET or you publish a self-contained app, you'll need to make sure these libraries are installed:

- krb5
- libicu
- libopenssl1_1

If the target runtime environment's OpenSSL version is 1.1 or newer, you'll need to install `compat-openssl10`.

For more information about the dependencies, see [Self-contained Linux apps](#).

For .NET apps that use the `System.Drawing.Common` assembly, you'll also need the following dependency:

- [libgdiplus \(version 6.0.1 or later\)](#)

WARNING

You can install a recent version of `libgdiplus` by adding the Mono repository to your system. For more information, see <https://www.mono-project.com/download/stable/>.

Next steps

- [How to enable TAB completion for the .NET CLI](#)
- [Tutorial: Create a console application with .NET SDK using Visual Studio Code](#)

Install the .NET SDK or the .NET Runtime with Snap

9/20/2022 • 5 minutes to read • [Edit Online](#)

Use a Snap package to install the .NET SDK or .NET Runtime. Snaps are a great alternative to the package manager built into your Linux distribution. This article describes how to install .NET through [Snap](#).

A snap is a bundle of an app and its dependencies that works without modification across many different Linux distributions. Snaps are discoverable and installable from the Snap Store. For more information about Snap, see [Getting started with Snap](#).

Caution

Snap packages aren't supported in WSL2 on Windows 10. As an alternative, use the `dotnet-install` script or the package manager for the particular WSL2 distribution. It's not recommended but you can try to enable snap with an [unsupported workaround from the snapcraft forums](#).

.NET releases

Only ✓ supported versions of .NET SDK are available through Snap. All versions of the .NET Runtime are available through snap starting with version 2.1. The following table lists the .NET (and .NET Core) releases:

✓ SUPPORTED	✗ UNSUPPORTED
6 (LTS)	5
3.1 (LTS)	3.0
	2.2
	2.1
	2.0
	1.1
	1.0

For more information about the life cycle of .NET releases, see [.NET and .NET Core Support Policy](#).

SDK or Runtime

Install the SDK (which includes the runtime) if you want to develop .NET apps. Or, if you only need to run apps, install the Runtime. If you're installing the Runtime, we suggest you install the **ASP.NET Core Runtime** as it includes both .NET and ASP.NET Core runtimes.

If you've already installed the SDK or Runtime, use the `dotnet --list-sdks` and `dotnet --list-runtimes` commands to see which versions are installed. For more information, see [How to check that .NET is already installed](#).

Install the SDK

Snap packages for the .NET SDK are all published under the same identifier: `dotnet-sdk`. A specific version of

the SDK can be installed by specifying the channel. The SDK includes the corresponding runtime. The following table lists the channels:

.NET VERSION	SNAP PACKAGE OR CHANNEL
6 (LTS)	6.0 or latest/stable or lts/stable
5	5.0
3.1 (LTS)	3.1

Use the `snap install` command to install a .NET SDK snap package. Use the `--channel` parameter to indicate which version to install. If this parameter is omitted, `latest/stable` is used. In this example, `6.0` is specified:

```
sudo snap install dotnet-sdk --classic --channel=6.0
```

Next, register the `dotnet` command for the system with the `snap alias` command:

```
sudo snap alias dotnet-sdk.dotnet dotnet
```

This command is formatted as: `sudo snap alias {package}.{command} {alias}`. You can choose any `{alias}` name you would like. For example, you could name the command after the specific version installed by snap: `sudo snap alias dotnet-sdk.dotnet dotnet60`. When you use the command `dotnet60`, you'll invoke this specific version of .NET. But choosing a different alias is incompatible with most tutorials and examples as they expect a `dotnet` command to be used.

Install the runtime

Snap packages for the .NET Runtime are each published under their own package identifier. The following table lists the package identifiers:

.NET VERSION	SNAP PACKAGE
6 (LTS)	dotnet-runtime-60
5	dotnet-runtime-50
3.1 (LTS)	dotnet-runtime-31
3.0	dotnet-runtime-30
2.2	dotnet-runtime-22
2.1	dotnet-runtime-21

Use the `snap install` command to install a .NET Runtime snap package. In this example, .NET 6 is installed:

```
sudo snap install dotnet-runtime-60 --classic
```

Next, register the `dotnet` command for the system with the `snap alias` command:

```
sudo snap alias dotnet-runtime-60.dotnet dotnet
```

The command is formatted as: `sudo snap alias {package}.{command} {alias}`. You can choose any `{alias}` name you would like. For example, you could name the command after the specific version installed by snap: `sudo snap alias dotnet-runtime-60.dotnet dotnet60`. When you use the command `dotnet60`, you'll invoke a specific version of .NET. But choosing a different alias is incompatible with most tutorials and examples as they expect a `dotnet` command to be available.

Export the install location

The `DOTNET_ROOT` environment variable is often used by tools to determine where .NET is installed. When .NET is installed through Snap, this environment variable isn't configured. You should configure the `DOTNET_ROOT` environment variable in your profile. The path to the snap uses the following format: `/snap/{package}/current`. For example, if you installed the `dotnet-sdk` snap, use the following command to set the environment variable to where .NET is located:

```
export DOTNET_ROOT=/snap/dotnet-sdk/current
```

TIP

The preceding `export` command only sets the environment variable for the terminal session in which it was run.

You can edit your shell profile to permanently add the commands. There are a number of different shells available for Linux and each has a different profile. For example:

- **Bash Shell:** `~/.bash_profile`, `~/.bashrc`
- **Korn Shell:** `~/.kshrc` or `.profile`
- **Z Shell:** `~/.zshrc` or `.zprofile`

Edit the appropriate source file for your shell and add `export DOTNET_ROOT=/snap/dotnet-sdk/current`.

TLS/SSL Certificate errors

When .NET is installed through Snap, it's possible that on some distros the .NET TLS/SSL certificates may not be found and you may receive an error during `restore`:

```
Processing post-creation actions...
Running 'dotnet restore' on /home/myhome/test/test.csproj...
Restoring packages for /home/myhome/test/test.csproj...
/snap/dotnet-sdk/27/sdk/2.2.103/NuGet.targets(114,5): error : Unable to load the service index for source
https://api.nuget.org/v3/index.json. [/home/myhome/test/test.csproj]
/snap/dotnet-sdk/27/sdk/2.2.103/NuGet.targets(114,5): error : The SSL connection could not be established,
see inner exception. [/home/myhome/test/test.csproj]
/snap/dotnet-sdk/27/sdk/2.2.103/NuGet.targets(114,5): error : The remote certificate is invalid according
to the validation procedure. [/home/myhome/test/test.csproj]
```

To resolve this problem, set a few environment variables:

```
export SSL_CERT_FILE=[path-to-certificate-file]
export SSL_CERT_DIR=/dev/null
```

The certificate location will vary by distro. Here are the locations for the distros where the issue has been experienced.

DISTRIBUTION	LOCATION
Fedora	/etc/pki/ca-trust/extracted/pem/tls-ca-bundle.pem
OpenSUSE	/etc/ssl/ca-bundle.pem
Solus	/etc/ssl/certs/ca-certificates.crt

Troubles resolving dotnet

It's common for other apps, such as the OmniSharp extension for Visual Studio Code, to try to resolve the location of the .NET SDK. Typically, this is done by figuring out where the `dotnet` executable is located. A snap-installed .NET SDK may confuse these apps. When these apps can't resolve the .NET SDK, you'll see an error similar to one of the following messages:

- The SDK 'Microsoft.NET.Sdk' specified could not be found
- The SDK 'Microsoft.NET.Sdk.Web' specified could not be found
- The SDK 'Microsoft.NET.Sdk.Razor' specified could not be found

To fix this problem, symlink the snap `dotnet` executable to the location that the program is looking for. Two common paths the `dotnet` command is looking for are `/usr/local/bin/dotnet` and `/usr/share/dotnet`. For example, to link the current .NET SDK snap package, use the following command:

```
ln -s /snap/dotnet-sdk/current/dotnet /usr/local/bin/dotnet
```

You can also review these GitHub issues for information about these problems:

- [SDK resolver doesn't work with snap installations of SDK on Linux](#)
- [It wasn't possible to find any installed .NET SDKs](#)

The `dotnet` alias

It's possible that if you created the `dotnet` alias for the snap-installed .NET, you'll have a conflict. Use the `snap unalias dotnet` command to remove it, and then add a different alias if you want.

Next steps

- [How to enable TAB completion for the .NET CLI](#)
- [Tutorial: Create a console application with .NET SDK using Visual Studio Code](#)

Install .NET on Linux by using an install script or by extracting binaries

9/20/2022 • 4 minutes to read • [Edit Online](#)

This article demonstrates how to install the .NET SDK or the .NET Runtime on Linux by using the install script or by extracting the binaries. For a list of distributions that support the built-in package manager, see [Install .NET on Linux](#).

You can also install .NET with snap. For more information, see [Install the .NET SDK or the .NET Runtime with Snap](#).

Install the SDK (which includes the runtime) if you want to develop .NET apps. Or, if you only need to run apps, install the Runtime. If you're installing the Runtime, we suggest you install the **ASP.NET Core Runtime** as it includes both .NET and ASP.NET Core runtimes.

If you've already installed the SDK or Runtime, use the `dotnet --list-sdks` and `dotnet --list-runtimes` commands to see which versions are installed. For more information, see [How to check that .NET is already installed](#).

.NET releases

The following table lists the .NET (and .NET Core) releases:

✓ SUPPORTED	✗ UNSUPPORTED
6 (LTS)	5
3.1 (LTS)	3.0
	2.2
	2.1
	2.0
	1.1
	1.0

For more information about the life cycle of .NET releases, see [.NET and .NET Core Support Policy](#).

Dependencies

It's possible that when you install .NET, specific dependencies may not be installed, such as when [manually installing](#). The following list details Linux distributions that are supported by Microsoft and have dependencies you may need to install. Check the distribution page for more information:

- [Alpine](#)
- [Debian](#)
- [CentOS](#)

- [Fedora](#)
- [RHEL and CentOS Stream](#)
- [SLES](#)
- [Ubuntu](#)

For generic information about the dependencies, see [Self-contained Linux apps](#).

RPM dependencies

If your distribution wasn't previously listed, and is RPM-based, you may need the following dependencies:

- krb5-libs
- libicu
- openssl-libs

If the target runtime environment's OpenSSL version is 1.1 or newer, you'll need to install `compat-openssl10`.

DEB dependencies

If your distribution wasn't previously listed, and is debian-based, you may need the following dependencies:

- libc6
- libgcc1
- libgssapi-krb5-2
- libicu67
- libssl1.1
- libstdc++6
- zlib1g

Common dependencies

For .NET apps that use the `System.Drawing.Common` assembly, you'll also need the following dependency:

- [libgdipplus \(version 6.0.1 or later\)](#)

WARNING

You can install a recent version of `libgdipplus` by adding the Mono repository to your system. For more information, see <https://www.monoproject.com/download/stable/>.

Scripted install

The [dotnet-install scripts](#) are used for automation and non-admin installs of the **SDK** and **Runtime**. You can download the script from <https://dot.net/v1/dotnet-install.sh>.

IMPORTANT

Bash is required to run the script.

Before running this script, you'll need to grant permission for this script to run as an executable:

```
sudo chmod +x ./dotnet-install.sh
```

The script defaults to installing the latest SDK [long term support \(LTS\)](#) version, which is .NET 6. To install the current release, which may not be an (LTS) version, use the `-c current` parameter.

```
./dotnet-install.sh -c Current
```

To install .NET Runtime instead of the SDK, use the `--runtime` parameter.

```
./dotnet-install.sh -c Current --runtime aspnetcore
```

You can install a specific version by altering the `-c` parameter to indicate the specific version. The following command installs .NET SDK 6.0.

```
./dotnet-install.sh -c 6.0
```

For more information, see [dotnet-install scripts reference](#).

Manual install

As an alternative to the package managers, you can download and manually install the SDK and runtime. Manual installation is commonly used as part of continuous integration testing or on an unsupported Linux distribution. For a developer or user, it's better to use a package manager.

First, download a **binary** release for either the SDK or the runtime from one of the following sites. If you install the .NET SDK, you will not need to install the corresponding runtime:

- ✓ [.NET 6 downloads](#)
- ✓ [.NET Core 3.1 downloads](#)
- [All .NET Core downloads](#)

Next, extract the downloaded file and use the `export` command to set `DOTNET_ROOT` to the extracted folder's location and then ensure .NET is in PATH. This should make the .NET CLI commands available at the terminal.

Alternatively, after downloading the .NET binary, the following commands may be run from the directory where the file is saved to extract the runtime. This will also make the .NET CLI commands available at the terminal and set the required environment variables. **Remember to change the `DOTNET_FILE` value to the name of the downloaded binary:**

```
DOTNET_FILE=dotnet-sdk-6.0.100-linux-x64.tar.gz
export DOTNET_ROOT=$(pwd)/.dotnet

mkdir -p "$DOTNET_ROOT" && tar zxf "$DOTNET_FILE" -C "$DOTNET_ROOT"

export PATH=$PATH:$DOTNET_ROOT:$DOTNET_ROOT/tools
```

This approach lets you install different versions into separate locations and choose explicitly which one to use by which application.

Set environment variables system-wide

If you used the previous install script, the variables set only apply to your current terminal session. Add them to your shell profile. There are a number of different shells available for Linux and each has a different profile. For example:

- **Bash Shell:** `~/.bash_profile`, `~/.bashrc`
- **Korn Shell:** `~/.kshrc` or `.profile`
- **Z Shell:** `~/.zshrc` or `.zprofile`

Set the following two environment variables in your shell profile:

- `DOTNET_ROOT`

This variable is set to the folder .NET was installed to, such as `$HOME/.dotnet`:

```
export DOTNET_ROOT=$HOME/.dotnet
```

- `PATH`

This variable should include both the `DOTNET_ROOT` folder and the user's `.dotnet/tools` folder:

```
export PATH=$PATH:$HOME/.dotnet:$HOME/.dotnet/tools
```

Next steps

- [How to enable TAB completion for the .NET CLI](#)
- [Tutorial: Create a console application with .NET SDK using Visual Studio Code](#)

How to remove the .NET Runtime and SDK

9/20/2022 • 6 minutes to read • [Edit Online](#)

Over time, as you install updated versions of the .NET runtime and SDK, you may want to remove outdated versions of .NET from your machine. Removing older versions of the runtime may change the runtime chosen to run shared framework applications, as detailed in the article on [.NET version selection](#).

Should I remove a version?

The [.NET version selection](#) behaviors and the runtime compatibility of .NET across updates enables safe removal of previous versions. .NET runtime updates are compatible within a major version **band** such as 6.x and 5.x. Additionally, newer releases of the .NET SDK generally maintain the ability to build applications that target previous versions of the runtime in a compatible manner.

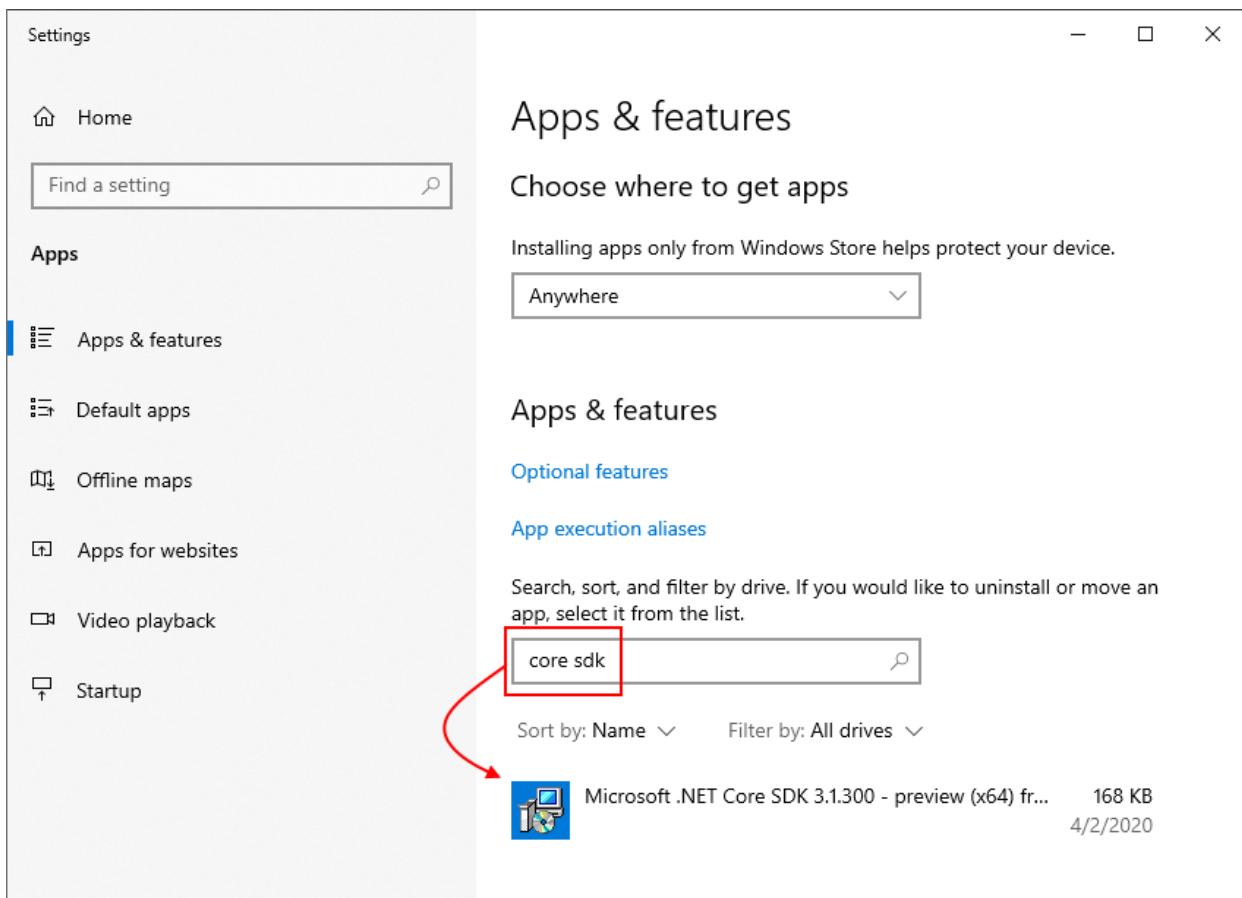
In general, you only need the latest SDK and latest patch version of the runtimes required for your application. Instances where you might want to keep older SDK or runtime versions include maintaining *project.json*-based applications. Unless your application has specific reasons for earlier SDKs or runtimes, you may safely remove older versions.

Determine what is installed

The .NET CLI has options you can use to list the versions of the SDK and runtime that are installed on your computer. Use `dotnet --list-sdks` to see the list of installed SDKs and `dotnet --list-runtimes` for the list of runtimes. For more information, see [How to check that .NET is already installed](#).

Uninstall .NET

.NET uses the Windows Apps & features dialog to remove versions of the .NET runtime and SDK. The following figure shows the Apps & features dialog. You can search for **core** or **.net** to filter and show installed versions of .NET.



Select any versions you want to remove from your computer and click **Uninstall**.

The best way for you to uninstall .NET is to mirror the action you used to install .NET. The specifics depend on your chosen Linux distribution and the installation method.

IMPORTANT

For Red Hat installations, consult the [Red Hat Product Documentation for .NET](#).

There's no need to first uninstall the .NET SDK when upgrading it using a package manager, unless you're upgrading from a preview version that was manually installed. The package manager `update` or `refresh` commands will automatically remove the older version upon the successful installation of a newer version. If you have a preview version installed, uninstall it.

If you installed .NET using a package manager, use that same package manager to uninstall the .NET SDK or runtime. .NET installations support most popular package managers. Consult the documentation for your distribution's package manager for the precise syntax in your environment:

- `apt-get(8)` is used by Debian based systems, including Ubuntu.
- `yum(8)` is used on Fedora, CentOS, and Oracle Linux.
- `zypper(8)` is used on openSUSE and SUSE Linux Enterprise System (SLES).
- `dnf(8)` is used on Fedora.

In almost all cases, the command to remove a package is `remove`.

The package name for the .NET SDK installation for most package managers is `dotnet-sdk`, followed by the version number. Starting with the version 2.1.300 of the .NET SDK and version 2.1 of the runtime, only the major and minor version numbers are necessary: for example, the .NET SDK version 2.1.300 can be referenced as the package `dotnet-sdk-2.1`. Prior versions require the entire version string: for example, `dotnet-sdk-2.1.200` would be required for version 2.1.200 of the .NET SDK.

For machines that have installed only the runtime, and not the SDK, the package name is `dotnet-runtime-<version>` for the .NET runtime, and `aspnetcore-runtime-<version>` for the entire runtime stack.

TIP

.NET Core installations earlier than 2.0 didn't uninstall the host application when the SDK was uninstalled using the package manager. Using `apt-get`, the command is:

```
apt-get remove dotnet-host
```

There's no version attached to `dotnet-host`.

If you installed using a tarball, you must remove .NET using the manual method.

On Linux, you must remove the SDKs and runtimes separately, by removing the versioned directories. These directories may vary depending on your Linux distribution. Removing them deletes the SDK and runtime from disk. For example, to remove the 1.0.1 SDK and runtime, you would use the following bash commands:

```
version="1.0.1"
sudo rm -rf /usr/share/dotnet/sdk/$version
sudo rm -rf /usr/share/dotnet/shared/Microsoft.NETCore.App/$version
sudo rm -rf /usr/share/dotnet/shared/Microsoft.AspNetCore.All/$version
sudo rm -rf /usr/share/dotnet/shared/Microsoft.AspNetCore.App/$version
sudo rm -rf /usr/share/dotnet/host/fxr/$version
```

IMPORTANT

The version folders may not match the "version" you're uninstalling. The individual runtimes and SDKs that are installed with a single .NET release may have different versions. For example, you may have installed ASP.NET Core 5 Runtime, which installed the 5.0.2 ASP.NET Core runtime and the 5.0.8 .NET runtime. Each has a different versioned folder. For more information, see [Overview of how .NET is versioned](#).

The parent directories for the SDK and runtime are listed in the output from the `dotnet --list-sdks` and `dotnet --list-runtimes` command, as shown in the earlier table.

On Mac, you must remove the SDKs and runtimes separately, by removing the versioned directories. Removing them deletes the SDK and runtime from disk. For example, to remove the 1.0.1 SDK and runtime, you would use the following bash commands:

```
version="1.0.1"
sudo rm -rf /usr/local/share/dotnet/sdk/$version
sudo rm -rf /usr/local/share/dotnet/shared/Microsoft.NETCore.App/$version
sudo rm -rf /usr/local/share/dotnet/shared/Microsoft.AspNetCore.All/$version
sudo rm -rf /usr/local/share/dotnet/shared/Microsoft.AspNetCore.App/$version
sudo rm -rf /usr/local/share/dotnet/host/fxr/$version
```

IMPORTANT

The version folders may not match the "version" you're uninstalling. The individual runtimes and SDKs that are installed with .NET may have different versions. For example, you may have installed .NET 5 Runtime, which installed the 5.0.2 ASP.NET Core runtime and the 5.0.8 .NET runtime. For more information, see [Overview of how .NET is versioned](#).

IMPORTANT

If you're using an Arm-based Mac, such as one with an M1 chip, review the folder paths described in [Install .NET on Arm-based Macs](#).

The parent directories for the SDK and runtime are listed in the output from the `dotnet --list-sdks` and `dotnet --list-runtimes` command, as shown in the earlier table.

.NET Uninstall Tool

The [.NET Uninstall Tool](#) (`dotnet-core-uninstall`) lets you remove .NET SDKs and runtimes from a system. A collection of options is available to specify which versions should be uninstalled.

Visual Studio dependency on .NET SDK versions

Before Visual Studio 2019 version 16.3, Visual Studio installers called the standalone SDK installer for .NET Core version 2.1 or 2.2. As a result, the SDK versions appear in the Windows **Apps & features** dialog. Removing .NET SDKs that were installed by Visual Studio using the standalone installer may break Visual Studio. If Visual Studio has problems after you uninstall SDKs, run Repair on that specific version of Visual Studio. The following table shows some of the Visual Studio dependencies on .NET Core SDK versions:

VISUAL STUDIO VERSION	.NET CORE SDK VERSION
Visual Studio 2019 version 16.2	.NET Core SDK 2.2.4xx, 2.1.8xx
Visual Studio 2019 version 16.1	.NET Core SDK 2.2.3xx, 2.1.7xx
Visual Studio 2019 version 16.0	.NET Core SDK 2.2.2xx, 2.1.6xx
Visual Studio 2017 version 15.9	.NET Core SDK 2.2.1xx, 2.1.5xx
Visual Studio 2017 version 15.8	.NET Core SDK 2.1.4xx

Starting with Visual Studio 2019 version 16.3, Visual Studio is in charge of its own copy of the .NET SDK. For that reason, you no longer see those SDK versions in the **Apps & features** dialog.

Remove the NuGet fallback folder

Before .NET Core 3.0 SDK, the .NET Core SDK installers used a folder named *NuGetFallbackFolder* to store a cache of NuGet packages. This cache was used during operations such as `dotnet restore` or `dotnet build /t:Restore`. The *NuGetFallbackFolder* is located at `C:\Program Files\dotnet\sdk` on Windows and at `/usr/local/share/dotnet/sdk` on macOS.

You may want to remove this folder, if:

- You're only developing using .NET Core 3.0 SDK or .NET 5 or later versions.
- You're developing using .NET Core SDK versions earlier than 3.0, but you can work online.

If you want to remove the NuGet fallback folder, you can delete it, but you'll need administrative privileges to do so.

It's not recommended to delete the *dotnet* folder. Doing so would remove any global tools you've previously installed. Also, on Windows:

- You'll break Visual Studio 2019 version 16.3 and later versions. You can run **Repair** to recover.
- If there are .NET Core SDK entries in the **Apps & features** dialog, they'll be orphaned.

Manage .NET project and item templates

9/20/2022 • 4 minutes to read • [Edit Online](#)

.NET Core provides a template system that enables users to install or uninstall templates from NuGet, a NuGet package file, or a file system directory. This article describes how to manage .NET Core templates through the .NET SDK CLI.

For more information about creating templates, see [Tutorial: Create templates](#).

Install template

Templates are installed through the `dotnet new` SDK command with the `-i` parameter. You can either provide the NuGet package identifier of a template, or a folder that contains the template files.

NuGet hosted package

.NET CLI templates are uploaded to [NuGet](#) for wide distribution. Templates can also be installed from a private feed. Instead of uploading a template to a NuGet feed, *nupkg* template files can be distributed and manually installed, as described in the [Local NuGet package](#) section.

For more information about configuring NuGet feeds, see [dotnet nuget add source](#).

To install a template pack from the default NuGet feed, use the `dotnet new -i {package-id}` command:

```
dotnet new -i Microsoft.DotNet.Web.Spa.ProjectTemplates
```

To install a template pack from the default NuGet feed with a specific version, use the

`dotnet new -i {package-id}::{version}` command:

```
dotnet new -i Microsoft.DotNet.Web.Spa.ProjectTemplates::2.2.6
```

Local NuGet package

When a template pack is created, a *nupkg* file is generated. If you have a *nupkg* file containing templates, you can install it with the `dotnet new -i {path-to-package}` command:

```
dotnet new -i c:\code\NuGet-Packages\Some.Templates.1.0.0.nupkg
```

```
dotnet new -i ~/code/NuGet-Packages/Some.Templates.1.0.0.nupkg
```

Folder

As an alternative to installing template from a *nupkg* file, you can also install templates from a folder directly with the `dotnet new -i {folder-path}` command. The folder specified is treated as the template pack identifier for any template found. Any template found in the specified folder's hierarchy is installed.

```
dotnet new -i c:\code\NuGet-Packages\some-folder\
```

```
dotnet new -i ~/code/NuGet-Packages/some-folder/
```

The `{folder-path}` specified on the command becomes the template pack identifier for all templates found. As specified in the [List templates](#) section, you can get a list of templates installed with the `dotnet new -u` command. In this example, the template pack identifier is shown as the folder used for install:

```
dotnet new -u
Template Instantiation Commands for .NET CLI

Currently installed items:

... cut to save space ...

c:\code\NuGet-Packages\some-folder
Templates:
A Template Console Class (templateconsole) C#
Project for some technology (contosoproject) C#
Uninstall Command:
dotnet new -u c:\code\NuGet-Packages\some-folder
```

```
dotnet new -u
Template Instantiation Commands for .NET CLI

Currently installed items:

... cut to save space ...

/home/username/code/templates
Templates:
A Template Console Class (templateconsole) C#
Project for some technology (contosoproject) C#
Uninstall Command:
dotnet new -u /home/username/code/templates
```

Uninstall template

Templates are uninstalled through the `dotnet new` SDK command with the `-u` parameter. You can either provide the NuGet package identifier of a template, or a folder that contains the template files.

NuGet package

After a NuGet template pack is installed, either from a NuGet feed or a `nupkg` file, you can uninstall it by referencing the NuGet package identifier.

To uninstall a template pack, use the `dotnet new -u {package-id}` command:

```
dotnet new -u Microsoft.DotNet.Web.Spa.ProjectTemplates
```

Folder

When templates are installed through a [folder path](#), the folder path becomes the template pack identifier.

To uninstall a template pack, use the `dotnet new -u {package-folder-path}` command:

```
dotnet new -u c:\code\NuGet-Packages\some-folder
```

```
dotnet new -u /home/username/code/templates
```

List templates

By using the standard `uninstall` command without a package identifier, you can see a list of installed templates along with the command that uninstalls each template.

```
dotnet new -u
Template Instantiation Commands for .NET CLI

Currently installed items:

... cut to save space ...

c:\code\NuGet-Packages\some-folder
Templates:
A Template Console Class (templateconsole) C#
Project for some technology (contosoproject) C#
Uninstall Command:
dotnet new -u c:\code\NuGet-Packages\some-folder
```

Install templates from other SDKs

If you've installed each version of the SDK sequentially, for example you installed SDK 2.0, then SDK 2.1, and so on, you'll have every SDK's templates installed. However, if you start with a later SDK version, like 3.1, only the templates for [LTS \(long term support\) releases](#) are included, which at the time of the SDK 3.1 release is SDK 2.1 and SDK 3.1. Templates for any other release aren't included.

The .NET Core templates are available on NuGet, and you can install them like any other template. For more information, see [Install NuGet hosted package](#).

SDK	NUGET PACKAGE IDENTIFIER
.NET Core 2.1	Microsoft.DotNet.Common.ProjectTemplates.2.1
.NET Core 2.2	Microsoft.DotNet.Common.ProjectTemplates.2.2
.NET Core 3.0	Microsoft.DotNet.Common.ProjectTemplates.3.0
.NET Core 3.1	Microsoft.DotNet.Common.ProjectTemplates.3.1
.NET 5.0	Microsoft.DotNet.Common.ProjectTemplates.5.0
.NET 6.0	Microsoft.DotNet.Common.ProjectTemplates.6.0
ASP.NET Core 2.1	Microsoft.DotNet.Web.ProjectTemplates.2.1
ASP.NET Core 2.2	Microsoft.DotNet.Web.ProjectTemplates.2.2
ASP.NET Core 3.0	Microsoft.DotNet.Web.ProjectTemplates.3.0
ASP.NET Core 3.1	Microsoft.DotNet.Web.ProjectTemplates.3.1
ASP.NET Core 5.0	Microsoft.DotNet.Web.ProjectTemplates.5.0
ASP.NET Core 6.0	Microsoft.DotNet.Web.ProjectTemplates.6.0

For example, the .NET Core SDK includes templates for a console app targeting .NET Core 2.1 and .NET Core 3.1. If you wanted to target .NET Core 3.0, you would need to install the 3.0 templates.

1. Try creating an app that targets .NET Core 3.0.

```
dotnet new console --framework netcoreapp3.0
```

If you see an error message, you need to install the templates.

Couldn't find an installed template that matches the input, searching online for one that does...

2. Install the .NET Core 3.0 project templates.

```
dotnet new -i Microsoft.DotNet.Common.ProjectTemplates.3.0
```

3. Try creating the app a second time.

```
dotnet new console --framework netcoreapp3.0
```

And you should see a message indicating the project was created.

The template "Console Application" was created successfully.

Processing post-creation actions... Running 'dotnet restore' on path-to-project-file.csproj...

Determining projects to restore... Restore completed in 1.05 sec for path-to-project-file.csproj.

Restore succeeded.

See also

- [Tutorial: Create an item template](#)
- [dotnet new](#)
- [dotnet nuget add source](#)

macOS Catalina Notarization and the impact on .NET downloads and projects

9/20/2022 • 4 minutes to read • [Edit Online](#)

Beginning with macOS Catalina (version 10.15), all software built after June 1, 2019, and distributed with Developer ID, must be notarized. This requirement applies to the .NET runtime, .NET SDK, and software created with .NET. This article describes the common scenarios you may face with .NET and macOS notarization.

Installing .NET

The installers for .NET (both runtime and SDK) have been notarized since February 18, 2020. Prior released versions aren't notarized. You can manually install a non-notarized version of .NET by first downloading the installer, and then using the `sudo installer` command. For more information, see [Download and manually install for macOS](#).

Native appHost

In .NET SDK 7 and later versions, an **appHost**, which is a native Mach-O executable, is produced for your app. This executable is usually invoked by .NET when your project compiles, publishes, or is run with the `dotnet run` command. The non-**appHost** version of your app is a *dll* file that can be invoked by the `dotnet <app.dll>` command.

When run locally, the SDK signs the apphost using [ad hoc signing](#), which allows the app to run locally. When distributing your app, you'll need to properly sign your app according to Apple guidance.

You can also distribute your app without the apphost and rely on users to run your app using `dotnet`. To turn off **appHost** generation, add the `UseAppHost` boolean setting in the project file and set it to `false`. You can also toggle the appHost with the `-p:UseAppHost` parameter on the command line for the specific `dotnet` command you run:

- Project file

```
<PropertyGroup>
  <UseAppHost>false</UseAppHost>
</PropertyGroup>
```

- Command-line parameter

```
dotnet run -p:UseAppHost=false
```

An **appHost** is required when you publish your app [self-contained](#) and you cannot disable it.

For more information about the `UseAppHost` setting, see [MSBuild properties for Microsoft.NET.Sdk](#).

Context of the appHost

When the appHost is enabled in your project, and you use the `dotnet run` command to run your app, the app is invoked in the context of the appHost and not the default host (the default host is the `dotnet` command). If the appHost is disabled in your project, the `dotnet run` command runs your app in the context of the default host. Even if the appHost is disabled, publishing your app as self-contained generates an appHost executable, and

users use that executable to run your app. Running your app with `dotnet <filename.dll>` invokes the app with the default host, the shared runtime.

When an app using the appHost is invoked, the certificate partition accessed by the app is different from the notarized default host. If your app must access the certificates installed through the default host, use the `dotnet run` command to run your app from its project file, or use the `dotnet <filename.dll>` command to start the app directly.

More information about this scenario is provided in the [ASP.NET Core and macOS and certificates](#) section.

ASP.NET Core, macOS, and certificates

.NET provides the ability to manage certificates in the macOS Keychain with the `System.Security.Cryptography.X509Certificates` class. Access to the macOS Keychain uses the application's identity as the primary key when deciding which partition to consider. For example, unsigned applications store secrets in the unsigned partition, but signed applications store their secrets in partitions only they can access. The source of execution that invokes your app decides which partition to use.

.NET provides three sources of execution: `appHost`, default host (the `dotnet` command), and a custom host. Each execution model may have different identities, either signed or unsigned, and has access to different partitions within the Keychain. Certificates imported by one mode may not be accessible from another. For example, the notarized versions of .NET have a default host that is signed. Certificates are imported into a secure partition based on its identity. These certificates aren't accessible from a generated appHost, as the appHost is ad-hoc signed.

Another example, by default, ASP.NET Core imports a default SSL certificate through the default host. ASP.NET Core applications that use an appHost won't have access to this certificate and will receive an error when .NET detects the certificate isn't accessible. The error message provides instructions on how to fix this problem.

If certificate sharing is required, macOS provides configuration options with the `security` utility.

For more information on how to troubleshoot ASP.NET Core certificate issues, see [Enforce HTTPS in ASP.NET Core](#).

Default entitlements

.NET's default host (the `dotnet` command) has a set of default entitlements. These entitlements are required for proper operation of .NET. It's possible that your application may need additional entitlements, in which case you'll need to generate and use an `appHost` and then add the necessary entitlements locally.

Default set of entitlements for .NET:

- `com.apple.security.cs.allow-jit`
- `com.apple.security.cs.allow-unsigned-executable-memory`
- `com.apple.security.cs.allow-dyld-environment-variables`
- `com.apple.security.cs.disable-library-validation`

Notarize a .NET app

If you want your application to run on macOS Catalina (version 10.15) or higher, you'll want to notarize your app. The appHost you submit with your application for notarization should be used with at least the same [default entitlements](#) for .NET Core.

Next steps

- [Install .NET on macOS](#).

Troubleshoot *fxr*, *libhostfxr.so*, and *FrameworkList.xml* errors

9/20/2022 • 3 minutes to read • [Edit Online](#)

When you try to use .NET 5+ (and .NET Core), commands such as `dotnet new` and `dotnet run` may fail with a message related to something not being found. Some of the error messages may be similar to the following:

- **System.IO.FileNotFoundException**

```
System.IO.FileNotFoundException: Could not find file  
'/usr/share/dotnet/packs/Microsoft.NETCore.App.Ref/5.0.0/data/FrameworkList.xml'.
```

- **A fatal error occurred.**

```
A fatal error occurred. The required library libhostfxr.so could not be found.
```

or

```
A fatal error occurred. The folder [/usr/share/dotnet/host/fxr] does not exist.
```

or

```
A fatal error occurred, the folder [/usr/share/dotnet/host/fxr] does not contain any version-numbered  
child folders.
```

- **Generic messages about dotnet not found**

A general message may appear that indicates the SDK isn't found, or that the package has already been installed.

One symptom of these problems is that both the `/usr/lib64/dotnet` and `/usr/share/dotnet` folders are on your system.

What's going on

This generally happens when two Linux package repositories provide .NET packages. While Microsoft provides a Linux package repository to source .NET packages, some Linux distributions also provide .NET packages, such as:

- Arch
- CentOS
- CentOS Stream
- Fedora
- RHEL

Mixing .NET packages from two different sources will most likely lead to issues since the packages may place things at different paths, and may be compiled differently.

Solutions

The solution to these problems is to use .NET from one package repository. Which repository to pick, and how to do it, varies by use-case and the Linux distribution.

If your distribution provides .NET packages, it's recommended that you use that package repository instead of Microsoft's.

- 1. I only use .NET and no other packages from the Microsoft repository, and my distribution provides .NET packages.**

If you only use the Microsoft repository for .NET packages and not for any other Microsoft package such as `mdatp`, `powershell`, or `mssql`, then:

- a. Remove the Microsoft repository
- b. Remove the .NET related packages from your OS
- c. Install the .NET packages from the distribution repository

For Fedora, CentOS 8+, RHEL 8+, use the following bash commands:

```
sudo dnf remove packages-microsoft-prod
sudo dnf remove 'dotnet*' 'aspnet*' 'netstandard*'
sudo dnf install dotnet-sdk-5.0
```

- 2. I want to use the distribution provided .NET packages, but I also use the Microsoft repository for other packages.**

If you use the Microsoft repository for Microsoft packages such as `mdatp`, `powershell`, or `mssql`, but you don't want to use the repository for .NET, then:

- a. Configure the Microsoft repository to exclude any .NET package
- b. Remove the .NET related packages from your OS
- c. Install the .NET packages from the distribution repository

For Fedora, CentOS 8+, RHEL 8+, use the following bash commands:

```
echo 'excludepkgs=dotnet*,aspnet*,netstandard*' | sudo tee -a /etc/yum.repos.d/microsoft-prod.repo
sudo dnf remove 'dotnet*' 'aspnet*' 'netstandard*'
sudo dnf install dotnet-sdk-5.0
```

- 3. I need a recent version of .NET that's not provided by the Linux distribution repositories.**

In this case, keep the Microsoft repository, but configure it so .NET packages from the Microsoft repository are considered a higher priority. Then, remove the already-installed .NET packages and then re-install the .NET packages from the Microsoft repository.

For Fedora, CentOS 8+, RHEL 8+, use the following bash commands:

```
echo 'priority=50' | sudo tee -a /etc/yum.repos.d/microsoft-prod.repo
sudo dnf remove 'dotnet*' 'aspnet*' 'netstandard*'
sudo dnf install dotnet-sdk-5.0
```

- 4. I've encountered a bug in the Linux distribution version of .NET, I need the latest Microsoft version.**

Use solution 3 to solve this problem.

Online references

Many of these problems have been reported by users such as yourself. The following is a list of those issues. You can read through them for insights on what may be happening:

- System.IO.FileNotFoundException and
'/usr/share/dotnet/packs/Microsoft.NETCore.App.Ref/5.0.0/data/FrameworkList.xml'
 - [SDK #15785: unable to build brand new project after upgrading to 5.0.3](#)
 - [SDK #15863: "MSB4018 ResolveTargetingPackAssets task failed unexpectedly" after updating to 5.0.103](#)
 - [SDK #17411: dotnet build always throwing error](#)
 - [SDK #12075: dotnet 3.1.301 on Fedora 32 unable to find FrameworkList.xml because it doesn't exist](#)
- Fatal error: */libhostfxr.so* couldn't be found
 - [SDK #17570: After updating Fedora 33 to 34 and dotnet 5.0.5 to 5.0.6 I get error regarding libhostfxr.so:](#)
- Fatal error: folder */host/fxr* doesn't exist
 - [Core #5746: The folder does not exist when installing 3.1 on CentOS 8 with packages.microsoft.com repo enabled](#)
 - [SDK #15476: A fatal error occurred. The folder \[/usr/share/dotnet/host/fxr\] does not exist:](#)
- Fatal error: folder */host/fxr* doesn't contain any version-numbered child folders
 - [Installer #9254: Error when install dotnet/core/aspnet:3.1 on CentOS 8 - Folder does not contain any version-numbered child folders](#)
 - [StackOverflow: Error when install dotnet/core/aspnet:3.1 on CentOS 8 - Folder does not contain any version-numbered child folders](#)
- Generic errors without clear messages
 - [Core #4605: cannot run "dotnet new console"](#)
 - [Core #4644: Cannot install .NET Core SDK 2.1 on Fedora 32](#)
 - [Runtime #49375: After updating to 5.0.200-1 using package manager, it appears that no sdks are installed](#)

Next steps

- [Install .NET on Linux](#)
- [How to remove the .NET Runtime and SDK](#)
- [Tutorial: Create a new app with Visual Studio Code.](#)
- [Tutorial: Containerize a .NET app.](#)

How to check that .NET is already installed

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article teaches you how to check which versions of the .NET runtime and SDK are installed on your computer. If you have an integrated development environment, such as Visual Studio or Visual Studio for Mac, .NET may have already been installed.

Installing an SDK installs the corresponding runtime.

If any command in this article fails, you don't have the runtime or SDK installed. For more information, see the install articles for [Windows](#), [macOS](#), or [Linux](#).

Check SDK versions

You can see which versions of the .NET SDK are currently installed with a terminal. Open a terminal and run the following command.

```
dotnet --list-sdks
```

You get output similar to the following.

```
2.1.500 [C:\program files\dotnet\sdk]
2.1.502 [C:\program files\dotnet\sdk]
2.1.504 [C:\program files\dotnet\sdk]
2.1.600 [C:\program files\dotnet\sdk]
2.1.602 [C:\program files\dotnet\sdk]
3.1.100 [C:\program files\dotnet\sdk]
5.0.100 [C:\program files\dotnet\sdk]
6.0.100 [C:\program files\dotnet\sdk]
```

```
2.1.500 [/home/user/dotnet/sdk]
2.1.502 [/home/user/dotnet/sdk]
2.1.504 [/home/user/dotnet/sdk]
2.1.600 [/home/user/dotnet/sdk]
2.1.602 [/home/user/dotnet/sdk]
3.1.100 [/home/user/dotnet/sdk]
5.0.100 [/home/user/dotnet/sdk]
6.0.100 [/home/user/dotnet/sdk]
```

```
2.1.500 [/usr/local/share/dotnet/sdk]
2.1.502 [/usr/local/share/dotnet/sdk]
2.1.504 [/usr/local/share/dotnet/sdk]
2.1.600 [/usr/local/share/dotnet/sdk]
2.1.602 [/usr/local/share/dotnet/sdk]
3.1.100 [/usr/local/share/dotnet/sdk]
5.0.100 [/usr/local/share/dotnet/sdk]
6.0.100 [/usr/local/share/dotnet/sdk]
```

Check runtime versions

You can see which versions of the .NET runtime are currently installed with the following command.

```
dotnet --list-runtimes
```

You get output similar to the following.

```
Microsoft.AspNetCore.All 2.1.7 [c:\program files\dotnet\shared\Microsoft.AspNetCore.All]
Microsoft.AspNetCore.All 2.1.13 [c:\program files\dotnet\shared\Microsoft.AspNetCore.All]
Microsoft.AspNetCore.App 2.1.7 [c:\program files\dotnet\shared\Microsoft.AspNetCore.App]
Microsoft.AspNetCore.App 2.1.13 [c:\program files\dotnet\shared\Microsoft.AspNetCore.App]
Microsoft.AspNetCore.App 3.1.0 [c:\program files\dotnet\shared\Microsoft.AspNetCore.App]
Microsoft.AspNetCore.App 5.0.0 [c:\program files\dotnet\shared\Microsoft.AspNetCore.App]
Microsoft.AspNetCore.App 6.0.0 [c:\program files\dotnet\shared\Microsoft.AspNetCore.App]
Microsoft.NETCore.App 2.1.7 [c:\program files\dotnet\shared\Microsoft.NETCore.App]
Microsoft.NETCore.App 2.1.13 [c:\program files\dotnet\shared\Microsoft.NETCore.App]
Microsoft.NETCore.App 3.1.0 [c:\program files\dotnet\shared\Microsoft.NETCore.App]
Microsoft.NETCore.App 5.0.0 [c:\program files\dotnet\shared\Microsoft.NETCore.App]
Microsoft.NETCore.App 6.0.0 [c:\program files\dotnet\shared\Microsoft.NETCore.App]
Microsoft.WindowsDesktop.App 3.0.0 [c:\program files\dotnet\shared\Microsoft.WindowsDesktop.App]
Microsoft.WindowsDesktop.App 3.1.0 [c:\program files\dotnet\shared\Microsoft.WindowsDesktop.App]
Microsoft.WindowsDesktop.App 5.0.0 [c:\program files\dotnet\shared\Microsoft.WindowsDesktop.App]
Microsoft.WindowsDesktop.App 6.0.0 [c:\program files\dotnet\shared\Microsoft.WindowsDesktop.App]
```

```
Microsoft.AspNetCore.All 2.1.7 [/home/user/dotnet/shared/Microsoft.AspNetCore.All]
Microsoft.AspNetCore.All 2.1.13 [/home/user/dotnet/shared/Microsoft.AspNetCore.All]
Microsoft.AspNetCore.App 2.1.7 [/home/user/dotnet/shared/Microsoft.AspNetCore.App]
Microsoft.AspNetCore.App 2.1.13 [/home/user/dotnet/shared/Microsoft.AspNetCore.App]
Microsoft.AspNetCore.App 3.1.0 [/home/user/dotnet/shared/Microsoft.AspNetCore.App]
Microsoft.AspNetCore.App 5.0.0 [/home/user/dotnet/shared/Microsoft.AspNetCore.App]
Microsoft.AspNetCore.App 6.0.0 [/home/user/dotnet/shared/Microsoft.AspNetCore.App]
Microsoft.NETCore.App 2.1.7 [/home/user/dotnet/shared/Microsoft.NETCore.App]
Microsoft.NETCore.App 2.1.13 [/home/user/dotnet/shared/Microsoft.NETCore.App]
Microsoft.NETCore.App 3.1.0 [/home/user/dotnet/shared/Microsoft.NETCore.App]
Microsoft.NETCore.App 5.0.0 [/home/user/dotnet/shared/Microsoft.NETCore.App]
Microsoft.NETCore.App 6.0.0 [/home/user/dotnet/shared/Microsoft.NETCore.App]
```

```
Microsoft.AspNetCore.All 2.1.7 [/usr/local/share/dotnet/shared/Microsoft.AspNetCore.All]
Microsoft.AspNetCore.All 2.1.13 [/usr/local/share/dotnet/shared/Microsoft.AspNetCore.All]
Microsoft.AspNetCore.App 2.1.7 [/usr/local/share/dotnet/shared/Microsoft.AspNetCore.App]
Microsoft.AspNetCore.App 2.1.13 [/usr/local/share/dotnet/shared/Microsoft.AspNetCore.App]
Microsoft.AspNetCore.App 3.1.0 [/usr/local/share/dotnet/shared/Microsoft.AspNetCore.App]
Microsoft.AspNetCore.App 5.0.0 [/usr/local/share/dotnet/shared/Microsoft.AspNetCore.App]
Microsoft.AspNetCore.App 6.0.0 [/usr/local/share/dotnet/shared/Microsoft.AspNetCore.App]
Microsoft.NETCore.App 2.1.7 [/usr/local/share/dotnet/shared/Microsoft.NETCore.App]
Microsoft.NETCore.App 2.1.13 [/usr/local/share/dotnet/shared/Microsoft.NETCore.App]
Microsoft.NETCore.App 3.1.0 [/usr/local/share/dotnet/shared/Microsoft.NETCore.App]
Microsoft.NETCore.App 5.0.0 [/usr/local/share/dotnet/shared/Microsoft.NETCore.App]
Microsoft.NETCore.App 6.0.0 [/usr/local/share/dotnet/shared/Microsoft.NETCore.App]
```

Check for install folders

It's possible that .NET is installed but not added to the `PATH` variable for your operating system or user profile. In this case, the commands from the previous sections may not work. As an alternative, you can check that the .NET install folders exist.

When you install .NET from an installer or script, it's installed to a standard folder. Much of the time the installer or script you're using to install .NET gives you an option to install to a different folder. If you choose to install to a different folder, adjust the start of the folder path.

- **dotnet executable**

C:\program files\dotnet\dotnet.exe

- **.NET SDK**

C:\program files\dotnet\sdk\{version}

- **.NET Runtime**

C:\program files\dotnet\shared\{runtime-type}\{version}

- **dotnet executable**

/home/user/share/dotnet/dotnet

- **.NET SDK**

/home/user/share/dotnet/sdk/{version}

- **.NET Runtime**

/home/user/share/dotnet/shared/{runtime-type}/{version}

- **dotnet executable**

/usr/local/share/dotnet/dotnet

- **.NET SDK**

/usr/local/share/dotnet/sdk/{version}

- **.NET Runtime**

/usr/local/share/dotnet/shared/{runtime-type}/{version}

More information

You can see both the SDK versions and runtime versions with the command `dotnet --info`. You'll also get other environmental related information, such as the operating system version and runtime identifier (RID).

Next steps

- [Install the .NET Runtime and SDK for Windows.](#)
- [Install the .NET Runtime and SDK for macOS.](#)
- [Install the .NET Runtime and SDK for Linux.](#)

See also

- [Determine which .NET Framework versions are installed](#)

How to install localized IntelliSense files for .NET

9/20/2022 • 3 minutes to read • [Edit Online](#)

IntelliSense is a code-completion aid that's available in different integrated development environments (IDEs), such as Visual Studio. By default, when you're developing .NET projects, the SDK only includes the English version of the IntelliSense files. This article explains:

- How to install the localized version of those files.
- How to modify the Visual Studio installation to use a different language.

Prerequisites

- [.NET Core 3.1 SDK](#) or a later version, such as the [.NET 6 SDK](#).
- [Visual Studio 2019 version 16.3](#) or a later version.

Download and install the localized IntelliSense files

IMPORTANT

This procedure requires that you have administrator permission to copy the IntelliSense files to the .NET installation folder.

1. Go to the [Download IntelliSense files](#) page.
2. Download the IntelliSense file for the language and version you'd like to use.
3. Extract the contents of the zip file.
4. Navigate to the .NET Intellisense folder.
 - a. Navigate to the .NET installation folder. By default, it's under `%ProgramFiles%\dotnet\packs`.
 - b. Choose which SDK you want to install the IntelliSense for, and navigate to the associated path. You have the following options:

SDK TYPE	PATH
.NET 5+ and .NET Core	<code>Microsoft.NETCore.App.Ref</code>
Windows Desktop	<code>Microsoft.WindowsDesktop.App.Ref</code>
.NET Standard	<code>NETStandard.Library.Ref</code>

- c. Navigate to the version you want to install the localized IntelliSense for. For example, `6.0.0`.
- d. Open the `ref` folder.
- e. Open the moniker folder. For example, `net6.0`.

So, the full path that you'd navigate to would look similar to `C:\Program Files\dotnet\packs\Microsoft.NETCore.App.Ref\6.0.0\ref\net6.0`.

5. Create a subfolder inside the moniker folder you just opened. The name of the folder indicates which language you want to use. The following table specifies the different options:

LANGUAGE	FOLDER NAME
Brazilian Portuguese	<i>pt-br</i>
Chinese (simplified)	<i>zh-hans</i>
Chinese (traditional)	<i>zh-hant</i>
French	<i>fr</i>
German	<i>de</i>
Italian	<i>it</i>
Japanese	<i>ja</i>
Korean	<i>ko</i>
Russian	<i>ru</i>
Spanish	<i>es</i>

6. Copy the *.xm*/files you extracted in step 3 to this new folder. The *.xm*/files are broken down by SDK folders, so copy them to the matching SDK you chose in step 4.

Modify Visual Studio language

For Visual Studio to use a different language for IntelliSense, install the appropriate language pack. This can be done [during installation](#) or at a later time by modifying the Visual Studio installation. If you already have Visual Studio configured to the language of your choice, your IntelliSense installation is ready.

Install the language pack

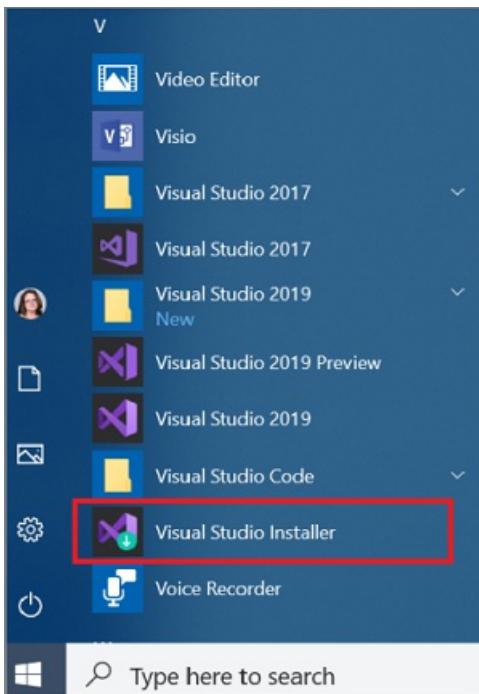
If you didn't install the desired language pack during setup, update Visual Studio as follows to install the language pack:

IMPORTANT

To install, update, or modify Visual Studio, you must log on with an account that has administrator permission. For more information, see [User permissions and Visual Studio](#).

1. Find the Visual Studio Installer on your computer.

For example, on a computer running Windows 10, select **Start**, and then scroll to the letter **V**, where it's listed as **Visual Studio Installer**.



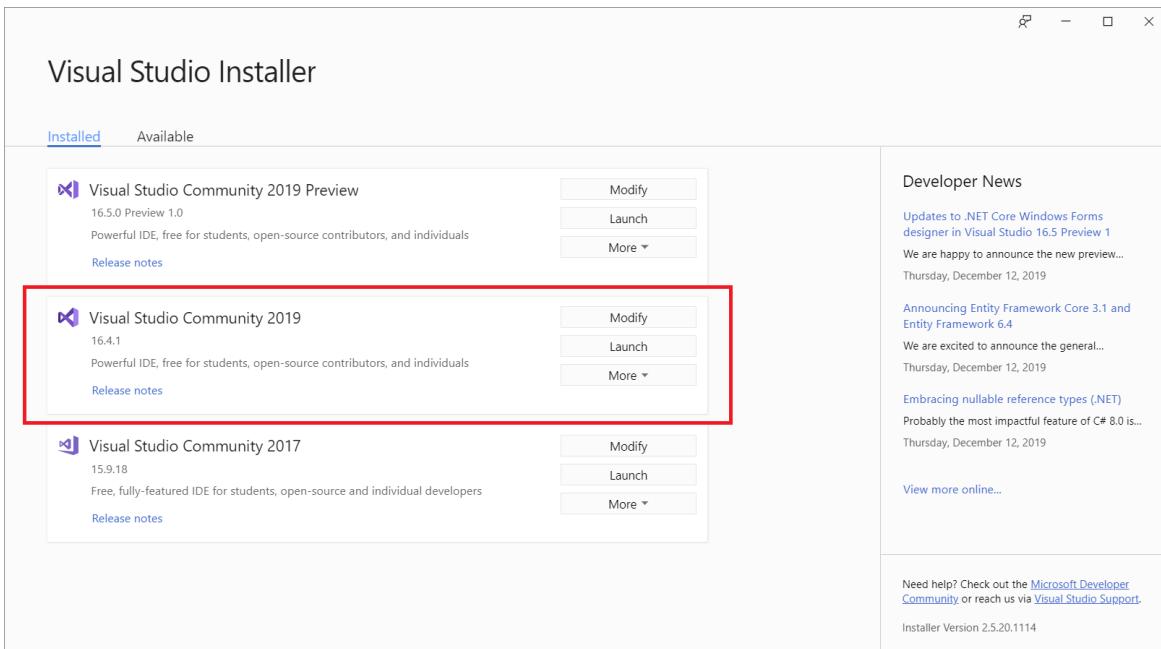
NOTE

You can also find the Visual Studio Installer in the following location:

```
C:\Program Files (x86)\Microsoft Visual Studio\Installer\vs_installer.exe
```

You might have to update the installer before continuing. If so, follow the prompts.

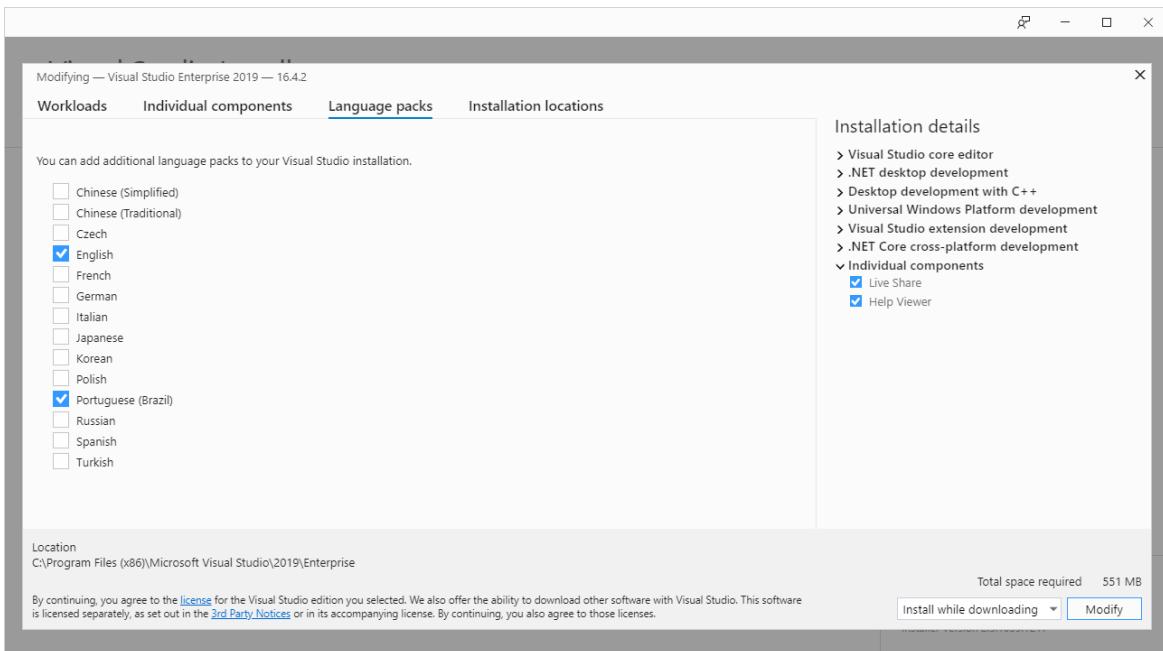
2. In the installer, look for the edition of Visual Studio that you want to add the language pack to, and then choose **Modify**.



IMPORTANT

If you don't see a **Modify** button but you see an **Update** one instead, you need to update your Visual Studio before you can modify your installation. After the update is finished, the **Modify** button should appear.

3. In the **Language packs** tab, select or deselect the languages you want to install or uninstall.



4. Choose **Modify**. The update starts.

Modify language settings in Visual Studio

Once you've installed the desired language packs, modify your Visual Studio settings to use a different language:

1. Open Visual Studio.
2. On the start window, choose **Continue without code**.
3. On the menu bar, select **Tools > Options**. The Options dialog opens.
4. Under the **Environment** node, choose **International Settings**.
5. On the **Language** drop-down, select the desired language. Choose **OK**.
6. A dialog informs you that you have to restart Visual Studio for the changes to take effect. Choose **OK**.
7. Restart Visual Studio.

After this, your IntelliSense should work as expected when you open a .NET project that targets the version of the IntelliSense files you just installed.

See also

- [IntelliSense in Visual Studio](#)

What is .NET? Introduction and overview

9/20/2022 • 8 minutes to read • [Edit Online](#)

.NET is a free, cross-platform, open source developer platform for building many kinds of applications. .NET is built on a high-performance runtime that is used in production by many high-scale apps.

Cloud apps

- [Cloud native apps](#)
- [Console apps](#)
- [Serverless functions in the cloud](#)
- [Web apps, web APIs, and microservices](#)

Cross-platform client apps

- [Desktop apps](#)
- [Games](#)
- [Mobile apps](#)

Windows apps

- [Windows Desktop apps](#)
 - [Windows Forms](#)
 - [Windows WPF](#)
 - [Universal Windows Platform \(UWP\)](#)
- [Windows services](#)

Other app types

- [Machine learning](#)
- [Internet of Things \(IoT\)](#)

Features

.NET features allow developers to productively write reliable and performant code.

- [Asynchronous code](#)
- [Attributes](#)
- [Reflection](#)
- [Code analyzers](#)
- [Delegates and lambdas](#)
- [Events](#)
- [Exceptions](#)
- [Garbage collection](#)
- [Generic types](#)
- [LINQ \(Language Integrated Query\).](#)
- [Parallel programming](#)
- Type inference - [C#](#), [F#](#), [Visual Basic](#).
- [Type system](#)
- [Unsafe code](#)

Using .NET

.NET apps and libraries are built from source code and a project file, using the [.NET CLI](#) or an Integrated Development Environment (IDE) like [Visual Studio](#).

The following example is a minimal .NET app:

Project file:

```
<Project Sdk="Microsoft.NET.Sdk">
<PropertyGroup>
<OutputType>Exe</OutputType>
<TargetFramework>net6.0</TargetFramework>
</PropertyGroup>
</Project>
```

Source code:

```
Console.WriteLine("Hello, World!");
```

The app can be built and run with the [.NET CLI](#):

```
% dotnet run
Hello, World!
```

It can also be built and run as two separate steps. The following example is for an app that is named *app*:

```
% dotnet build
% ./bin/Debug/net6.0/app
Hello, World!
```

Binary distributions

- [.NET SDK](#) -- Set of tools, libraries, and runtimes for development, building, and testing apps.
- [.NET Runtimes](#) -- Set of runtimes and libraries, for running apps.

You can download .NET from:

- [The Microsoft download site](#).
- [Containers](#).
- [Linux package managers](#).

Free and open source

.NET is free, open source, and is a [.NET Foundation project](#). .NET is maintained by Microsoft and the community on GitHub in [several repositories](#).

.NET source and binaries are licensed with the [MIT license](#). Additional [licenses apply on Windows](#) for binary distributions.

Support

Microsoft supports .NET on Android, Apple, Linux, and Windows operating systems. It can be used on Arm64,

x64, and x86 architectures. It's also supported in emulated environments, like [macOS Rosetta 2](#).

New versions of .NET are released annually in November. .NET releases in odd-numbered years are Long-term Support (LTS) releases and are supported for three years. Releases in even-numbered years are Short-term Support (STS) releases and are supported for 18 months. The quality level, breaking change policies, and all other aspects of the releases are the same. For more information, see [Releases and support](#).

The .NET Team at Microsoft works collaboratively with other organizations to distribute and support .NET in various ways.

[Red Hat supports .NET](#) on Red Hat Enterprise Linux (RHEL).

[Samsung supports .NET](#) on Tizen platforms.

Runtime

The [Common Language Runtime \(CLR\)](#) is the foundation all .NET apps are built on. The [fundamental features of the runtime](#) are:

- Garbage collection.
- Memory safety and type safety.
- High level support for programming languages.
- Cross-platform design.

.NET is sometimes called a "managed code" runtime. It's called *managed* primarily because it uses a garbage collector for memory management and because it enforces type and memory safety. The CLR virtualizes (or abstracts) various operating system and hardware concepts, such as memory, threads, and exceptions.

The CLR was designed to be a cross-platform runtime from its inception. It has been ported to multiple operating systems and architectures. Cross-platform .NET code typically does not need to be recompiled to run in new environments. Instead, you just need to install a different runtime to run your app.

The runtime exposes various [diagnostics](#) services and APIs for debuggers, [dumps](#) and [tracing](#) tools, and [observability](#). The observability implementation is primarily [built around OpenTelemetry](#), enabling [flexible application monitoring](#) and site reliability engineering (SRE).

The runtime offers low-level C-style interop functionality, via a combination of [P/Invoke](#), value types, and the ability to [blit](#) values across the native/managed-code boundary.

Languages

The runtime is designed to support multiple programming languages. C#, F#, and Visual Basic languages are supported by Microsoft and are designed in collaboration with the community.

- [C#](#) is a modern, object-oriented, and type-safe programming language. It has its roots in the C family of languages and will be immediately familiar to C, C++, Java, and JavaScript programmers.
- [F#](#) is an interoperable programming language for writing succinct, robust, and performant code. F# programming is data-oriented, where code involves transforming data with functions.
- [Visual Basic](#) uses a more verbose syntax that is closer to ordinary human language. It can be an easier language to learn for people new to programming.

Compilation

.NET apps (as written in a high-level language like C#) are compiled to an [Intermediate Language \(IL\)](#). IL is a compact code format that can be supported on any operating system or architecture. Most .NET apps use APIs

that are supported in multiple environments, requiring only the .NET runtime to run.

IL needs to be compiled to native code in order to execute on a CPU, for example Arm64 or x64. .NET supports both Ahead-Of-Time (AOT) and Just-In-Time (JIT) compilation models.

- On Android, Linux, macOS, and Linux, JIT compilation is the default, and AOT is optional (for example, with [ReadyToRun](#)).
- On [iOS](#), AOT is mandatory (except when running in the simulator).
- In WebAssembly (Wasm) environments, AOT is mandatory.

The advantage of the JIT is that it can compile an app (unmodified) to the CPU instructions and calling conventions in a given environment, per the underlying operating system and hardware. It can also compile code at higher or lower levels of quality to enable better startup and steady-state throughput performance.

The advantage of AOT is that it provides the best app startup and can (in some cases) result in smaller deployments. The primary downside is that binaries must be built for each separate deployment target (the same as any other native code). AOT code is not compatible with some reflection patterns.

Runtime libraries

.NET has a comprehensive standard set of class libraries. These libraries provide implementations for many general-purpose and workload-specific types and utility functionality.

Here are some examples of types defined in the .NET runtime libraries:

- Every .NET type derives from the [System.Object](#) type.
- Primitive value types, such as [System.Boolean](#) and [System.Int32](#).
- Collections, such as [System.Collections.Generic.List<T>](#) and [System.Collections.Generic.Dictionary< TKey, TValue >](#).
- Data types, such as [System.Data.DataSet](#) and [System.Data.DataTable](#).
- Network utility types, such as [System.Net.Http.HttpClient](#).
- [File and stream I/O](#) utility types, such as [System.IO.FileStream](#) and [System.IO.TextWriter](#).
- [Serialization](#) utility types, such as [System.Text.Json.JsonSerializer](#) and [System.Xml.Serialization.XmlSerializer](#).
- High-performance types, such as [System.Span<T>](#), [System.Numerics.Vector](#), and [Pipelines](#).

For more information, see the [Runtime libraries overview](#).

NuGet Package Manager

[NuGet](#) is the package manager for .NET. It enables developers to share compiled binaries with each other.

[NuGet.org](#) offers many [popular packages](#) from the community.

Tools

The [.NET SDK](#) is a set of libraries and [tools](#) for developing and running .NET applications. It includes the [MSBuild](#) build engine, the [Roslyn](#) (C# and Visual Basic) compiler, and the [F#](#) compiler. Most commands are run by using the [dotnet](#) command. The CLI tools can be used for local development and continuous integration.

The [Visual Studio](#) family of IDEs offer excellent support for .NET and the C#, F#, and Visual Basic languages.

[GitHub Codespaces](#) and [GitHub security features](#) support .NET.

Notebooks

.NET Interactive is a group of CLI tools and APIs that enable users to create interactive experiences across the

web, markdown, and notebooks.

For more information, see the following resources:

- [.NET In-Browser Tutorial](#)
- [Using .NET notebooks with Jupyter on your machine](#)
- [.NET Interactive documentation](#)

CI/CD

MSBuild and the .NET CLI can be used with various continuous integration tools and environments, such as:

- [GitHub Actions](#)
- [Azure DevOps](#)
- [CAKE for C#](#)
- [FAKE for F#](#)

For more information, see [Use the .NET SDK in Continuous Integration \(CI\) environments](#).

Deployment models

.NET apps can be [published in two different modes](#):

- *Self-contained* apps include the .NET runtime and dependent libraries. They can be [single-file](#) or multi-file. Users of the application can run it on a machine that doesn't have the .NET runtime installed. Self-contained apps always target a single operating system and architecture configuration.
- *Framework-dependent* apps require a compatible version of the .NET runtime, typically installed globally. Framework-dependent apps can be published for a single operating system and architecture configuration or as "portable," targeting all supported configurations.

.NET apps are launched with a native executable, by default. The executable is both operating system and architecture-specific. Apps can also be launched with the `dotnet` command.

Apps can be [deployed in containers](#). Microsoft provides [container images](#) for various target environments.

.NET history

In 2002, Microsoft released [.NET Framework](#), a development platform for creating Windows apps. Today .NET Framework is at version 4.8 and remains [fully supported by Microsoft](#).

In 2014, Microsoft introduced .NET Core as a cross-platform, open-source successor to .NET Framework. This new [implementation of .NET](#) kept the name .NET Core through version 3.1. The next version after .NET Core 3.1 was named .NET 5.

New .NET versions continue to be released annually, each a major version number higher. They include significant new features and often enable new scenarios.

.NET ecosystem

There are multiple variants of .NET, each supporting a different type of app. The reason for multiple variants is part historical, part technical.

.NET implementations (historical order):

- **.NET Framework** -- It provides access to the broad capabilities of Windows and Windows Server. Also extensively used for Windows-based cloud computing. The original .NET.
- **Mono** -- A cross-platform implementation of .NET Framework. The original community and open source

.NET. Used for Android, iOS, and Wasm apps.

- **.NET (Core)** -- A cross-platform and open source implementation of .NET, rethought for the cloud age while remaining significantly compatible with .NET Framework. Used for Linux, macOS, and Windows apps.

Next steps

- [Choose a .NET tutorial](#)
- [Try .NET in your browser](#)
- [Take a tour of C#](#)
- [Take a tour of F#](#)

.NET implementations

9/20/2022 • 2 minutes to read • [Edit Online](#)

A .NET app is developed for one or more *implementations of .NET*. Implementations of .NET include .NET Framework, .NET 5+ (and .NET Core), and Mono.

Each implementation of .NET includes the following components:

- One or more runtimes—for example, .NET Framework CLR and .NET 5 CLR.
- A class library—for example, .NET Framework Base Class Library and .NET 5 Base Class Library.
- Optionally, one or more application frameworks—for example, [ASP.NET](#), [Windows Forms](#), and [Windows Presentation Foundation \(WPF\)](#) are included in .NET Framework and .NET 5+.
- Optionally, development tools. Some development tools are shared among multiple implementations.

There are four .NET implementations that Microsoft supports:

- .NET 5 (and .NET Core) and later versions
- .NET Framework
- Mono
- UWP

.NET 6 is currently the primary implementation, and the one that's the focus of ongoing development. .NET 6 is built on a single code base that supports multiple platforms and many workloads, such as Windows desktop apps and cross-platform console apps, cloud services, and websites. [Some workloads](#), such as .NET WebAssembly build tools, are available as optional installations.

.NET 5 and later versions

.NET 5+, previously referred to as .NET Core, is a cross-platform implementation of .NET that's designed to handle server and cloud workloads at scale. It also supports other workloads, including desktop apps. It runs on Windows, macOS, and Linux. It implements .NET Standard, so code that targets .NET Standard can run on .NET 5+. [ASP.NET Core](#), [Windows Forms](#), and [Windows Presentation Foundation \(WPF\)](#) all run on .NET 5+.

.NET 6 is the latest version of this .NET implementation.

For more information, see the following resources:

- [.NET introduction](#)
- [.NET vs. .NET Framework for server apps](#)
- [.NET 5+ and .NET Standard](#)

.NET Framework

.NET Framework is the original .NET implementation that has existed since 2002. Versions 4.5 and later implement .NET Standard, so code that targets .NET Standard can run on those versions of .NET Framework. It contains additional Windows-specific APIs, such as APIs for Windows desktop development with Windows Forms and WPF. .NET Framework is optimized for building Windows desktop applications.

For more information, see the [.NET Framework guide](#).

Mono

Mono is a .NET implementation that is mainly used when a small runtime is required. It is the runtime that powers Xamarin applications on Android, macOS, iOS, tvOS, and watchOS and is focused primarily on a small footprint. Mono also powers games built using the Unity engine.

It supports all of the currently published .NET Standard versions.

Historically, Mono implemented the larger API of .NET Framework and emulated some of the most popular capabilities on Unix. It is sometimes used to run .NET applications that rely on those capabilities on Unix.

Mono is typically used with a just-in-time compiler, but it also features a full static compiler (ahead-of-time compilation) that is used on platforms like iOS.

For more information, see the [Mono documentation](#).

Universal Windows Platform (UWP)

UWP is an implementation of .NET that is used for building modern, touch-enabled Windows applications and software for the Internet of Things (IoT). It's designed to unify the different types of devices that you may want to target, including PCs, tablets, phones, and even the Xbox. UWP provides many services, such as a centralized app store, an execution environment (AppContainer), and a set of Windows APIs to use instead of Win32 (WinRT).

Apps can be written in C++, C#, Visual Basic, and JavaScript.

For more information, see [Introduction to the Universal Windows Platform](#).

.NET class libraries

9/20/2022 • 3 minutes to read • [Edit Online](#)

Class libraries are the [shared library](#) concept for .NET. They enable you to componentize useful functionality into modules that can be used by multiple applications. They can also be used as a means of loading functionality that is not needed or not known at application startup. Class libraries are described using the [.NET Assembly file format](#).

There are three types of class libraries that you can use:

- **Platform-specific** class libraries have access to all the APIs in a given platform (for example, .NET Framework on Windows, Xamarin iOS), but can only be used by apps and libraries that target that platform.
- **Portable** class libraries have access to a subset of APIs, and can be used by apps and libraries that target multiple platforms.
- **.NET Standard** class libraries are a merger of the platform-specific and portable library concept into a single model that provides the best of both.

Platform-specific class libraries

Platform-specific libraries are bound to a single .NET platform (for example, .NET Framework on Windows) and can therefore take significant dependencies on a known execution environment. Such an environment exposes a known set of APIs (.NET and OS APIs) and maintains and exposes expected state (for example, Windows registry).

Developers who create platform-specific libraries can fully exploit the underlying platform. The libraries will only ever run on that given platform, making platform checks or other forms of conditional code unnecessary (modulo single sourcing code for multiple platforms).

Platform-specific libraries have been the primary class library type for the .NET Framework. Even as other .NET implementations emerged, platform-specific libraries remained the dominant library type.

Portable class libraries

Portable libraries are supported on multiple .NET implementations. They can still take dependencies on a known execution environment, however, the environment is a synthetic one that's generated by the intersection of a set of concrete .NET implementations. Exposed APIs and platform assumptions are a subset of what would be available to a platform-specific library.

You choose a platform configuration when you create a portable library. The platform configuration is the set of platforms that you need to support (for example, .NET Framework 4.5+, Windows Phone 8.0+). The more platforms you opt to support, the fewer APIs and fewer platform assumptions you can make, the lowest common denominator. This characteristic can be confusing at first, since people often think "more is better" but find that more supported platforms results in fewer available APIs.

Many library developers have switched from producing multiple platform-specific libraries from one source (using conditional compilation directives) to portable libraries. There are [several approaches](#) for accessing platform-specific functionality within portable libraries, with bait-and-switch being the most widely accepted technique at this point.

.NET Standard class libraries

.NET Standard libraries are a replacement of the platform-specific and portable libraries concepts. They are platform-specific in the sense that they expose all functionality from the underlying platform (no synthetic platforms or platform intersections). They are portable in the sense that they work on all supporting platforms.

.NET Standard exposes a set of library *contracts*. .NET implementations must support each contract fully or not at all. Each implementation, therefore, supports a set of .NET Standard contracts. The corollary is that each .NET Standard class library is supported on the platforms that support its contract dependencies.

.NET Standard does not expose the entire functionality of .NET Framework (nor is that a goal), however, the libraries do expose many more APIs than Portable Class Libraries.

The following implementations support .NET Standard libraries:

- .NET Core
- .NET Framework
- Mono
- Universal Windows Platform (UWP)

For more information, see [.NET Standard](#).

Mono class libraries

Class libraries are supported on Mono, including the three types of libraries described previously. Mono is often viewed as a cross-platform implementation of .NET Framework. In part, this is because platform-specific .NET Framework libraries can run on the Mono runtime without modification or recompilation. This characteristic was in place before the creation of portable class libraries, so was an obvious choice to enable binary portability between .NET Framework and Mono (although it only worked in one direction).

.NET Standard

9/20/2022 • 14 minutes to read • [Edit Online](#)

.NET Standard is a formal specification of .NET APIs that are available on multiple .NET implementations. The motivation behind .NET Standard was to establish greater uniformity in the .NET ecosystem. .NET 5 and later versions adopt a different approach to establishing uniformity that eliminates the need for .NET Standard in most scenarios. However, if you want to share code between .NET Framework and any other .NET implementation, such as .NET Core, your library should target .NET Standard 2.0. [No new versions of .NET Standard will be released](#), but .NET 5, .NET 6, and all future versions will continue to support .NET Standard 2.1 and earlier.

For information about choosing between .NET 5+ and .NET Standard, see [.NET 5+ and .NET Standard](#) later in this article.

.NET Standard versions

.NET Standard is versioned. Each new version adds more APIs. When a library is built against a certain version of .NET Standard, it can run on any .NET implementation that implements that version of .NET Standard (or higher).

Targeting a higher version of .NET Standard allows a library to use more APIs but means it can only be used on more recent versions of .NET. Targeting a lower version reduces the available APIs but means the library can run in more places.

Select .NET Standard version

- [1.0](#)
- [1.1](#)
- [1.2](#)
- [1.3](#)
- [1.4](#)
- [1.5](#)
- [1.6](#)
- [2.0](#)
- [2.1](#)

.NET Standard 1.0 has 7,949 of the 37,118 available APIs.

.NET IMPLEMENTATION	VERSION SUPPORT
.NET and .NET Core	1.0, 1.1, 2.0, 2.1, 2.2, 3.0, 3.1, 5.0, 6.0
.NET Framework	4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8
Mono	4.6, 5.4, 6.4
Xamarin.iOS	10.0, 10.14, 12.16
Xamarin.Mac	3.0, 3.8, 5.16

.NET IMPLEMENTATION	VERSION SUPPORT
Xamarin.Android	7.0, 8.0, 10.0
Universal Windows Platform	8.0, 8.1, 10.0, 10.0.16299, TBD
Unity	2018.1

For more information, see [.NET Standard 1.0](#). For an interactive table, see [.NET Standard versions](#).

Which .NET Standard version to target

We recommend you target .NET Standard 2.0, unless you need to support an earlier version. Most general-purpose libraries should not need APIs outside of .NET Standard 2.0. .NET Standard 2.0 is supported by all modern platforms and is the recommended way to support multiple platforms with one target.

If you need to support .NET Standard 1.x, we recommend that you *also* target .NET Standard 2.0. .NET Standard 1.x is distributed as a granular set of NuGet packages, which creates a large package dependency graph and results in developers downloading a lot of packages when building. For more information, see [Cross-platform targeting](#) and [.NET 5+ and .NET Standard](#) later in this article.

.NET Standard versioning rules

There are two primary versioning rules:

- Additive: .NET Standard versions are logically concentric circles: higher versions incorporate all APIs from previous versions. There are no breaking changes between versions.
- Immutable: Once shipped, .NET Standard versions are frozen.

There will be no new .NET Standard versions after 2.1. For more information, see [.NET 5+ and .NET Standard](#) later in this article.

Specification

The .NET Standard specification is a standardized set of APIs. The specification is maintained by .NET implementers, specifically Microsoft (includes .NET Framework, .NET Core, and Mono) and Unity.

Official artifacts

The official specification is a set of .cs files that define the APIs that are part of the standard. The [ref directory](#) in the (now archived) [dotnet/standard repository](#) defines the .NET Standard APIs.

The [NETStandard.Library](#) metapackage ([source](#)) describes the set of libraries that define (in part) one or more .NET Standard versions.

A given component, like `System.Runtime`, describes:

- Part of .NET Standard (just its scope).
- Multiple versions of .NET Standard, for that scope.

Derivative artifacts are provided to enable more convenient reading and to enable certain developer scenarios (for example, using a compiler).

- [API list in markdown](#).
- Reference assemblies, distributed as NuGet packages and referenced by the [NETStandard.Library](#) metapackage.

Package representation

The primary distribution vehicle for the .NET Standard reference assemblies is NuGet packages.

Implementations are delivered in a variety of ways, appropriate for each .NET implementation.

NuGet packages target one or more [frameworks](#). .NET Standard packages target the ".NET Standard" framework. You can target the .NET Standard framework using the `netstandard` [compact TFM](#) (for example, `netstandard1.4`). Libraries that are intended to run on multiple implementations of .NET should target this framework. For the broadest set of APIs, target `netstandard2.0` since the number of available APIs more than doubled between .NET Standard 1.6 and 2.0.

The `NETStandard.Library` metapackage references the complete set of NuGet packages that define .NET Standard. The most common way to target `netstandard` is by referencing this metapackage. It describes and provides access to the ~40 .NET libraries and associated APIs that define .NET Standard. You can reference additional packages that target `netstandard` to get access to additional APIs.

Versioning

The specification is not singular, but a linearly versioned set of APIs. The first version of the standard establishes a baseline set of APIs. Subsequent versions add APIs and inherit APIs defined by previous versions. There is no established provision for removing APIs from the Standard.

.NET Standard is not specific to any one .NET implementation, nor does it match the versioning scheme of any of those implementations.

As noted earlier, there will be no new .NET Standard versions after 2.1.

Target .NET Standard

You can [build .NET Standard Libraries](#) using a combination of the `netstandard` framework and the `NETStandard.Library` metapackage.

.NET Framework compatibility mode

Starting with .NET Standard 2.0, the .NET Framework compatibility mode was introduced. This compatibility mode allows .NET Standard projects to reference .NET Framework libraries as if they were compiled for .NET Standard. Referencing .NET Framework libraries doesn't work for all projects, such as libraries that use Windows Presentation Foundation (WPF) APIs.

For more information, see [.NET Framework compatibility mode](#).

.NET Standard libraries and Visual Studio

In order to build .NET Standard libraries in Visual Studio, make sure you have [Visual Studio 2022](#), [Visual Studio 2019](#), or [Visual Studio 2017](#) version 15.3 or later installed on Windows, or [Visual Studio for Mac](#) version 7.1 or later installed on macOS.

If you only need to consume .NET Standard 2.0 libraries in your projects, you can also do that in Visual Studio 2015. However, you need NuGet client 3.6 or higher installed. You can download the NuGet client for Visual Studio 2015 from the [NuGet downloads](#) page.

.NET 5+ and .NET Standard

.NET 5 and .NET 6 are single products with a uniform set of capabilities and APIs that can be used for Windows desktop apps and cross-platform console apps, cloud services, and websites. The .NET 5 [TFMs](#), for example, reflect this broad range of scenarios:

- `net5.0`

This TFM is for code that runs everywhere. With a few exceptions, it includes only technologies that work

cross-platform. For .NET 5 code, `net5.0` replaces both `netcoreapp` and `netstandard` TFMs.

- `net5.0-windows`

This is an example of an [OS-specific TFM](#) that add OS-specific functionality to everything that `net5.0` refers to.

When to target `net5.0` or `net6.0` vs. `netstandard`

For existing code that targets `netstandard`, there's no need to change the TFM to `net5.0` or `net6.0`. .NET 5 and .NET 6 implement .NET Standard 2.1 and earlier. The only reason to retarget from .NET Standard to .NET 5+ would be to gain access to more runtime features, language features, or APIs. For example, in order to use C# 9, you need to target .NET 5 or a later version. You can multitarget .NET 5 or .NET 6 and .NET Standard to get access to newer features and still have your library available to other .NET implementations.

Here are some guidelines for new code for .NET 5+:

- App components

If you're using libraries to break down an application into several components, we recommend you target `net5.0` or `net6.0`. For simplicity, it's best to keep all projects that make up your application on the same version of .NET. Then you can assume the same BCL features everywhere.

- Reusable libraries

If you're building reusable libraries that you plan to ship on NuGet, consider the trade-off between reach and available feature set. .NET Standard 2.0 is the latest version that's supported by .NET Framework, so it gives good reach with a fairly large feature set. We don't recommend targeting .NET Standard 1.x, as you'd limit the available feature set for a minimal increase in reach.

If you don't need to support .NET Framework, you could go with .NET Standard 2.1 or .NET 5/6. We recommend you skip .NET Standard 2.1 and go straight to .NET 6. Most widely used libraries will multi-target for both .NET Standard 2.0 and .NET 5+. Supporting .NET Standard 2.0 gives you the most reach, while supporting .NET 5+ ensures you can leverage the latest platform features for customers that are already on .NET 5+.

.NET Standard problems

Here are some problems with .NET Standard that help explain why .NET 5 and later versions are the better way to share code across platforms and workloads:

- Slowness to add new APIs

.NET Standard was created as an API set that all .NET implementations would have to support, so there was a review process for proposals to add new APIs. The goal was to standardize only APIs that could be implemented in all current and future .NET platforms. The result was that if a feature missed a particular release, you might have to wait for a couple of years before it got added to a version of the Standard. Then you'd wait even longer for the new version of .NET Standard to be widely supported.

Solution in .NET 5+: When a feature is implemented, it's already available for every .NET 5+ app and library because the code base is shared. And since there's no difference between the API specification and its implementation, you're able to take advantage of new features much quicker than with .NET Standard.

- Complex versioning

The separation of the API specification from its implementations results in complex mapping between API specification versions and implementation versions. This complexity is evident in the table shown earlier in this article and the instructions for how to interpret it.

Solution in .NET 5+: There's no separation between a .NET 5+ API specification and its implementation.

The result is a simplified TFM scheme. There's one TFM prefix for all workloads: `net5.0` or `net6.0` is used for libraries, console apps, and web apps. The only variation is a [suffix that specifies platform-specific APIs](#) for a particular platform, such as `net5.0-windows` or `net6.0-windows`. Thanks to this TFM naming convention, you can easily tell whether a given app can use a given library. No version number equivalents table, like the one for .NET Standard, is needed.

- Platform-unsupported exceptions at run time

.NET Standard exposes platform-specific APIs. Your code might compile without errors and appear to be portable to any platform even if it isn't portable. When it runs on a platform that doesn't have an implementation for a given API, you get run-time errors.

Solution in .NET 5+: The .NET 5+ SDKs include code analyzers that are enabled by default. The platform compatibility analyzer detects unintentional use of APIs that aren't supported on the platforms you intend to run on. For more information, see [Platform compatibility analyzer](#).

.NET Standard not deprecated

.NET Standard is still needed for libraries that can be used by multiple .NET implementations. We recommend you target .NET Standard in the following scenarios:

- Use `netstandard2.0` to share code between .NET Framework and all other implementations of .NET.
- Use `netstandard2.1` to share code between Mono, Xamarin, and .NET Core 3.x.

See also

- [.NET Standard versions \(source\)](#)
- [.NET Standard versions \(interactive UI\)](#)
- [Build a .NET Standard library](#)
- [Cross-platform targeting](#)

Releases and support for .NET (.NET 5+ and .NET Core)

9/20/2022 • 7 minutes to read • [Edit Online](#)

Microsoft ships major releases, minor releases, and servicing updates (patches) for .NET 5 (and .NET Core) and later versions. This article explains release types, servicing updates, SDK feature bands, support periods, and support options.

Release types

Information about the type of each release is encoded in the version number in the form *major.minor.patch*.

For example:

- .NET 5 and NET 6 are major releases.
- .NET Core 3.1 is the first minor release after the .NET Core 3.0 major release.
- .NET Core 5.0.15 is the fifteenth patch for .NET 5.

Major releases

Major releases include new features, new public API surface area, and bug fixes. Examples include .NET 5 and .NET 6. Due to the nature of the changes, these releases are expected to have breaking changes. Major releases install side by side with previous major releases.

Minor releases

Minor releases also include new features, public API surface area, and bug fixes, and may also have breaking changes. An example is .NET Core 3.1. The difference between these and major releases is that the magnitude of the changes is smaller. An application upgrading from .NET Core 3.0 to 3.1 has a smaller jump to move forward. Minor releases install side by side with previous minor releases.

Servicing updates

Servicing updates (patches) ship almost every month, and these updates carry both security and non-security bug fixes. For example, .NET 5.0.8 is the eighth update for .NET 5. When these updates include security fixes, they're released on "patch Tuesday", which is always the second Tuesday of the month. Servicing updates are expected to maintain compatibility. Starting with .NET Core 3.1, servicing updates are upgrades that remove the preceding update. For example, the latest servicing update for 3.1 removes the previous 3.1 update upon successful installation.

Feature bands (SDK only)

Versioning for the .NET SDK works slightly differently from the .NET runtime. To align with new Visual Studio releases, .NET SDK updates sometimes include new features or new versions of components like MSBuild and NuGet. These new features or components may be incompatible with the versions that shipped in previous SDK updates for the same major or minor version.

To differentiate such updates, the .NET SDK uses the concept of feature bands. For example, the first .NET 5 SDK was 5.0.100. This release corresponds to the 3.1.1xx *feature band*. Feature bands are defined in the hundreds groups in the third section of the version number. For example, 5.0.101 and 5.0.201 are versions in two different feature bands while 5.0.101 and 5.0.199 are in the same feature band. When .NET SDK 5.0.101 is installed, .NET SDK 5.1.100 is removed from the machine if it exists. When .NET SDK 5.0.200 is installed on the same machine, .NET SDK 5.0.101 isn't removed.

Runtime roll-forward and compatibility

Major and minor updates install side by side with previous versions. An application built to target a specific *major:minor* version continues to use that targeted runtime even if a newer version is installed. The app doesn't automatically roll forward to use a newer *major:minor* version of the runtime unless you opt in for this behavior. An application that was built to target .NET Core 3.0 doesn't automatically start running on .NET Core 3.1. We recommend rebuilding the app and testing against a newer major or minor runtime version before deploying to production. For more information, see [Framework-dependent apps roll forward](#) and [Self-contained deployment runtime roll forward](#).

Servicing updates are treated differently from major and minor releases. An application built to target .NET 5.0.0 runs on the 5.0.0 runtime by default. It automatically rolls forward to use a newer 5.0.1 runtime when that servicing update is installed. This behavior is the default because we want security fixes to be used as soon as they're installed without any other action needed. You can opt out from this default roll forward behavior.

.NET version lifecycles

.NET Core, .NET 5, and later versions adopt the [modern lifecycle](#) rather than the [fixed lifecycle](#) that has been used for .NET Framework releases. Products with fixed lifecycles provide a long fixed period of support, for example, 5 years of mainstream support and another 5 years of extended support. Mainstream support includes security and non-security fixes, while extended support provides security fixes only. Products that adopt a modern lifecycle have a more service-like support model, with shorter support periods and more frequent releases.

Release tracks

There are two support tracks for releases:

- *Current* releases

These versions are supported until six months after the next major or minor release ships. Previously (.NET Core 3.0 and earlier), these releases were supported for only three months after the next major or minor release shipped.

Example:

- .NET Core 3.0 shipped in September 2019 and was followed by .NET Core 3.1 in December 2019.
- .NET Core 3.0 support ended in March 2020, 3 months after 3.1 shipped.

- *Long Term Support* (LTS) releases

These versions are supported for a minimum of 3 years, or 1 year after the next LTS release ships if that date is later.

Example:

- .NET Core 3.1 is an LTS release and was released in December 2019. It's supported for 3 years, until December, 2022.
- .NET 5 is a Current release and was released in November 2020. It's supported for 18 months, until May, 2022.
- .NET 6 is an LTS release and was released in November, 2021. It's supported for 3 years, until November, 2024.

Releases alternate between LTS and Current, so it's possible for an earlier release to be supported longer than a later release. For example, .NET Core 3.1 is an LTS release with support through December 2022. The .NET 5 release shipped almost a year later but goes out of support earlier, in May 2022.

Servicing updates ship monthly and include both security and non-security (reliability, compatibility, and stability) fixes. Servicing updates are supported until the next servicing update is released. Servicing updates have runtime roll forward behavior. That means that applications default to running on the latest installed

runtime servicing update.

How to choose a release

If you're building a service and expect to continue updating it on a regular basis, then a Current release like .NET 5 may be your best option to stay up to date with the latest features .NET has to offer.

If you're building a client application that will be distributed to consumers, stability may be more important than access to the latest features. Your application might need to be supported for a certain period before the consumer can upgrade to the next version of the application. In that case, an LTS release like .NET 6 might be the right option.

Servicing updates

.NET servicing updates are supported until the next servicing update is released. The release cadence is monthly.

You need to regularly install servicing updates to ensure that your apps are in a secure and supported state. For example, if the latest servicing update for .NET 5 is 5.0.8 and we ship 5.0.9, then 5.0.8 is no longer the latest. The supported servicing level for .NET 5 is then 5.0.9.

For information about the latest servicing updates for each major and minor version, see the [.NET downloads page](#).

End of support

End of support refers to the date after which Microsoft no longer provides fixes, updates, or technical assistance for a product version. Before this date, make sure you have moved to using a supported version. Versions that are out of support no longer receive security updates that protect your applications and data.

Supported operating systems

.NET 5 (and .NET Core) and later versions can be run on a range of operating systems. Each of these operating systems has a lifecycle defined by its sponsor organization (for example, Microsoft, Red Hat, or Apple). We take these lifecycle schedules into account when adding and removing support for operating system versions.

When an operating system version goes out of support, we stop testing that version and providing support for that version. Users need to move forward to a supported operating system version to get support.

For more information, see the [.NET OS Lifecycle Policy](#).

Get support

You have a choice between Microsoft assisted support and Community support.

Microsoft support

For assisted support, [contact a Microsoft Support Professional](#).

You need to be on a supported servicing level (the latest available servicing update) to be eligible for support. If a system is running .NET 5 and the 5.0.8 servicing update has been released, then 5.0.8 needs to be installed as a first step.

Community support

For community support, see the [Community page](#).

See also

For more information, including supported date ranges for each version of .NET, see the [Support Policy](#).

Ecma standards

9/20/2022 • 2 minutes to read • [Edit Online](#)

The C# Language and the Common Language Infrastructure (CLI) specifications are standardized through [Ecma International](#). The first editions of these standards were published by Ecma in December 2001.

Subsequent revisions to the standards have been developed by the TC49-TG2 (C#) and TC49-TG3 (CLI) task groups within the Programming Languages Technical Committee ([TC49](#)), and adopted by the Ecma General Assembly and subsequently by ISO/IEC JTC 1 via the ISO Fast-Track process.

Latest standards

The following official Ecma documents are available for [C#](#) and the [CLI \(TR-84\)](#):

- **The C# Language Standard (version 6.0):** [ECMA-334.pdf](#)
- **The Common Language Infrastructure:** [ECMA-335.pdf](#).
- **Information Derived from the Partition IV XML File:** [ECMA-084.pdf](#) format.

The official ISO/IEC documents are available from the ISO/IEC [Publicly Available Standards](#) page. These links are direct from that page:

- **Information technology - Programming languages - C#:** [ISO/IEC 23270:2018](#)
- **Information technology — Common Language Infrastructure (CLI) Partitions I to VI:** [ISO/IEC 23271:2012](#)
- **Information technology — Common Language Infrastructure (CLI) — Technical Report on Information Derived from Partition IV XML File:** [ISO/IEC TR 23272:2011](#)

.NET Glossary

9/20/2022 • 14 minutes to read • [Edit Online](#)

The primary goal of this glossary is to clarify meanings of selected terms and acronyms that appear frequently in the .NET documentation.

AOT

Ahead-of-time compiler.

Similar to [JIT](#), this compiler also translates [IL](#) to machine code. In contrast to JIT compilation, AOT compilation happens before the application is executed and is usually performed on a different machine. Because AOT tool chains don't compile at run time, they don't have to minimize time spent compiling. That means they can spend more time optimizing. Since the context of AOT is the entire application, the AOT compiler also performs cross-module linking and whole-program analysis, which means that all references are followed and a single executable is produced.

See [CoreRT](#) and [.NET Native](#).

app model

A [workload](#)-specific API. Here are some examples:

- ASP.NET
- ASP.NET Web API
- Entity Framework (EF)
- Windows Presentation Foundation (WPF)
- Windows Communication Foundation (WCF)
- Windows Workflow Foundation (WF)
- Windows Forms (WinForms)

ASP.NET

The original ASP.NET implementation that ships with the .NET Framework, also known as ASP.NET 4.x.

Sometimes ASP.NET is an umbrella term that refers to both the original ASP.NET and ASP.NET Core. The meaning that the term carries in any given instance is determined by context. Refer to ASP.NET 4.x when you want to make it clear that you're not using ASP.NET to mean both implementations.

See [ASP.NET documentation](#).

ASP.NET Core

A cross-platform, high-performance, open-source implementation of ASP.NET.

See [ASP.NET Core documentation](#).

assembly

A [.dll](#) or [.exe](#) file that can contain a collection of APIs that can be called by applications or other assemblies.

An assembly may include types such as interfaces, classes, structures, enumerations, and delegates. Assemblies

in a project's *bin* folder are sometimes referred to as *binaries*. See also [library](#).

BCL

Base Class Library.

A set of libraries that comprise the System.* (and to a limited extent Microsoft.*) namespaces. The BCL is a general purpose, lower-level framework that higher-level application frameworks, such as ASP.NET Core, build on.

The source code of the BCL for [.NET 5 \(and .NET Core\) and later versions](#) is contained in the [.NET runtime repository](#). Most of these BCL APIs are also available in .NET Framework, so you can think of this source code as a fork of the .NET Framework BCL source code.

The following terms often refer to the same collection of APIs that BCL refers to:

- [core .NET libraries](#)
- [framework libraries](#)
- [runtime libraries](#)
- [shared framework](#)

CLR

Common Language Runtime.

The exact meaning depends on the context. Common Language Runtime usually refers to the runtime of [.NET Framework](#) or the runtime of [.NET 5 \(and .NET Core\) and later versions](#).

A CLR handles memory allocation and management. A CLR is also a virtual machine that not only executes apps but also generates and compiles code on-the-fly using a [JIT](#) compiler.

The CLR implementation for .NET Framework is Windows only.

The CLR implementation for .NET 5 and later versions (also known as the Core CLR) is built from the same code base as the .NET Framework CLR. Originally, the Core CLR was the runtime of Silverlight and was designed to run on multiple platforms, specifically Windows and OS X. It's still a [cross-platform](#) runtime, now including support for many Linux distributions.

See also [runtime](#).

Core CLR

The Common Language Runtime for [.NET 5 \(and .NET Core\) and later versions](#).

See [CLR](#).

CoreRT

In contrast to the [CLR](#), CoreRT is not a virtual machine, which means it doesn't include the facilities to generate and run code on-the-fly because it doesn't include a [JIT](#). It does, however, include the [GC](#) and the ability for runtime type identification (RTTI) and reflection. However, its type system is designed so that metadata for reflection isn't required. Not requiring metadata enables having an [AOT](#) tool chain that can link away superfluous metadata and (more importantly) identify code that the app doesn't use. CoreRT is in development.

See [Intro to CoreRT](#) and [.NET Runtime Lab](#).

cross-platform

The ability to develop and execute an application that can be used on multiple different operating systems, such as Linux, Windows, and iOS, without having to rewrite specifically for each one. This enables code reuse and consistency between applications on different platforms.

See [platform](#).

ecosystem

All of the runtime software, development tools, and community resources that are used to build and run applications for a given technology.

The term ".NET ecosystem" differs from similar terms such as ".NET stack" in its inclusion of third-party apps and libraries. Here's an example in a sentence:

- "The motivation behind [.NET Standard](#) was to establish greater uniformity in the .NET ecosystem."

framework

In general, a comprehensive collection of APIs that facilitates development and deployment of applications that are based on a particular technology. In this general sense, ASP.NET Core and Windows Forms are examples of application frameworks. The words **framework** and [library](#) are often used synonymously.

The word "framework" has a different meaning in the following terms:

- [framework libraries](#)
- [.NET Framework](#)
- [shared framework](#)
- [target framework](#)
- [TFM \(target framework moniker\)](#)
- [framework-dependent app](#)

Sometimes "framework" refers to an [implementation of .NET](#). For example, an article may call .NET 5+ a framework.

framework libraries

Meaning depends on context. May refer to the framework libraries for [.NET 5 \(and .NET Core\) and later versions](#), in which case it refers to the same libraries that [BCL](#) refers to. It may also refer to the [ASP.NET Core](#) framework libraries, which build on the BCL and provide additional APIs for web apps.

GC

Garbage collector.

The garbage collector is an implementation of automatic memory management. The GC frees memory occupied by objects that are no longer in use.

See [Garbage Collection](#).

IL

Intermediate language.

Higher-level .NET languages, such as C#, compile down to a hardware-agnostic instruction set, which is called Intermediate Language (IL). IL is sometimes referred to as MSIL (Microsoft IL) or CIL (Common IL).

JIT

Just-in-time compiler.

Similar to [AOT](#), this compiler translates [IL](#) to machine code that the processor understands. Unlike AOT, JIT compilation happens on demand and is performed on the same machine that the code needs to run on. Since JIT compilation occurs during execution of the application, compile time is part of the run time. Thus, JIT compilers have to balance time spent optimizing code against the savings that the resulting code can produce. But a JIT knows the actual hardware and can free developers from having to ship different implementations.

implementation of .NET

An implementation of .NET includes:

- One or more runtimes. Examples: [CLR](#), [CoreRT](#).
- A class library that implements a version of .NET Standard and may include additional APIs. Examples: the [BCLs](#) for [.NET Framework](#) and [.NET 5 \(and .NET Core\) and later versions](#).
- Optionally, one or more application frameworks. Examples: [ASP.NET](#), Windows Forms, and WPF are included in .NET Framework and .NET 5+.
- Optionally, development tools. Some development tools are shared among multiple implementations.

Examples of .NET implementations:

- [.NET Framework](#)
- [.NET 5 \(and .NET Core\) and later versions](#)
- [Universal Windows Platform \(UWP\)](#)
- [Mono](#)

For more information, see [.NET implementations](#).

library

A collection of APIs that can be called by apps or other libraries. A .NET library is composed of one or more [assemblies](#).

The words [library](#) and [framework](#) are often used synonymously.

Mono

Mono is an open source, [cross-platform .NET implementation](#) that is mainly used when a small runtime is required. It is the runtime that powers Xamarin applications on Android, Mac, iOS, tvOS, and watchOS and is focused primarily on apps that require a small footprint.

It supports all of the currently published .NET Standard versions.

Historically, Mono implemented the larger API of the .NET Framework and emulated some of the most popular capabilities on Unix. It is sometimes used to run .NET applications that rely on those capabilities on Unix.

Mono is typically used with a [just-in-time compiler](#), but it also features a full [static compiler \(ahead-of-time compilation\)](#) that is used on platforms like iOS.

See the [Mono documentation](#).

.NET

- In general, .NET is the umbrella term for [.NET Standard](#) and all [.NET implementations](#) and workloads.

- More specifically, .NET refers to the implementation of .NET that is recommended for all new development: [.NET 5 \(and .NET Core\) and later versions](#).

For example, the first meaning is intended in phrases such as "implementations of .NET" or "the .NET development platform." The second meaning is intended in names such as [.NET SDK](#) and [.NET CLI](#).

.NET is always fully capitalized, never ".Net".

See [.NET documentation](#)

.NET 5+

The plus sign after a version number means "and later versions." See [.NET 5 and later versions](#).

.NET 5 and later versions

A cross-platform, high-performance, open-source implementation of .NET. Also referred to as .NET 5+. Includes a Common Language Runtime ([CLR](#)), an AOT runtime ([CoreRT](#), in development), a Base Class Library ([BCL](#)), and the [.NET SDK](#).

Earlier versions of this .NET implementation are known as [.NET Core](#). .NET 5 is the next version following .NET Core 3.1. Version 4 was skipped to avoid confusing this newer implementation of .NET with the older implementation that is known as [.NET Framework](#). The current version of .NET Framework is 4.8.

See [.NET documentation](#).

.NET CLI

A cross-platform toolchain for developing applications and libraries for [.NET 5 \(and .NET Core\) and later versions](#). Also known as the .NET Core CLI.

See [.NET CLI](#).

.NET Core

See [.NET 5 and later versions](#).

.NET Framework

An [implementation of .NET](#) that runs only on Windows. Includes the Common Language Runtime ([CLR](#)), the Base Class Library ([BCL](#)), and application framework libraries such as [ASP.NET](#), Windows Forms, and WPF.

See [.NET Framework Guide](#).

.NET Native

A compiler tool chain that produces native code ahead-of-time ([AOT](#)), as opposed to just-in-time ([JIT](#)).

Compilation happens on the developer's machine similar to the way a C++ compiler and linker works. It removes unused code and spends more time optimizing it. It extracts code from libraries and merges them into the executable. The result is a single module that represents the entire app.

UWP is the application framework supported by .NET Native.

See [.NET Native documentation](#).

.NET SDK

A set of libraries and tools that allow developers to create .NET applications and libraries for [.NET 5 \(and .NET Core\) and later versions](#). Also known as the .NET Core SDK.

Includes the [.NET CLI](#) for building apps, .NET libraries and runtime for building and running apps, and the dotnet executable (`dotnet.exe`) that runs CLI commands and runs applications.

See [.NET SDK Overview](#).

.NET Standard

A formal specification of .NET APIs that are available in each [.NET implementation](#).

The .NET Standard specification is sometimes called a library. Because a library includes API implementations, not only specifications (interfaces), it's misleading to call .NET Standard a "library."

See [.NET Standard](#).

NGEN

Native (image) generation.

You can think of this technology as a persistent [JIT](#) compiler. It usually compiles code on the machine where the code is executed, but compilation typically occurs at install time.

package

A NuGet package—or just a package—is a `.zip` file with one or more assemblies of the same name along with additional metadata such as the author name.

The `.zip` file has a `.nupkg` extension and may contain assets, such as `.dll` files and `.xml` files, for use with multiple [target frameworks](#) and versions. When installed in an app or library, the appropriate assets are selected based on the target framework specified by the app or library. The assets that define the interface are in the `ref` folder, and the assets that define the implementation are in the `lib` folder.

platform

An operating system and the hardware it runs on, such as Windows, macOS, Linux, iOS, and Android.

Here are examples of usage in sentences:

- ".NET Core is a cross-platform implementation of .NET."
- "PCL profiles represent Microsoft platforms, while .NET Standard is agnostic to platform."

Legacy .NET documentation sometimes uses ".NET platform" to mean either an [implementation of .NET](#) or the [.NET stack](#) including all implementations. Both of these usages tend to get confused with the primary (OS/hardware) meaning, so we try to avoid these usages.

"Platform" has a different meaning in the phrase "developer platform," which refers to software that provides tools and libraries for building and running apps. .NET is a cross-platform, open-source developer platform for building many different types of applications.

POCO

A POCO—or a plain old class/[CLR](#) object—is a .NET data structure that contains only public properties or fields. A POCO shouldn't contain any other members, such as:

- methods

- events
- delegates

These objects are used primarily as data transfer objects (DTOs). A pure *POCO* will not inherit another object, or implement an interface. It's common for POCOs to be used with serialization.

runtime

In general, the execution environment for a managed program. The OS is part of the runtime environment but is not part of the .NET runtime. Here are some examples of .NET runtimes in this sense of the word:

- Common Language Runtime ([CLR](#))
- .NET Native (for UWP)
- Mono runtime

The word "runtime" has a different meaning in some contexts:

- *.NET runtime* on the [.NET 5 download page](#).

You can download the *.NET runtime* or other runtimes, such as the *ASP.NET Core runtime*. A *runtime* in this usage is the set of components that must be installed on a machine to run a [framework-dependent](#) app on the machine. The .NET runtime includes the [CLR](#) and the .NET [shared framework](#), which provides the [BCL](#).

- *.NET runtime libraries*

Refers to the same libraries that [BCL](#) refers to. However, other runtimes, such as the ASP.NET Core runtime, have different [shared frameworks](#), with additional libraries that build on the BCL.

- [Runtime Identifier \(RID\)](#).

Runtime here means the OS platform and CPU architecture that a .NET app runs on, for example:

`linux-x64`.

- Sometimes "runtime" is used in the sense of an [implementation of .NET](#), as in the following examples:

- "The various .NET runtimes implement specific versions of .NET Standard. ... Each .NET runtime version advertises the highest .NET Standard version it supports ..."
- "Libraries that are intended to run on multiple runtimes should target this framework." (referring to .NET Standard)

shared framework

Meaning depends on context. The *.NET shared framework* refers to the libraries included in the [.NET runtime](#). In this case, the *shared framework* for [.NET 5 \(and .NET Core\) and later versions](#) refers to the same libraries that [BCL](#) refers to.

There are other shared frameworks. The *ASP.NET Core shared framework* refers to the libraries included in the [ASP.NET Core runtime](#), which includes the BCL plus additional APIs for use by web apps.

For [framework-dependent apps](#), the shared framework consists of libraries that are contained in assemblies installed in a folder on the machine that runs the app. For [self-contained apps](#), the shared framework assemblies are included with the app.

For more information, see [Deep-dive into .NET Core primitives, part 2: the shared framework](#).

stack

A set of programming technologies that are used together to build and run applications.

"The .NET stack" refers to .NET Standard and all .NET implementations. The phrase "a .NET stack" may refer to one implementation of .NET.

target framework

The collection of APIs that a .NET app or library relies on.

An app or library can target a version of [.NET Standard](#) (for example, .NET Standard 2.0), which is a specification for a standardized set of APIs across all [.NET implementations](#). An app or library can also target a version of a specific .NET implementation, in which case it gets access to implementation-specific APIs. For example, an app that targets Xamarin.iOS gets access to Xamarin-provided iOS API wrappers.

For some target frameworks (for example, [.NET Framework](#)) the available APIs are defined by the assemblies that a .NET implementation installs on a system, which may include application framework APIs (for example, ASP.NET, WinForms). For package-based target frameworks, the framework APIs are defined by the packages installed in the app or library.

See [Target Frameworks](#).

TFM

Target framework moniker.

A standardized token format for specifying the [target framework](#) of a .NET app or library. Target frameworks are typically referenced by a short name, such as `net462`. Long-form TFMs (such as `.NETFramework,Version=4.6.2`) exist but are not generally used to specify a target framework.

See [Target Frameworks](#).

UWP

Universal Windows Platform.

An [implementation of .NET](#) that is used for building touch-enabled Windows applications and software for the Internet of Things (IoT). It's designed to unify the different types of devices that you may want to target, including PCs, tablets, phones, and even the Xbox. UWP provides many services, such as a centralized app store, an execution environment (AppContainer), and a set of Windows APIs to use instead of Win32 (WinRT). Apps can be written in C++, C#, Visual Basic, and JavaScript. When using C# and Visual Basic, the .NET APIs are provided by [.NET 5 \(and .NET Core\) and later versions](#).

workload

A type of app someone is building. More generic than [app model](#). For example, at the top of every .NET documentation page, including this one, is a drop-down list for **Workloads**, which lets you switch to documentation for **Web**, **Mobile**, **Cloud**, **Desktop**, and **Machine Learning & Data**.

In some contexts, *workload* refers to a collection of Visual Studio features that you can choose to install to support a particular type of app. For an example, see [Select a workload](#).

See also

- [.NET fundamentals](#)
- [.NET Framework Guide](#)
- [ASP.NET Overview](#)

- [ASP.NET Core Overview](#)

.NET 6 C# console app template generates top-level statements

9/20/2022 • 5 minutes to read • [Edit Online](#)

Starting with .NET 6, the project template for new C# console apps generates the following code in the `Program.cs` file:

```
// See https://aka.ms/new-console-template for more information
Console.WriteLine("Hello, World!");
```

The new output uses recent C# features that simplify the code you need to write for a program. For .NET 5 and earlier versions, the console app template generates the following code:

```
using System;

namespace MyApp // Note: actual namespace depends on the project name.
{
    internal class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

These two forms represent the same program. Both are valid with C# 10.0. When you use the newer version, you only need to write the body of the `Main` method. The compiler synthesizes a `Program` class with a `Main` method and places all your top level statements in that `Main` method. You don't need to include the other program elements, the compiler generates them for you. You can learn more about the code the compiler generates when you use top level statements in the article on [top level statements](#) in the C# Guide's fundamentals section.

You have two options to work with tutorials that haven't been updated to use .NET 6+ templates:

- Use the new program style, adding new top-level statements as you add features.
- Convert the new program style to the older style, with a `Program` class and a `Main` method.

If you want to use the old templates, see [Use the old program style](#) later in this article.

Use the new program style

The features that make the new program simpler are *top-level statements*, *global using directives*, and *implicit using directives*.

The term [top-level statements](#) means the compiler generates the class and method elements for your main program. The compiler generated class and `Main` method are declared in the global namespace. You can look at the code for the new application and imagine that it contains the statements inside the `Main` method generated by earlier templates, but in the global namespace.

You can add more statements to the program, just like you can add more statements to your `Main` method in

the traditional style. You can access `args` (command-line arguments), use `await`, and set the exit code. You can even add functions. They're created as local functions nested inside the generated `Main` method. Local functions can't include any access modifiers (for example, `public` or `protected`).

Both top-level statements and `implicit using directives` simplify the code that makes up your application. To follow an existing tutorial, add any new statements to the `Program.cs` file generated by the template. You can imagine that the statements you write are between the open and closing braces in the `Main` method in the instructions of the tutorial.

If you'd prefer to use the older format, you can copy the code from the second example in this article, and continue the tutorial as before.

You can learn more about top-level statements in the tutorial exploration on [top-level statements](#).

Implicit `using` directives

The term *implicit using directives* means the compiler automatically adds a set of `using` directives based on the project type. For console applications, the following directives are implicitly included in the application:

- `using System;`
- `using System.IO;`
- `using System.Collections.Generic;`
- `using System.Linq;`
- `using System.Net.Http;`
- `using System.Threading;`
- `using System.Threading.Tasks;`

Other application types include more namespaces that are common for those application types.

If you need `using` directives that aren't implicitly included, you can add them to the `.cs` file that contains top-level statements or to other `.cs` files. For `using` directives that you need in all of the `.cs` files in an application, use [global using directives](#).

Disable implicit `using` directives

If you want to remove this behavior and manually control all namespaces in your project, add

`<ImplicitUsings>disable</ImplicitUsings>` to your project file in the `<PropertyGroup>` element, as shown in the following example:

```
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    ...
    <ImplicitUsings>disable</ImplicitUsings>
  </PropertyGroup>

</Project>
```

Global `using` directives

A *global using directive* imports a namespace for your whole application instead of a single file. These global directives can be added either by adding a `<Using>` item to the project file, or by adding the `global using` directive to a code file.

You can also add a `<Using>` item with a `Remove` attribute to your project file to remove a specific *implicit using*

[directive](#). For example, if the implicit `using` directives feature is turned on with `<ImplicitUsings>enable</ImplicitUsings>`, adding the following `<using>` item removes the `System.Net.Http` namespace from those that are implicitly imported:

```
<ItemGroup>
  <Using Remove="System.Net.Http" />
</ItemGroup>
```

Use the old program style

While a .NET 6 console app template generates the new style of top-level statements programs, using .NET 5 doesn't. By creating a .NET 5 project, you'll receive the old program style. Then, you can edit the project file to target .NET 6 but retain the old program style for the *Program.cs* file.

IMPORTANT

Creating a project that targets .NET 5 requires the .NET 5 templates. The .NET 5 templates can be installed manually with the `dotnet new --install` command or by [installing the .NET 5 SDK](#).

1. Create a new project that targets .NET 5.

```
dotnet new console --framework net5.0
```

2. Open the project file in a text editor and change `<TargetFramework>net5.0</TargetFramework>` to `<TargetFramework>net6.0</TargetFramework>`.

Here's a file diff that illustrates the changes:

```
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <OutputType>Exe</OutputType>
    - <TargetFramework>net5.0</TargetFramework>
    + <TargetFramework>net6.0</TargetFramework>
  </PropertyGroup>

</Project>
```

3. Optional step: you can still use some of the newer .NET 6 and C# features by adding the properties for implicit `using` directives and nullable context to the project file.

```
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>net6.0</TargetFramework>
    + <ImplicitUsings>enable</ImplicitUsings>
    + <Nullable>enable</Nullable>
  </PropertyGroup>

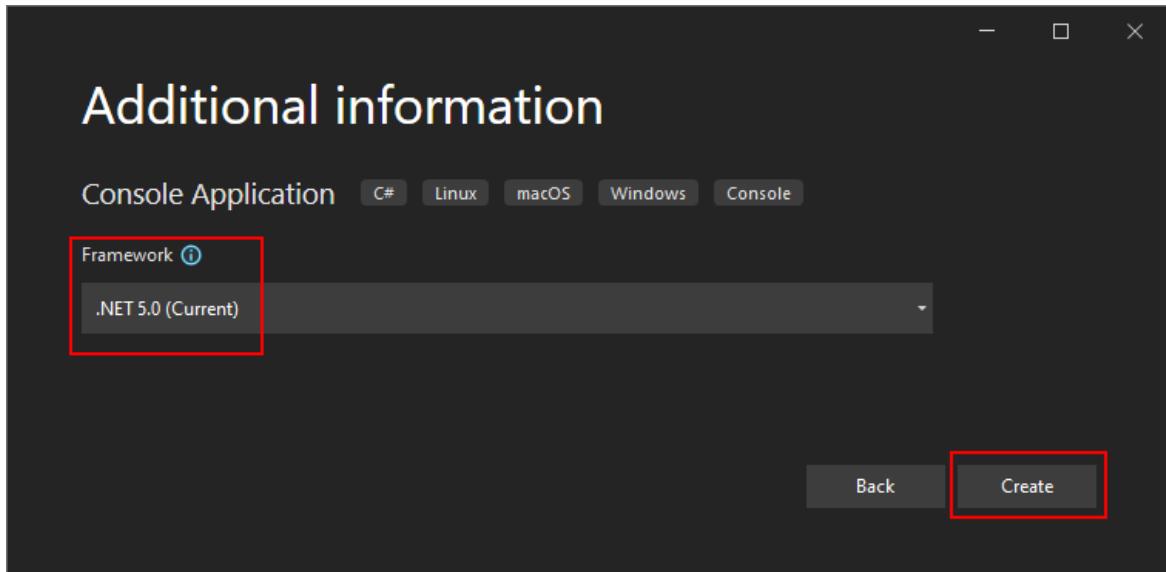
</Project>
```

Use the old program style in Visual Studio

When you create a new console project in Visual Studio, you're prompted with a dropdown box that identifies

which target framework you want to use. Change that value to **5.0**. After the project is created, edit the project file to change it back to **6.0**.

- When you create a new project, the setup steps will navigate to the **Additional information** setup page. On this page, change the framework setting from **.NET 6.0 (Long-term support)** to **.NET 5.0**, and then select the **Create** button.



- After your project is created, find the **Project Explorer** pane. Double-click on the project file and change `<TargetFramework>net5.0</TargetFramework>` to `<TargetFramework>net6.0</TargetFramework>`.

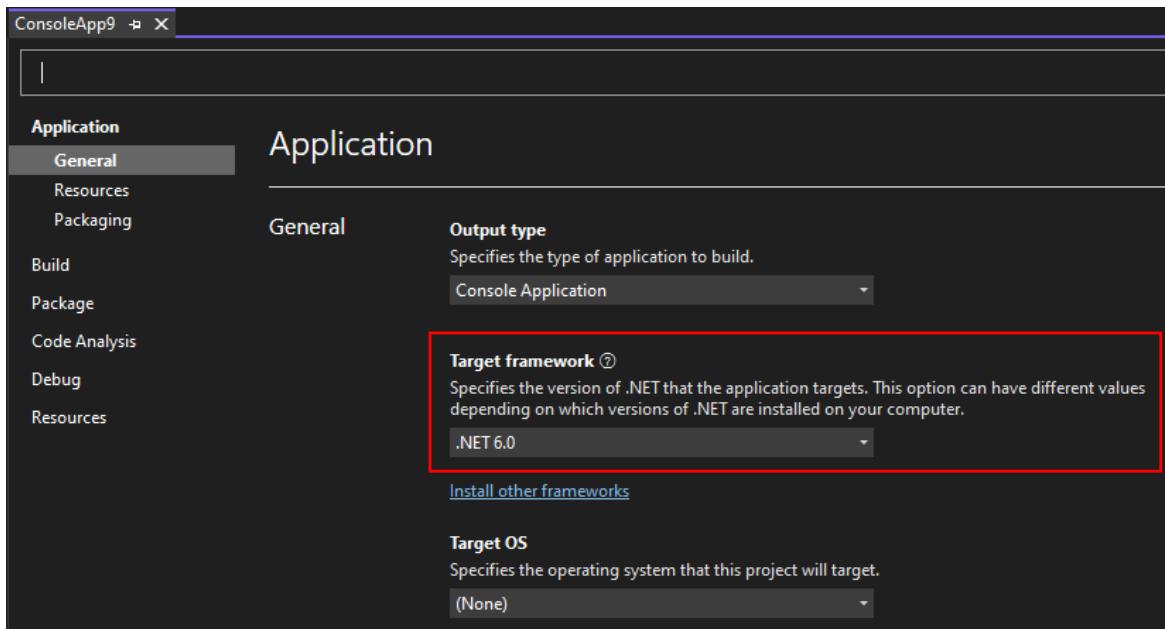
Here's a file diff that illustrates the changes:

```
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <OutputType>Exe</OutputType>
    - <TargetFramework>net5.0</TargetFramework>
    + <TargetFramework>net6.0</TargetFramework>
  </PropertyGroup>

</Project>
```

Alternatively, you can right-click on the project in the **Solution Explorer** pane, and select **Properties**. This opens up a settings page where you can change the **Target framework**.



3. Optional step: you can still use some of the newer .NET 6 and C# features by adding the properties for `implicit using` directives and `nullable context` to the project file.

```
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>net6.0</TargetFramework>
+   <ImplicitUsings>enable</ImplicitUsings>
+   <Nullable>enable</Nullable>
  </PropertyGroup>

</Project>
```

Template feedback

[Top-level statements](#) is a new feature in .NET 6. Add an up or down vote in [GitHub issue #27420](#) to let us know if you support the use of this feature in project templates.

Tutorial: Create a .NET console application using Visual Studio

9/20/2022 • 7 minutes to read • [Edit Online](#)

This tutorial shows how to create and run a .NET console application in Visual Studio 2022.

Prerequisites

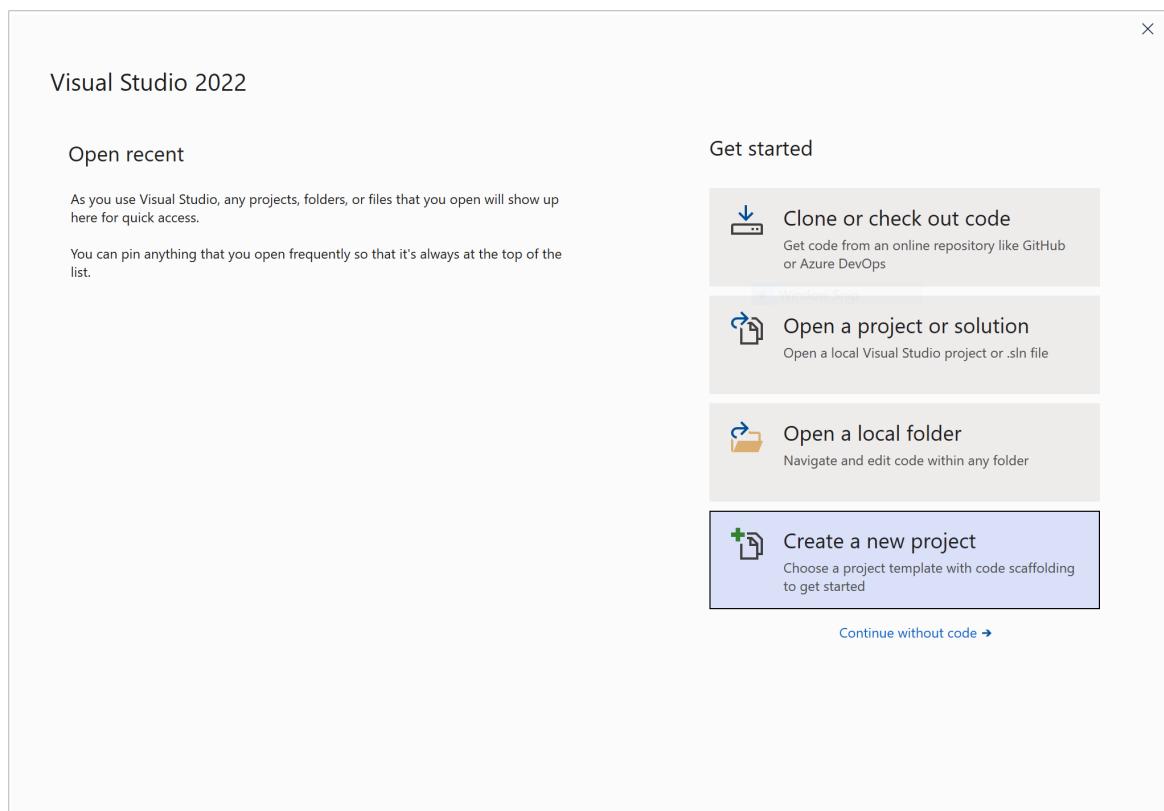
- [Visual Studio 2022 version 17.1 or later](#) with the **.NET desktop development** workload installed. The .NET 6 SDK is automatically installed when you select this workload.

For more information, see [Install the .NET SDK with Visual Studio](#).

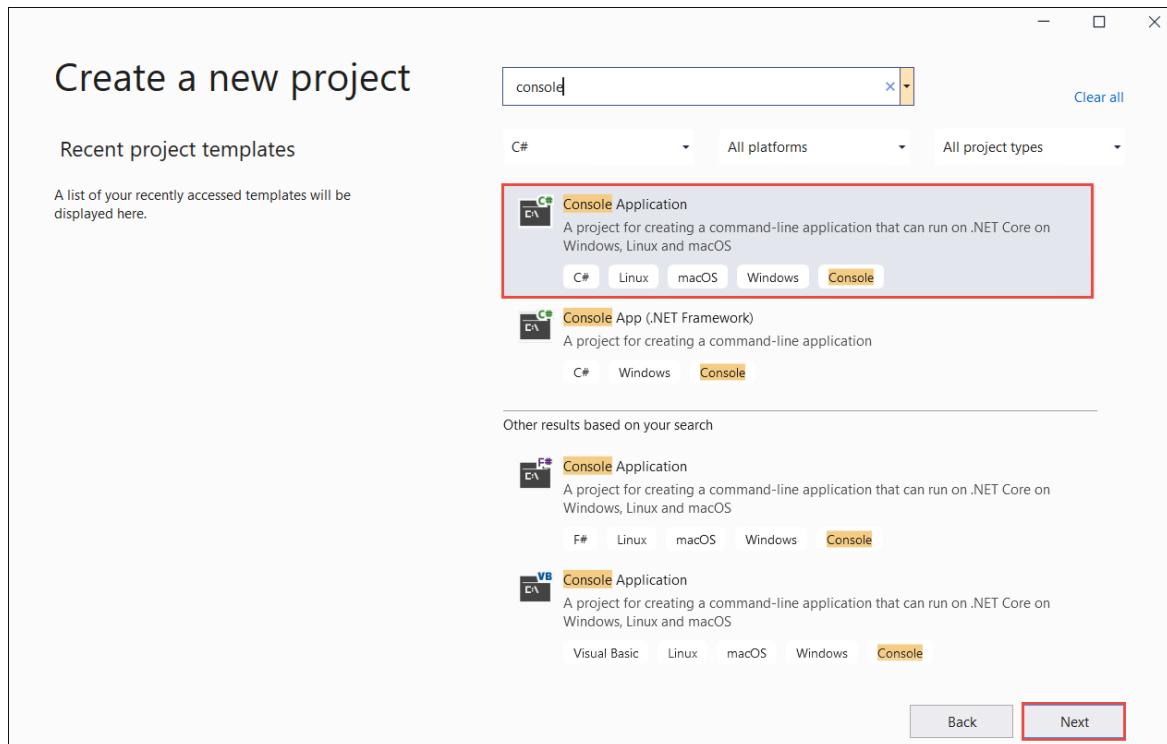
Create the app

Create a .NET console app project named "HelloWorld".

1. Start Visual Studio 2022.
2. On the start page, choose **Create a new project**.



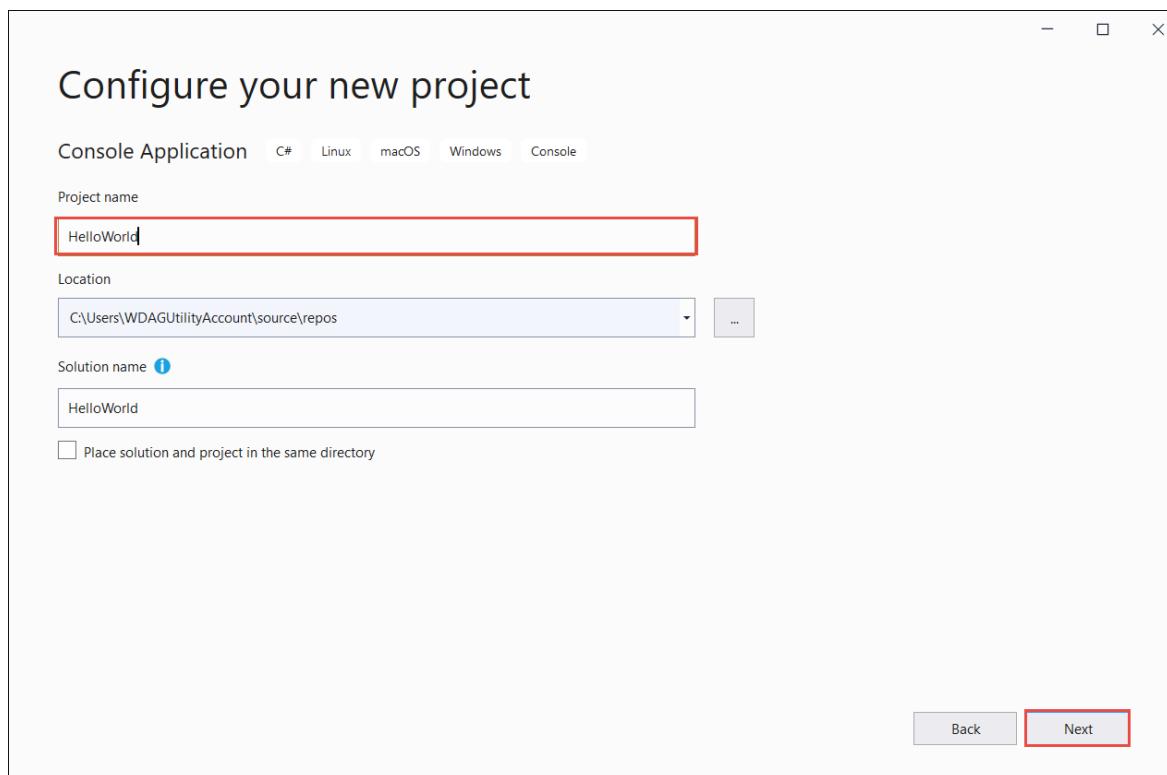
3. On the **Create a new project** page, enter **console** in the search box. Next, choose **C#** or **Visual Basic** from the language list, and then choose **All platforms** from the platform list. Choose the **Console Application** template, and then choose **Next**.



TIP

If you don't see the .NET templates, you're probably missing the required workload. Under the **Not finding what you're looking for?** message, choose the **Install more tools and features** link. The Visual Studio Installer opens. Make sure you have the **.NET desktop development** workload installed.

4. In the **Configure your new project** dialog, enter **HelloWorld** in the **Project name** box. Then choose **Next**.



5. In the **Additional information** dialog, select **.NET 6 (Long-term support)**, and then select **Create**.

The template creates a simple application that displays "Hello World" in the console window. The code is in the *Program.cs* or *Program.vb* file:

```
Console.WriteLine("Hello, World!");
```

```
Imports System

Module Program
    Sub Main(args As String())
        Console.WriteLine("Hello World!")
    End Sub
End Module
```

If the language you want to use is not shown, change the language selector at the top of the page.

6. For C#, the code is just a line that calls the [Console.WriteLine\(String\)](#) method to display "Hello World!" in the console window. Replace the contents of *Program.cs* with the following code:

```
namespace HelloWorld
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

' This step of the tutorial applies only to C#.

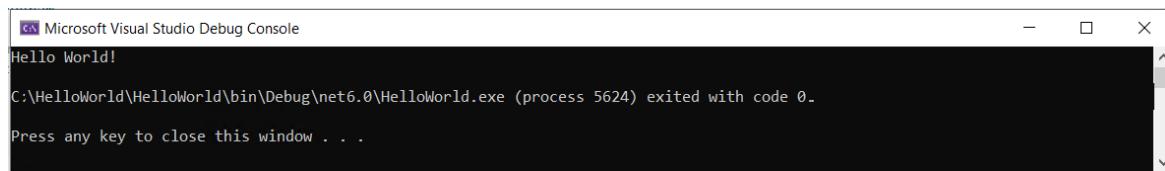
The code defines a class, `Program`, with a single method, `Main`, that takes a `String` array as an argument. `Main` is the application entry point, the method that's called automatically by the runtime when it launches the application. Any command-line arguments supplied when the application is launched are available in the `args` array.

In the latest version of C#, a new feature named [top-level statements](#) lets you omit the `Program` class and the `Main` method. Most existing C# programs don't use top-level statements, so this tutorial doesn't use this new feature. But it's available in C# 10, and whether you use it in your programs is a matter of style preference.

Run the app

1. Press **Ctrl+F5** to run the program without debugging.

A console window opens with the text "Hello World!" printed on the screen.



2. Press any key to close the console window.

Enhance the app

Enhance the application to prompt the user for their name and display it along with the date and time.

1. In *Program.cs* or *Program.vb*, replace the contents of the `Main` method, which is the line that calls `Console.WriteLine`, with the following code:

```
Console.WriteLine("What is your name?");
var name = Console.ReadLine();
var currentDate = DateTime.Now;
Console.WriteLine($"{Environment.NewLine}Hello, {name}, on {currentDate:d} at {currentDate:t}!");
Console.Write($"{Environment.NewLine}Press any key to exit...");
Console.ReadKey(true);
```

```
Console.WriteLine("What is your name?")
Dim name = Console.ReadLine()
Dim currentDate = DateTime.Now
Console.WriteLine($"{Environment.NewLine}Hello, {name}, on {currentDate:d} at {currentDate:t}")
Console.Write($"{Environment.NewLine}Press any key to exit...")
Console.ReadKey(True)
```

This code displays a prompt in the console window and waits until the user enters a string followed by the Enter key. It stores this string in a variable named `name`. It also retrieves the value of the `DateTime.Now` property, which contains the current local time, and assigns it to a variable named `currentDate`. And it displays these values in the console window. Finally, it displays a prompt in the console window and calls the `Console.ReadKey(Boolean)` method to wait for user input.

`Environment.NewLine` is a platform-independent and language-independent way to represent a line break. Alternatives are `\n` in C# and `vbcrlf` in Visual Basic.

The dollar sign (`$`) in front of a string lets you put expressions such as variable names in curly braces in the string. The expression value is inserted into the string in place of the expression. This syntax is referred to as [interpolated strings](#).

2. Press **Ctrl+F5** to run the program without debugging.
3. Respond to the prompt by entering a name and pressing the Enter key.

```
What is your name?
Maira

Hello, Maira, on 12/3/2019 at 3:36 AM!

Press any key to exit...
```

4. Press any key to close the console window.

Additional resources

- [Current releases and long-term support releases](#)

Next steps

In this tutorial, you created a .NET console application. In the next tutorial, you debug the app.

[Debug a .NET console application using Visual Studio](#)

This tutorial shows how to create and run a .NET console application in Visual Studio 2019.

Prerequisites

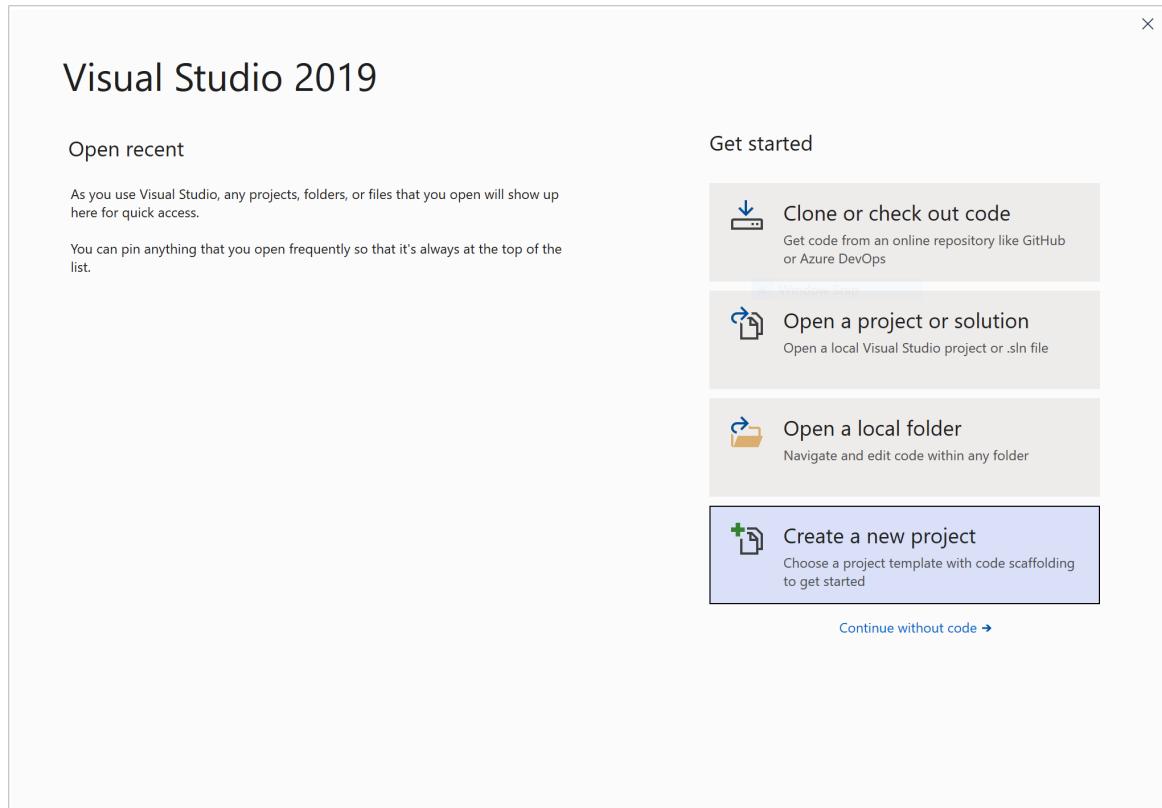
- [Visual Studio 2019 version 16.9.2 or a later version](#) with the **.NET Core cross-platform development** workload installed. The .NET 5.0 SDK is automatically installed when you select this workload.

For more information, see [Install the .NET SDK with Visual Studio](#).

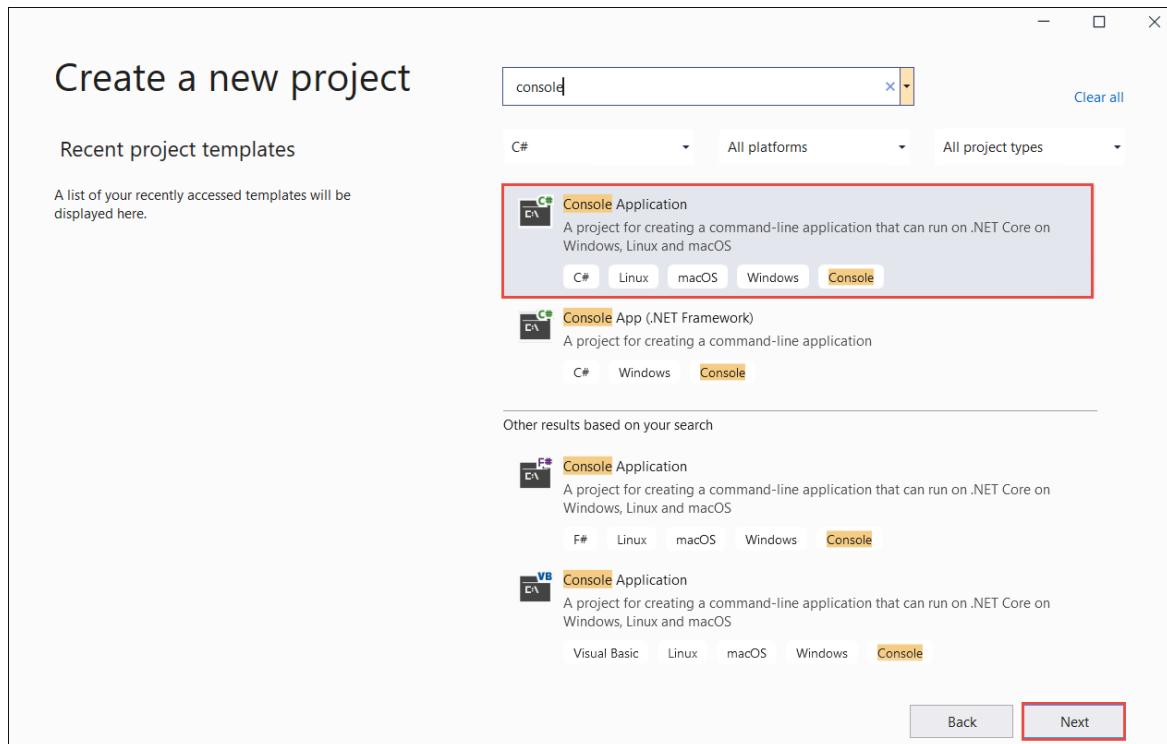
Create the app

Create a .NET console app project named "HelloWorld".

1. Start Visual Studio 2019.
2. On the start page, choose **Create a new project**.



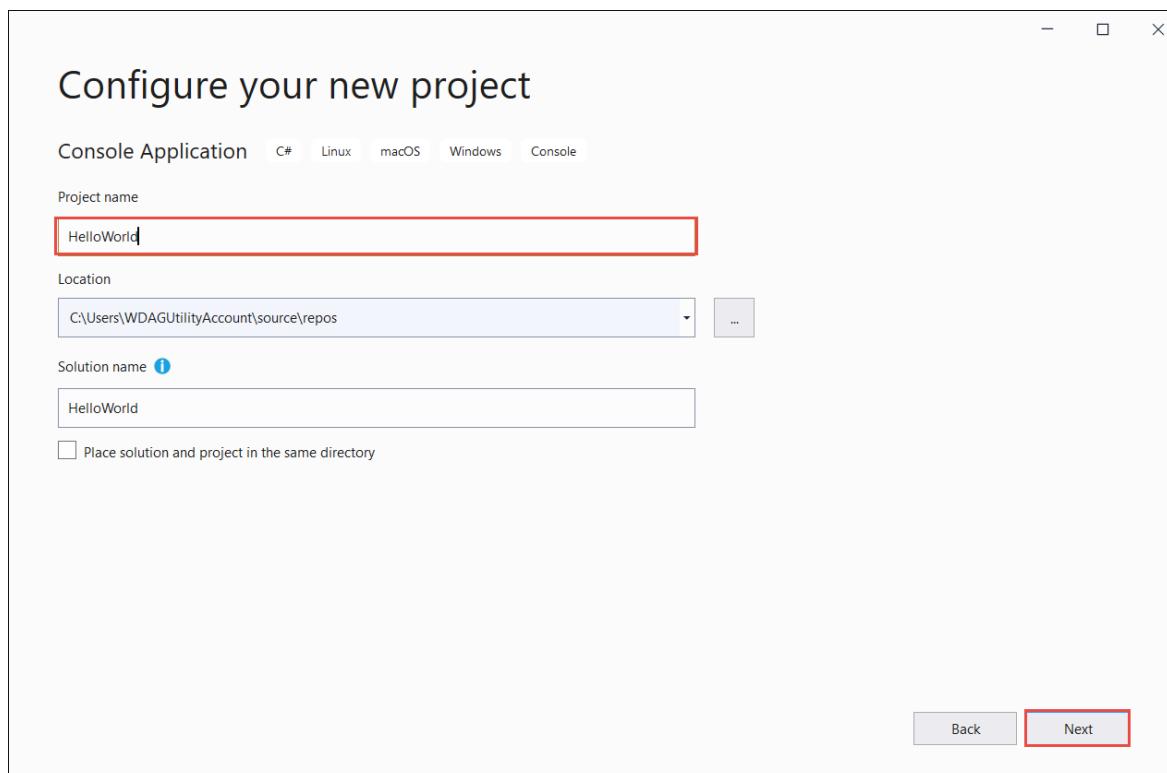
3. On the **Create a new project** page, enter **console** in the search box. Next, choose **C#** or **Visual Basic** from the language list, and then choose **All platforms** from the platform list. Choose the **Console Application** template, and then choose **Next**.



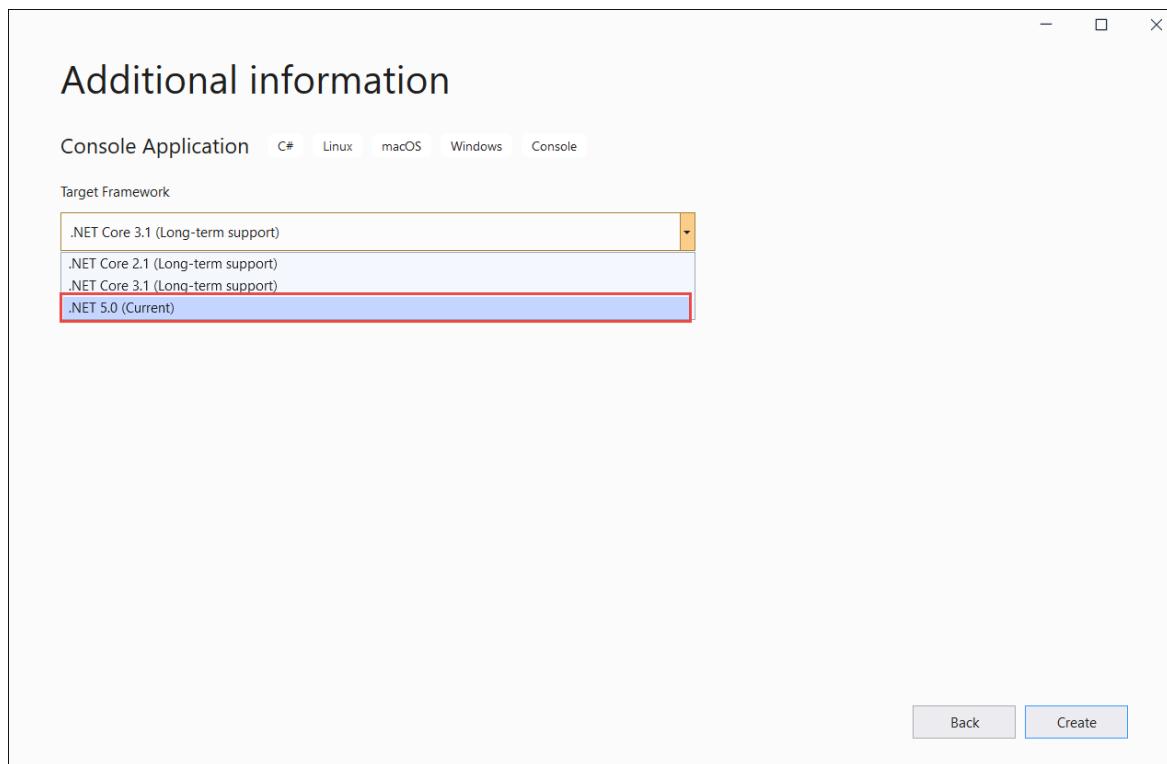
TIP

If you don't see the .NET templates, you're probably missing the required workload. Under the **Not finding what you're looking for?** message, choose the **Install more tools and features** link. The Visual Studio Installer opens. Make sure you have the **.NET Core cross-platform development** workload installed.

4. In the **Configure your new project** dialog, enter **HelloWorld** in the **Project name** box. Then choose **Next**.



5. In the **Additional information** dialog, select **.NET 5.0 (Current)**, and then select **Create**.



The template creates a simple "Hello World" application. It calls the `Console.WriteLine(String)` method to display "Hello World!" in the console window.

The template code defines a class, `Program`, with a single method, `Main`, that takes a `String` array as an argument:

```
using System;

namespace HelloWorld
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

```
Imports System

Module Program
    Sub Main(args As String())
        Console.WriteLine("Hello World!")
    End Sub
End Module
```

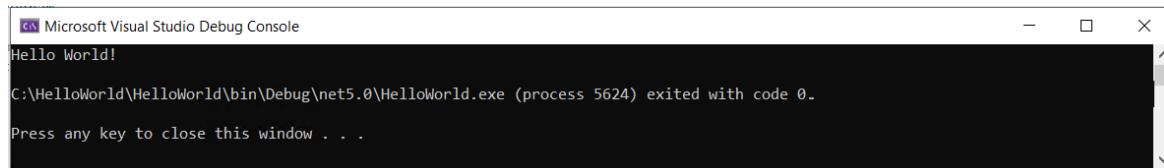
`Main` is the application entry point, the method that's called automatically by the runtime when it launches the application. Any command-line arguments supplied when the application is launched are available in the `args` array.

If the language you want to use is not shown, change the language selector at the top of the page.

Run the app

1. Press **Ctrl+F5** to run the program without debugging.

A console window opens with the text "Hello World!" printed on the screen.



The screenshot shows a Microsoft Visual Studio Debug Console window. The title bar says "Microsoft Visual Studio Debug Console". The main area displays the text "Hello World!". Below it, the command line shows the path "C:\HelloWorld\HelloWorld\bin\Debug\net5.0\HelloWorld.exe (process 5624) exited with code 0.". At the bottom, there is a prompt: "Press any key to close this window . . .".

2. Press any key to close the console window.

Enhance the app

Enhance the application to prompt the user for their name and display it along with the date and time.

1. In *Program.cs* or *Program.vb*, replace the contents of the `Main` method, which is the line that calls `Console.WriteLine`, with the following code:

```
Console.WriteLine("What is your name?");
var name = Console.ReadLine();
var currentDate = DateTime.Now;
Console.WriteLine($"{Environment.NewLine}Hello, {name}, on {currentDate:d} at {currentDate:t}!");
Console.Write($"{Environment.NewLine}Press any key to exit...");
Console.ReadKey(true);
```

```
Console.WriteLine("What is your name?")
Dim name = Console.ReadLine()
Dim currentDate = DateTime.Now
Console.WriteLine($"{Environment.NewLine}Hello, {name}, on {currentDate:d} at {currentDate:t}")
Console.Write($"{Environment.NewLine}Press any key to exit...")
Console.ReadKey(True)
```

This code displays a prompt in the console window and waits until the user enters a string followed by the Enter key. It stores this string in a variable named `name`. It also retrieves the value of the `DateTime.Now` property, which contains the current local time, and assigns it to a variable named `currentDate`. And it displays these values in the console window. Finally, it displays a prompt in the console window and calls the `Console.ReadKey(Boolean)` method to wait for user input.

`Environment.NewLine` is a platform-independent and language-independent way to represent a line break. Alternatives are `\n` in C# and `vbcrlf` in Visual Basic.

The dollar sign (`$`) in front of a string lets you put expressions such as variable names in curly braces in the string. The expression value is inserted into the string in place of the expression. This syntax is referred to as **interpolated strings**.

2. Press **Ctrl+F5** to run the program without debugging.
3. Respond to the prompt by entering a name and pressing the Enter key.

```
C:\Projects\HelloWorld\HelloWorld\bin\Debug\netcoreapp3.1\HelloWorld.exe
What is your name?
Maira
Hello, Maira, on 12/3/2019 at 3:36 AM!
Press any key to exit...
```

4. Press any key to close the console window.

Additional resources

- [Current releases and long-term support releases](#)

Next steps

In this tutorial, you created a .NET console application. In the next tutorial, you debug the app.

[Debug a .NET console application using Visual Studio](#)

This tutorial is only available for .NET 5 and .NET 6. Select one of those options at the top of the page.

Tutorial: Debug a .NET console application using Visual Studio

9/20/2022 • 13 minutes to read • [Edit Online](#)

This tutorial introduces the debugging tools available in Visual Studio.

Prerequisites

- This tutorial works with the console app that you create in [Create a .NET console application using Visual Studio](#).

Use Debug build configuration

Debug and *Release* are Visual Studio's built-in build configurations. You use the Debug build configuration for debugging and the Release configuration for the final release distribution.

In the Debug configuration, a program compiles with full symbolic debug information and no optimization. Optimization complicates debugging, because the relationship between source code and generated instructions is more complex. The release configuration of a program has no symbolic debug information and is fully optimized.

By default, Visual Studio uses the Debug build configuration, so you don't need to change it before debugging.

1. Start Visual Studio.
2. Open the project that you created in [Create a .NET console application using Visual Studio](#).

The current build configuration is shown on the toolbar. The following toolbar image shows that Visual Studio is configured to compile the Debug version of the app:



Set a breakpoint

A *breakpoint* temporarily interrupts the execution of the application before the line with the breakpoint is executed.

1. Set a *breakpoint* on the line that displays the name, date, and time, by clicking in the left margin of the code window on that line. The left margin is to the left of the line numbers. Other ways to set a breakpoint are by placing the cursor in the line of code and then pressing F9 or choosing **Debug > Toggle Breakpoint** from the menu bar.

As the following image shows, Visual Studio indicates the line on which the breakpoint is set by highlighting it and displaying a red dot in the left margin.

A screenshot of the Visual Studio IDE showing the code editor for a C# file named Program.cs. The code defines a namespace HelloWorld and a class Program with a Main method. A red rectangle highlights the line of code where the breakpoint is set: "Console.WriteLine(\$"Hello, {name}, on {currentDate:d} at {currentDate:t}!");". The status bar at the bottom shows "No issues found".

2. Press F5 to run the program in Debug mode. Another way to start debugging is by choosing **Debug > Start Debugging** from the menu.
3. Enter a string in the console window when the program prompts for a name, and then press Enter.
4. Program execution stops when it reaches the breakpoint and before the `Console.WriteLine` method executes. The **Locals** window displays the values of variables that are defined in the currently executing method.

A screenshot of the Visual Studio IDE during a debug session. The code editor shows the same C# code as before. The Locals window is open, displaying variable values: args (string[0]), name ("jack"), and currentDate (4/26/2021 1:36:13 PM). The Diagnostic Tools window is also visible, showing a timeline of events and process memory usage. The status bar at the bottom shows "No issues found".

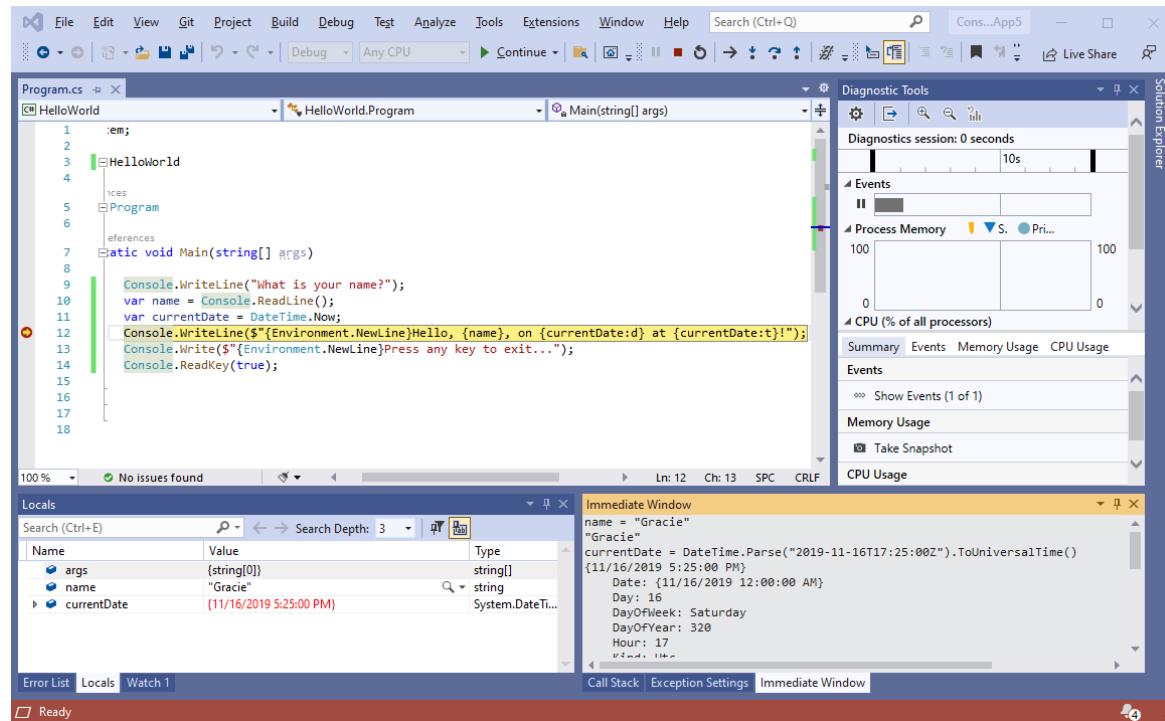
Use the Immediate window

The **Immediate** window lets you interact with the application you're debugging. You can interactively change the value of variables to see how it affects your program.

1. If the **Immediate** window is not visible, display it by choosing **Debug > Windows > Immediate**.
2. Enter `name = "Gracie"` in the **Immediate** window and press the Enter key.

3. Enter `currentDate = DateTime.Parse("2019-11-16T17:25:00Z").ToUniversalTime()` in the **Immediate** window and press the Enter key.

The **Immediate** window displays the value of the string variable and the properties of the `DateTime` value. In addition, the values of the variables are updated in the **Locals** window.



4. Press F5 to continue program execution. Another way to continue is by choosing **Debug > Continue** from the menu.

The values displayed in the console window correspond to the changes you made in the **Immediate** window.

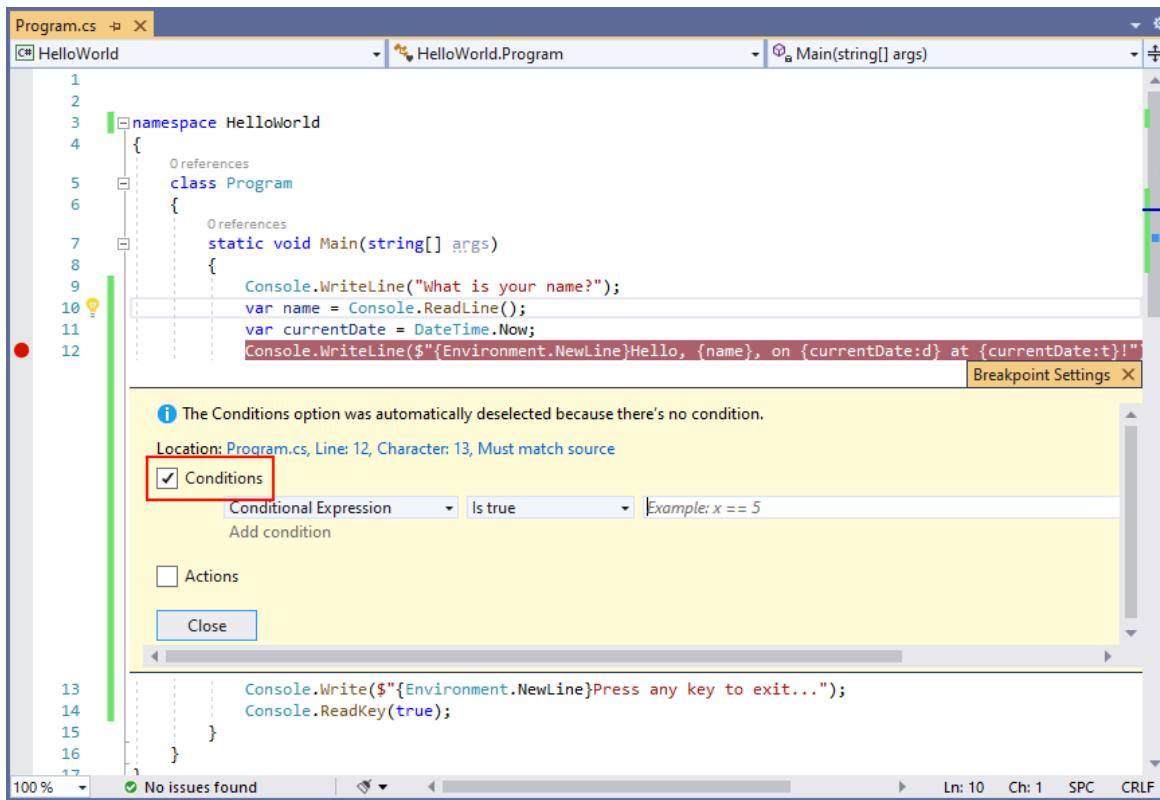
```
What is your name?  
jack  
  
Hello, Gracie, on 11/16/2019 at 5:25 PM!  
  
Press any key to exit...
```

5. Press any key to exit the application and stop debugging.

Set a conditional breakpoint

The program displays the string that the user enters. What happens if the user doesn't enter anything? You can test this with a useful debugging feature called a *conditional breakpoint*.

1. Right-click on the red dot that represents the breakpoint. In the context menu, select **Conditions** to open the **Breakpoint Settings** dialog. Select the box for **Conditions** if it's not already selected.



2. For the **Conditional Expression**, enter the following code in the field that shows example code that tests if `x` is 5.

```
String.IsNullOrEmpty(name)
```

```
String.IsNullOrEmpty(name)
```

Each time the breakpoint is hit, the debugger calls the `String.IsNullOrEmpty(name)` method, and it breaks on this line only if the method call returns `true`.

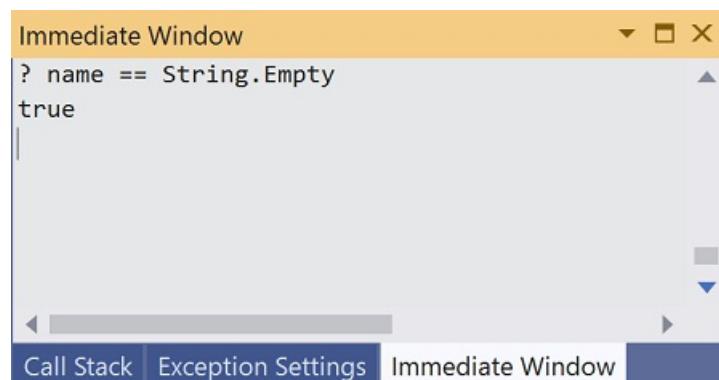
Instead of a conditional expression, you can specify a *hit count*, which interrupts program execution before a statement is executed a specified number of times. Another option is to specify a *filter condition*, which interrupts program execution based on such attributes as a thread identifier, process name, or thread name.

3. Select **Close** to close the dialog.
4. Start the program with debugging by pressing F5.
5. In the console window, press the Enter key when prompted to enter your name.
6. Because the condition you specified (`name` is either `null` or `String.Empty`) has been satisfied, program execution stops when it reaches the breakpoint and before the `Console.WriteLine` method executes.
7. Select the **Locals** window, which shows the values of variables that are local to the currently executing method. In this case, `Main` is the currently executing method. Observe that the value of the `name` variable is `""`, or `String.Empty`.
8. Confirm the value is an empty string by entering the following statement in the **Immediate** window and pressing Enter. The result is `true`.

```
? name == String.Empty
```

```
? String.IsNullOrEmpty(name)
```

The question mark directs the immediate window to [evaluate an expression](#).



9. Press F5 to continue program execution.
10. Press any key to close the console window and stop debugging.
11. Clear the breakpoint by clicking on the dot in the left margin of the code window. Other ways to clear a breakpoint are by pressing F9 or choosing Debug > Toggle Breakpoint while the line of code is selected.

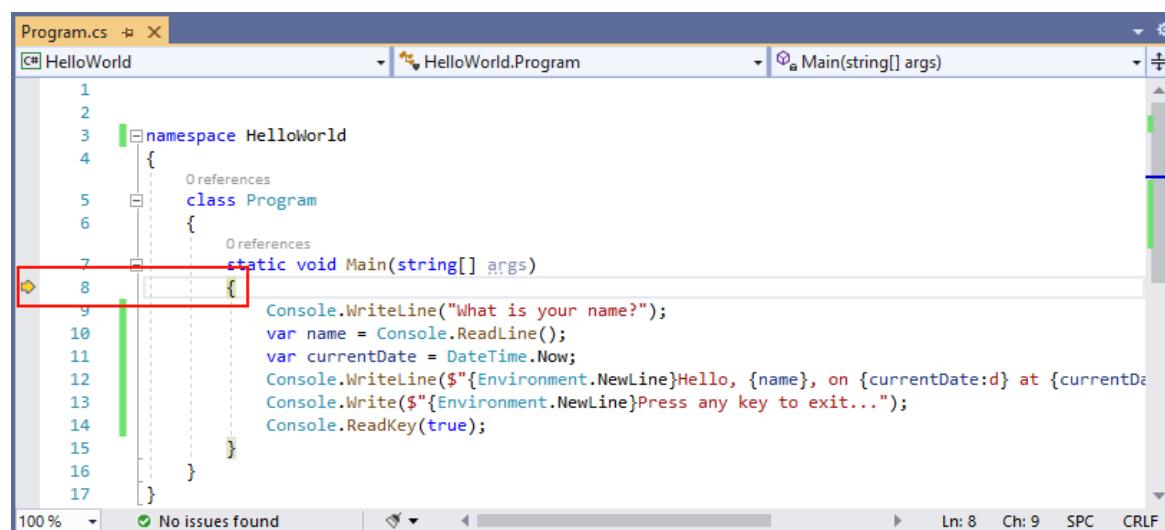
Step through a program

Visual Studio also allows you to step line by line through a program and monitor its execution. Ordinarily, you'd set a breakpoint and follow program flow through a small part of your program code. Since this program is small, you can step through the entire program.

1. Choose **Debug > Step Into**. Another way to debug one statement at a time is by pressing F11.

Visual Studio highlights and displays an arrow beside the next line of execution.

C#



Visual Basic

```
Imports System
Module Program
    Sub Main(args As String())
        Console.WriteLine("What is your name?")
        Dim name = Console.ReadLine()
        Dim currentDate = DateTime.Now
        Console.WriteLine($"{Environment.NewLine}Hello, {name}, on {currentDate:d} at {currentDate:t}")
        Console.Write($"{Environment.NewLine}Press any key to exit...")
        Console.ReadKey(True)
    End Sub
End Module
```

At this point, the **Locals** window shows that the `args` array is empty, and `name` and `currentDate` have default values. In addition, Visual Studio has opened a blank console window.

2. Press F11. Visual Studio now highlights the next line of execution. The **Locals** window is unchanged, and the console window remains blank.

C#

```
namespace HelloWorld
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("What is your name?"); ⏴1ms elapsed
            var name = Console.ReadLine();
            var currentDate = DateTime.Now;
            Console.WriteLine($"{Environment.NewLine}Hello, {name}, on {currentDate:d} at {currentDate:t}");
            Console.Write($"{Environment.NewLine}Press any key to exit...");
            Console.ReadKey(true);
        }
    }
}
```

Visual Basic

```
Imports System
Module Program
    Sub Main(args As String())
        Console.WriteLine("What is your name?") ⏴1ms elapsed
        Dim name = Console.ReadLine()
        Dim currentDate = DateTime.Now
        Console.WriteLine($"{Environment.NewLine}Hello, {name}, on {currentDate:d} at {currentDate:t}")
        Console.Write($"{Environment.NewLine}Press any key to exit...")
        Console.ReadKey(True)
    End Sub
End Module
```

3. Press F11. Visual Studio highlights the statement that includes the `name` variable assignment. The **Locals** window shows that `name` is `null`, and the console window displays the string "What is your name?".

4. Respond to the prompt by entering a string in the console window and pressing Enter. The console is unresponsive, and the string you entered isn't displayed in the console window, but the `Console.ReadLine` method will nevertheless capture your input.
5. Press F11. Visual Studio highlights the statement that includes the `currentDate` variable assignment. The **Locals** window shows the value returned by the call to the `Console.ReadLine` method. The console window also displays the string you entered at the prompt.
6. Press F11. The **Locals** window shows the value of the `currentDate` variable after the assignment from the `DateTime.Now` property. The console window is unchanged.
7. Press F11. Visual Studio calls the `Console.WriteLine(String, Object, Object)` method. The console window displays the formatted string.
8. Choose **Debug > Step Out**. Another way to stop step-by-step execution is by pressing **Shift+F11**.
The console window displays a message and waits for you to press a key.
9. Press any key to close the console window and stop debugging.

Use Release build configuration

Once you've tested the Debug version of your application, you should also compile and test the Release version. The Release version incorporates compiler optimizations that can sometimes negatively affect the behavior of an application. For example, compiler optimizations that are designed to improve performance can create race conditions in multithreaded applications.

To build and test the Release version of your console application, change the build configuration on the toolbar from **Debug** to **Release**.



When you press F5 or choose **Build Solution** from the **Build** menu, Visual Studio compiles the Release version of the application. You can test it as you did the Debug version.

Next steps

In this tutorial, you used Visual Studio debugging tools. In the next tutorial, you publish a deployable version of the app.

[Publish a .NET console application using Visual Studio](#)

This tutorial introduces the debugging tools available in Visual Studio.

Prerequisites

- This tutorial works with the console app that you create in [Create a .NET console application using Visual Studio](#).

Use Debug build configuration

Debug and *Release* are Visual Studio's built-in build configurations. You use the Debug build configuration for debugging and the Release configuration for the final release distribution.

In the Debug configuration, a program compiles with full symbolic debug information and no optimization. Optimization complicates debugging, because the relationship between source code and generated instructions is more complex. The release configuration of a program has no symbolic debug information and is fully

optimized.

By default, Visual Studio uses the Debug build configuration, so you don't need to change it before debugging.

1. Start Visual Studio.
 2. Open the project that you created in [Create a .NET console application using Visual Studio](#).

The current build configuration is shown on the toolbar. The following toolbar image shows that Visual Studio is configured to compile the Debug version of the app:

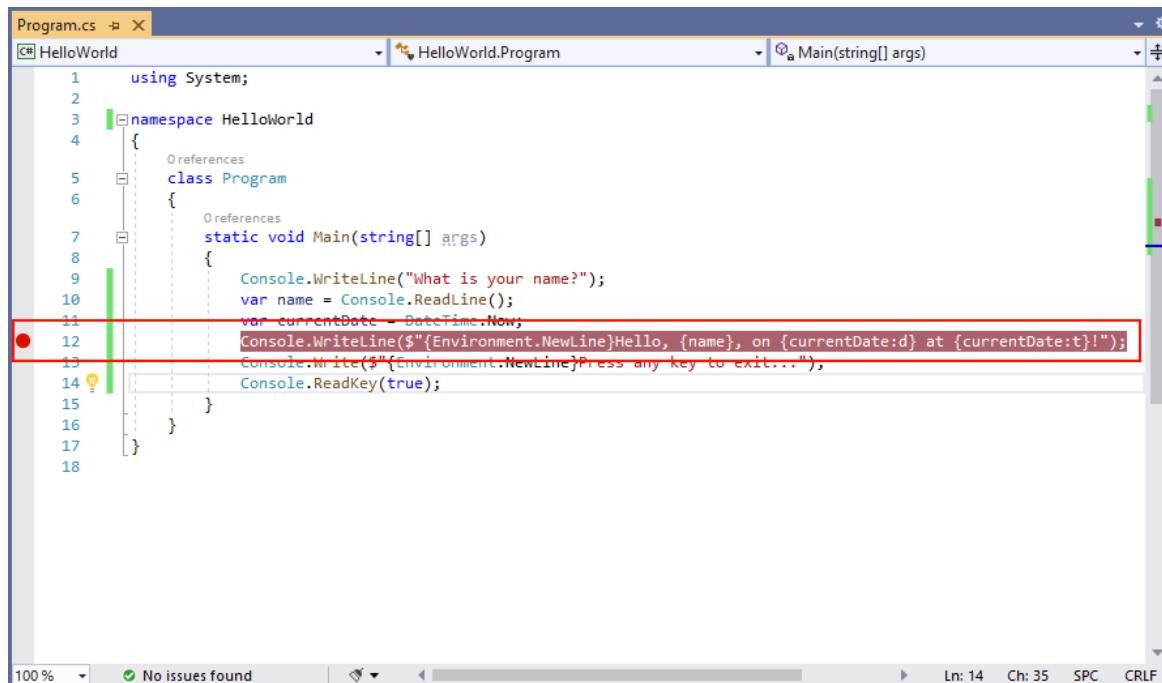


Set a breakpoint

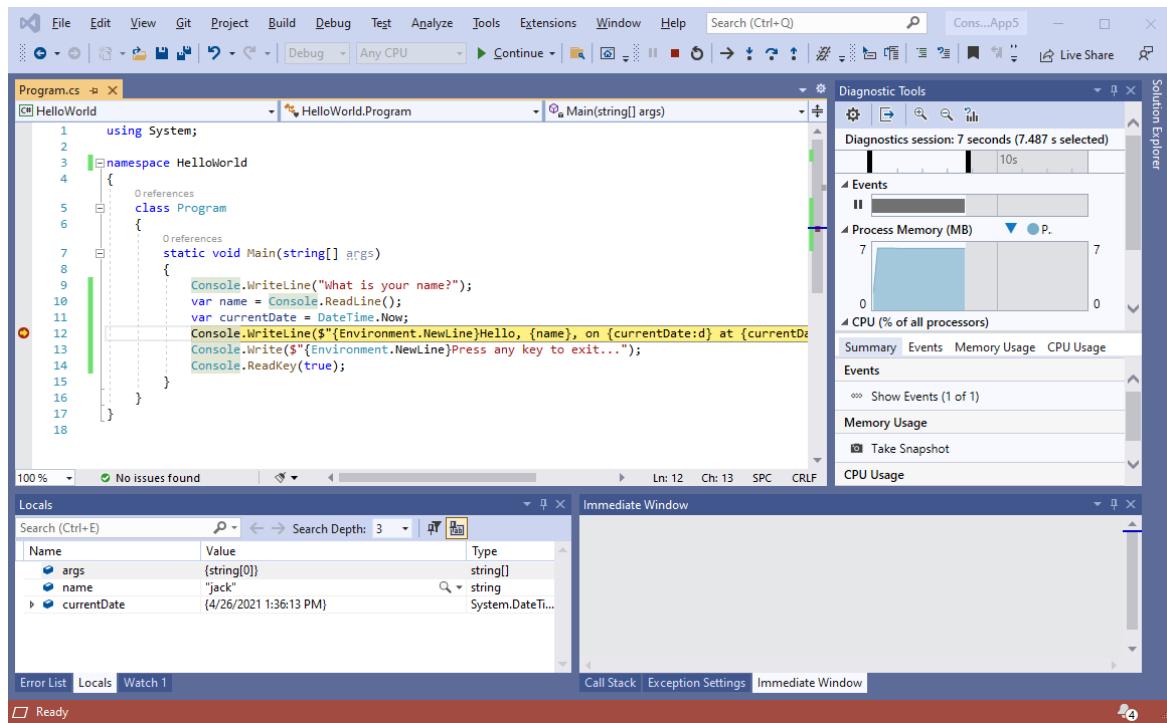
A *breakpoint* temporarily interrupts the execution of the application before the line with the breakpoint is executed.

1. Set a *breakpoint* on the line that displays the name, date, and time, by clicking in the left margin of the code window on that line. The left margin is to the left of the line numbers. Other ways to set a breakpoint are by placing the cursor in the line of code and then pressing F9 or choosing **Debug > Toggle Breakpoint** from the menu bar.

As the following image shows, Visual Studio indicates the line on which the breakpoint is set by highlighting it and displaying a red dot in the left margin.



2. Press F5 to run the program in Debug mode. Another way to start debugging is by choosing **Debug > Start Debugging** from the menu.
 3. Enter a string in the console window when the program prompts for a name, and then press Enter.
 4. Program execution stops when it reaches the breakpoint and before the `Console.WriteLine` method executes. The **Locals** window displays the values of variables that are defined in the currently executing method.

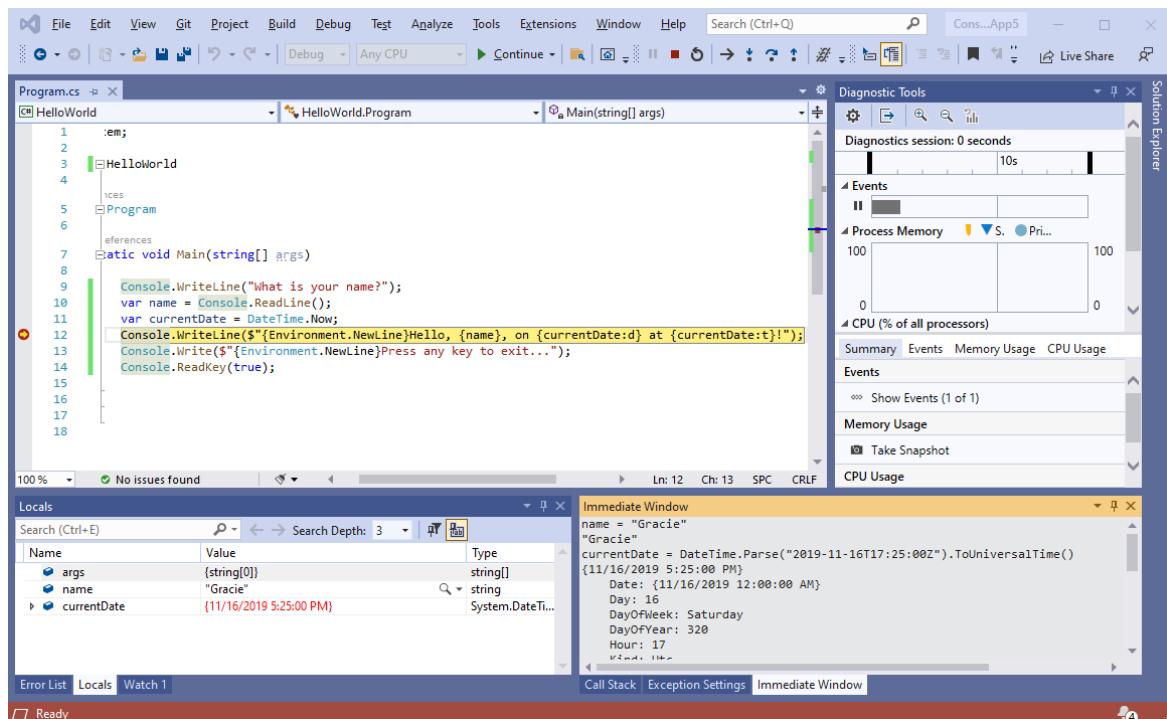


Use the Immediate window

The **Immediate** window lets you interact with the application you're debugging. You can interactively change the value of variables to see how it affects your program.

1. If the **Immediate** window is not visible, display it by choosing **Debug > Windows > Immediate**.
2. Enter `name = "Gracie"` in the **Immediate** window and press the **Enter** key.
3. Enter `currentDate = DateTime.Parse("2019-11-16T17:25:00Z").ToUniversalTime()` in the **Immediate** window and press the **Enter** key.

The **Immediate** window displays the value of the string variable and the properties of the **DateTime** value. In addition, the values of the variables are updated in the **Locals** window.



4. Press **F5** to continue program execution. Another way to continue is by choosing **Debug > Continue**

from the menu.

The values displayed in the console window correspond to the changes you made in the **Immediate** window.

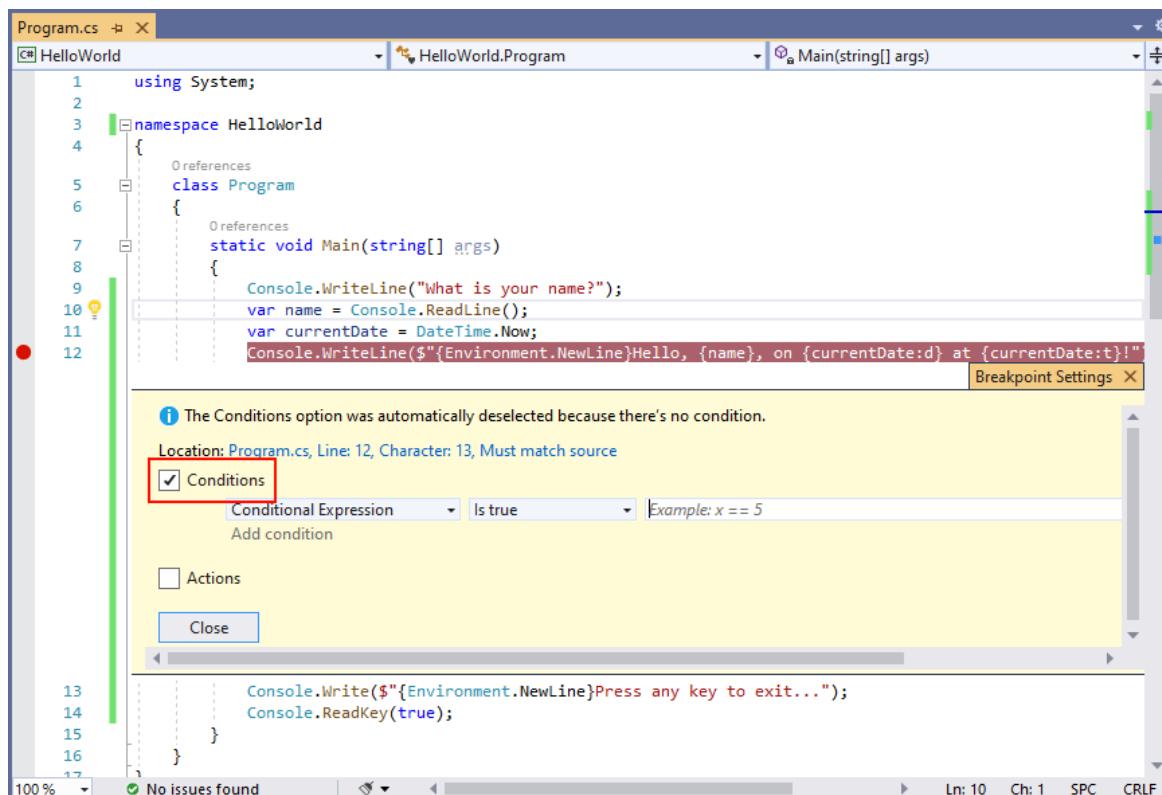
```
What is your name?  
jack  
  
Hello, Gracie, on 11/16/2019 at 5:25 PM!  
  
Press any key to exit...
```

5. Press any key to exit the application and stop debugging.

Set a conditional breakpoint

The program displays the string that the user enters. What happens if the user doesn't enter anything? You can test this with a useful debugging feature called a *conditional breakpoint*.

1. Right-click on the red dot that represents the breakpoint. In the context menu, select **Conditions** to open the **Breakpoint Settings** dialog. Select the box for **Conditions** if it's not already selected.



2. For the **Conditional Expression**, enter the following code in the field that shows example code that tests if `x` is 5. If the language you want to use is not shown, change the language selector at the top of the page.

```
String.IsNullOrEmpty(name)
```

```
String.IsNullOrEmpty(name)
```

Each time the breakpoint is hit, the debugger calls the `String.IsNullOrEmpty(name)` method, and it breaks on this line only if the method call returns `true`.

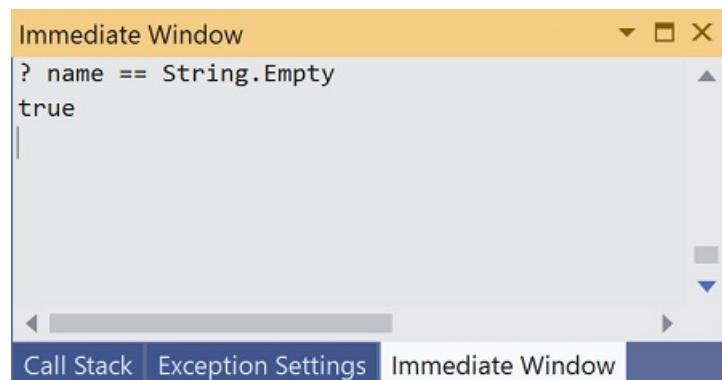
Instead of a conditional expression, you can specify a *hit count*, which interrupts program execution before a statement is executed a specified number of times. Another option is to specify a *filter condition*, which interrupts program execution based on such attributes as a thread identifier, process name, or thread name.

3. Select **Close** to close the dialog.
4. Start the program with debugging by pressing F5.
5. In the console window, press the Enter key when prompted to enter your name.
6. Because the condition you specified (`name` is either `null` or `String.Empty`) has been satisfied, program execution stops when it reaches the breakpoint and before the `Console.WriteLine` method executes.
7. Select the **Locals** window, which shows the values of variables that are local to the currently executing method. In this case, `Main` is the currently executing method. Observe that the value of the `name` variable is `""`, or `String.Empty`.
8. Confirm the value is an empty string by entering the following statement in the **Immediate** window and pressing Enter. The result is `true`.

```
? name == String.Empty
```

```
? String.IsNullOrEmpty(name)
```

The question mark directs the immediate window to [evaluate an expression](#).



9. Press F5 to continue program execution.
10. Press any key to close the console window and stop debugging.
11. Clear the breakpoint by clicking on the dot in the left margin of the code window. Other ways to clear a breakpoint are by pressing F9 or choosing **Debug > Toggle Breakpoint** while the line of code is selected.

Step through a program

Visual Studio also allows you to step line by line through a program and monitor its execution. Ordinarily, you'd set a breakpoint and follow program flow through a small part of your program code. Since this program is small, you can step through the entire program.

1. Choose **Debug > Step Into**. Another way to debug one statement at a time is by pressing F11.

Visual Studio highlights and displays an arrow beside the next line of execution.

```
Program.cs # X
HelloWorld
    using System;
    namespace HelloWorld
    {
        class Program
        {
            static void Main(string[] args)
            {
                Console.WriteLine("What is your name?");
                var name = Console.ReadLine();
                var currentDate = DateTime.Now;
                Console.WriteLine($"{Environment.NewLine}Hello, {name}, on {currentDate:d} at {currentDate:t}");
                Console.Write($"{Environment.NewLine}Press any key to exit...");
                Console.ReadKey(true);
            }
        }
    }
```

No issues found

Visual Basic

```
Program.vb # X
VB HelloWorld
    Imports System
    Module Program
        Sub Main(args As String())
            Console.WriteLine("What is your name?")
            Dim name = Console.ReadLine()
            Dim currentDate = DateTime.Now
            Console.WriteLine($"{Environment.NewLine}Hello, {name}, on {currentDate:d} at {currentDate:t}");
            Console.Write($"{Environment.NewLine}Press any key to exit...")
            Console.ReadKey(True)
        End Sub
    End Module
```

No issues found

At this point, the **Locals** window shows that the `args` array is empty, and `name` and `currentDate` have default values. In addition, Visual Studio has opened a blank console window.

2. Press F11. Visual Studio now highlights the next line of execution. The **Locals** window is unchanged, and the console window remains blank.

C#

```
Program.cs # X
HelloWorld
    using System;
    namespace HelloWorld
    {
        class Program
        {
            static void Main(string[] args)
            {
                Console.WriteLine("What is your name?"); ≤1ms elapsed
                var name = Console.ReadLine();
                var currentDate = DateTime.Now;
                Console.WriteLine($"{Environment.NewLine}Hello, {name}, on {currentDate:d} at {currentDate:t}");
                Console.Write($"{Environment.NewLine}Press any key to exit...");
                Console.ReadKey(true);
            }
        }
    }
```

No issues found

Visual Basic

```
Program.vb # X
VB HelloWorld      Program      Main
1 Imports System
2
3 Module Program
4 Sub Main(args As String())
5     Console.WriteLine("What is your name?")
6     Dim name = Console.ReadLine()
7     Dim currentDate = DateTime.Now
8     Console.WriteLine($"{Environment.NewLine}Hello, {name}, on {currentDate:d} at {currentDate:t}")
9     Console.WriteLine($"{Environment.NewLine}Press any key to exit...")
10    Console.ReadKey(True)
11 End Sub
12 End Module
13
```

No issues found

3. Press F11. Visual Studio highlights the statement that includes the `name` variable assignment. The **Locals** window shows that `name` is `null`, and the console window displays the string "What is your name?".
4. Respond to the prompt by entering a string in the console window and pressing Enter. The console is unresponsive, and the string you entered isn't displayed in the console window, but the `Console.ReadLine` method will nevertheless capture your input.
5. Press F11. Visual Studio highlights the statement that includes the `currentDate` variable assignment. The **Locals** window shows the value returned by the call to the `Console.ReadLine` method. The console window also displays the string you entered at the prompt.
6. Press F11. The **Locals** window shows the value of the `currentDate` variable after the assignment from the `DateTime.Now` property. The console window is unchanged.
7. Press F11. Visual Studio calls the `Console.WriteLine(String, Object, Object)` method. The console window displays the formatted string.
8. Choose **Debug > Step Out**. Another way to stop step-by-step execution is by pressing Shift+F11.
The console window displays a message and waits for you to press a key.

9. Press any key to close the console window and stop debugging.

Use Release build configuration

Once you've tested the Debug version of your application, you should also compile and test the Release version. The Release version incorporates compiler optimizations that can sometimes negatively affect the behavior of an application. For example, compiler optimizations that are designed to improve performance can create race conditions in multithreaded applications.

To build and test the Release version of your console application, change the build configuration on the toolbar from **Debug** to **Release**.



When you press F5 or choose **Build Solution** from the **Build** menu, Visual Studio compiles the Release version of the application. You can test it as you did the Debug version.

Next steps

In this tutorial, you used Visual Studio debugging tools. In the next tutorial, you publish a deployable version of the app.

Publish a .NET console application using Visual Studio

This tutorial is only available for .NET 5 and .NET 6. Select one of those options at the top of the page.

Tutorial: Publish a .NET console application using Visual Studio

9/20/2022 • 5 minutes to read • [Edit Online](#)

This tutorial shows how to publish a console app so that other users can run it. Publishing creates the set of files that are needed to run your application. To deploy the files, copy them to the target machine.

Prerequisites

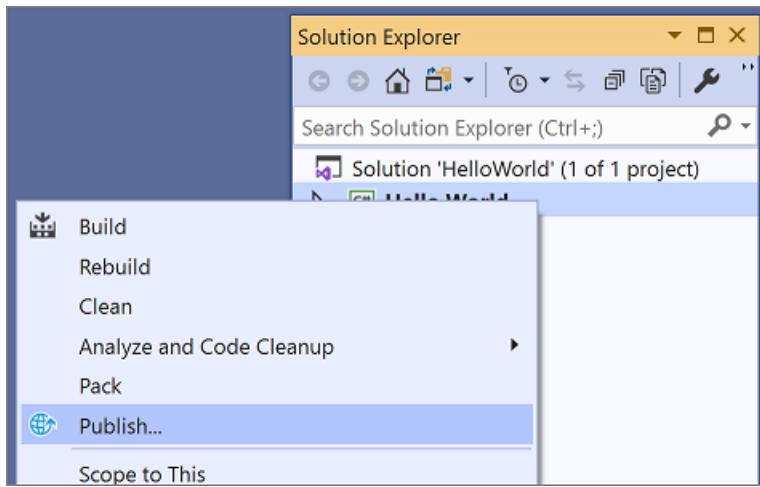
- This tutorial works with the console app that you create in [Create a .NET console application using Visual Studio](#).

Publish the app

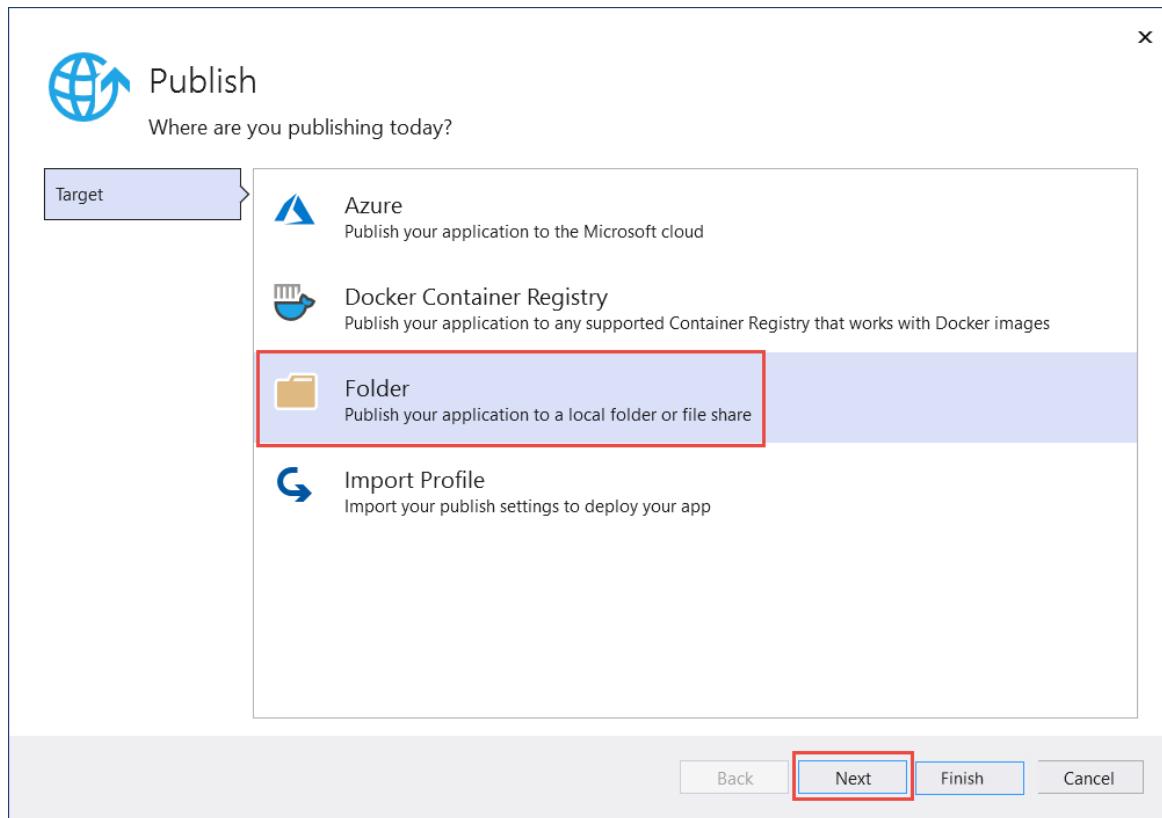
1. Start Visual Studio.
2. Open the *HelloWorld* project that you created in [Create a .NET console application using Visual Studio](#).
3. Make sure that Visual Studio is using the Release build configuration. If necessary, change the build configuration setting on the toolbar from **Debug** to **Release**.



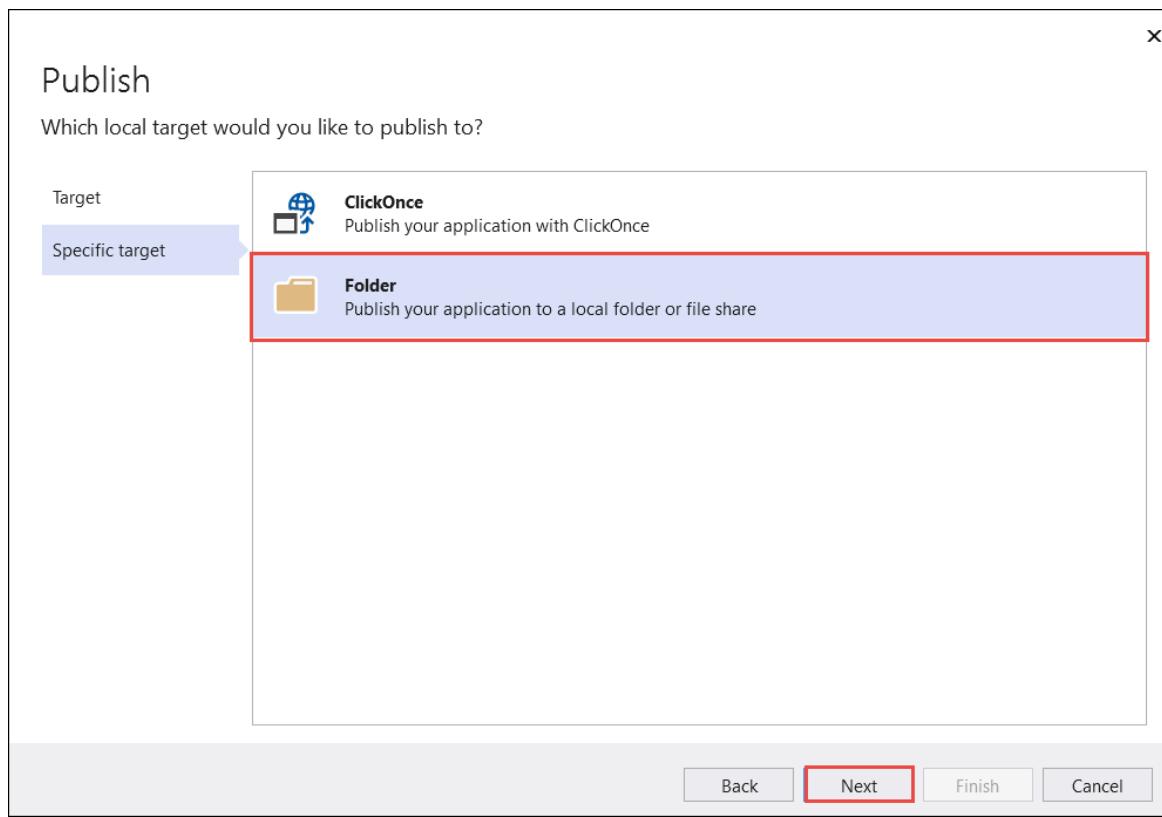
4. Right-click on the **HelloWorld** project (not the HelloWorld solution) and select **Publish** from the menu.



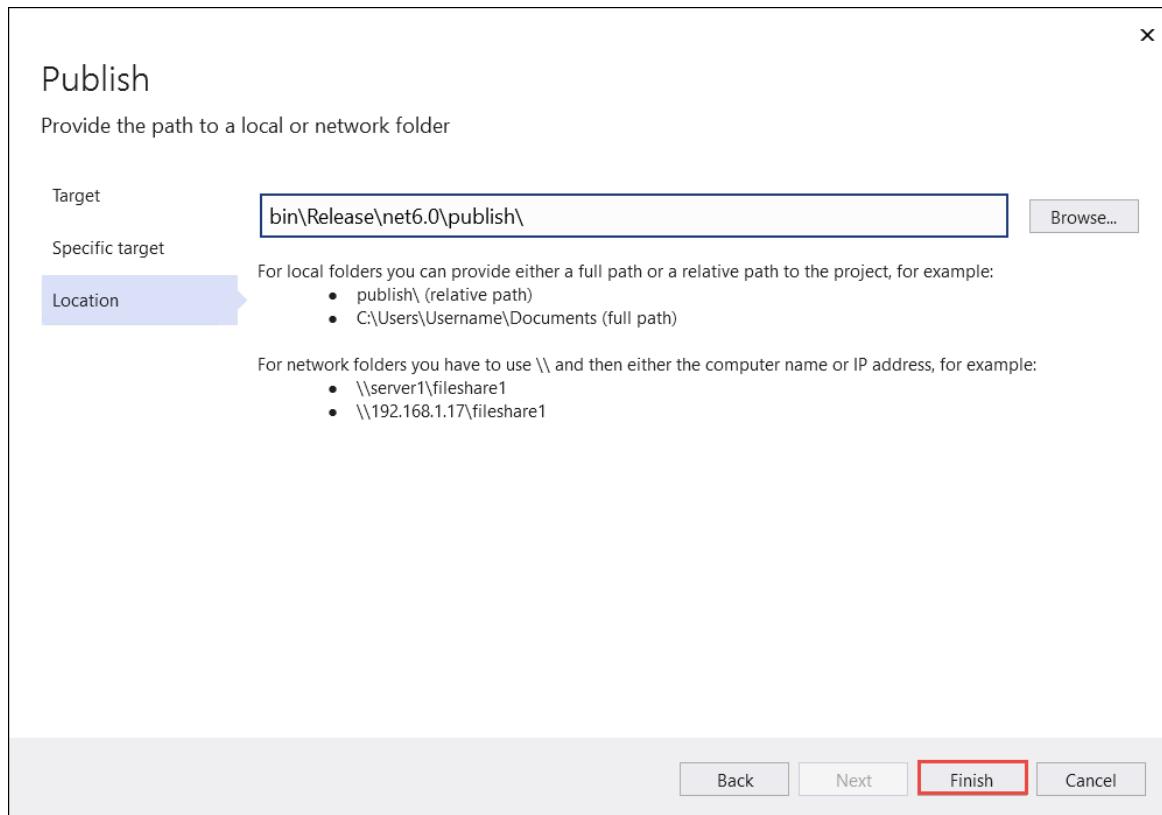
5. On the Target tab of the Publish page, select **Folder**, and then select **Next**.



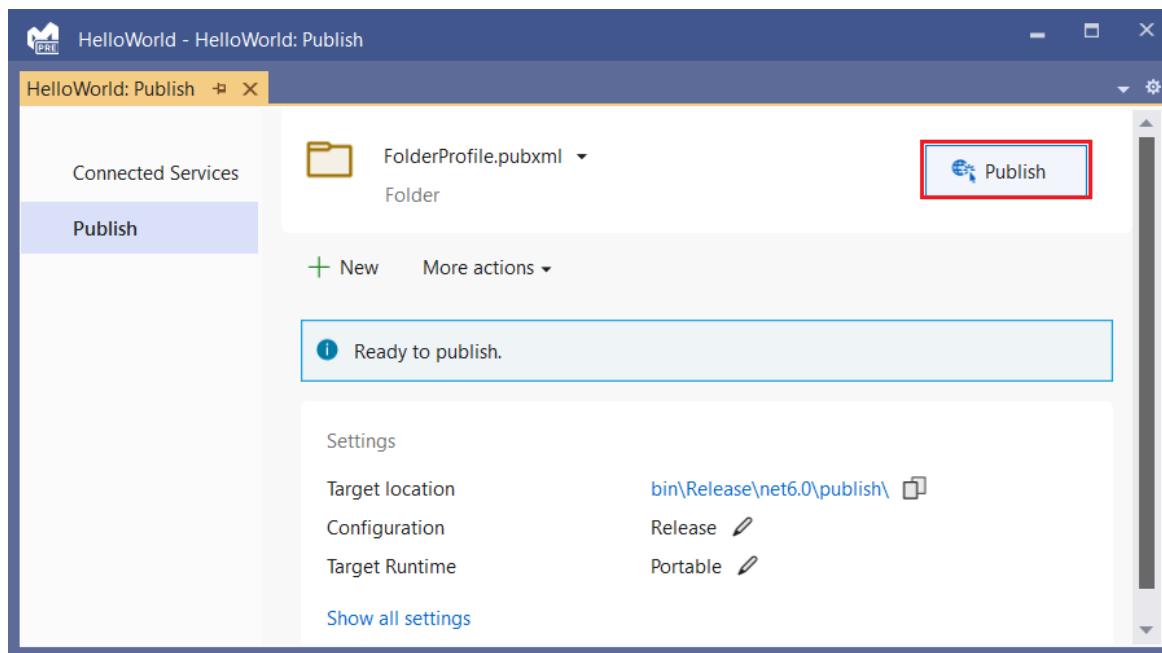
6. On the **Specific Target** tab of the **Publish** page, select **Folder**, and then select **Next**.



7. On the **Location** tab of the **Publish** page, select **Finish**.



8. On the **Publish** tab of the **Publish** window, select **Publish**.

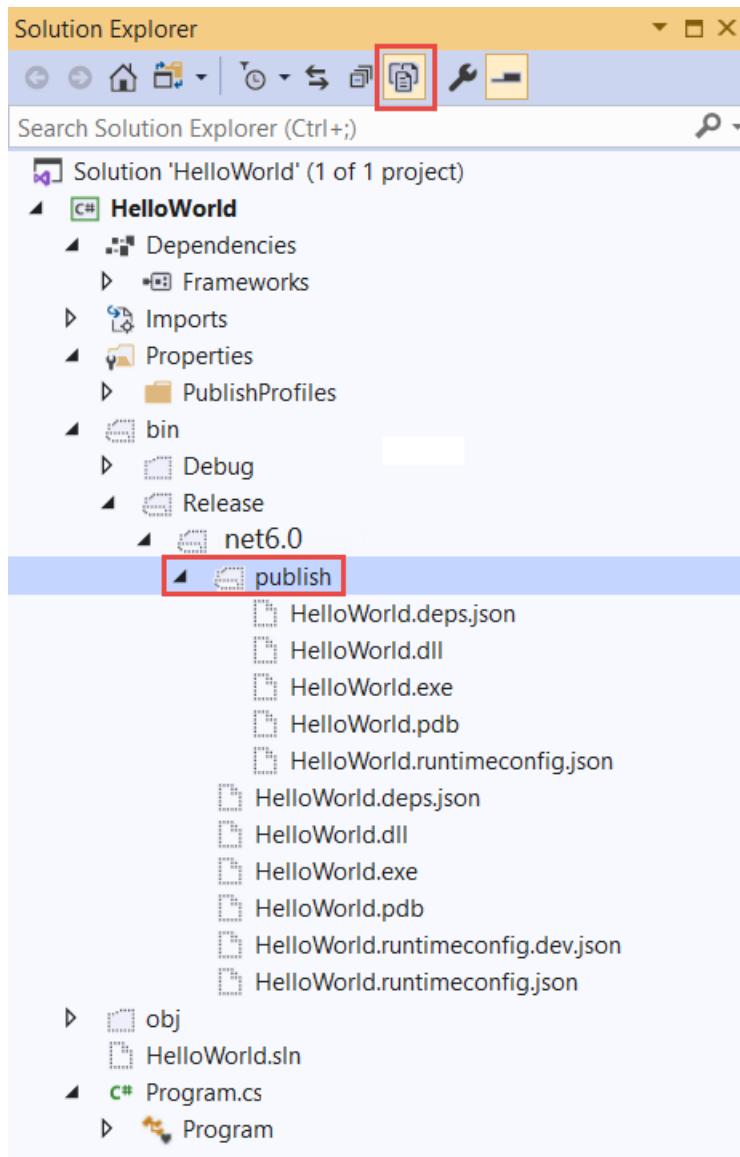


Inspect the files

By default, the publishing process creates a framework-dependent deployment, which is a type of deployment where the published application runs on machine that has the .NET runtime installed. Users can run the published app by double-clicking the executable or issuing the `dotnet HelloWorld.dll` command from a command prompt.

In the following steps, you'll look at the files created by the publish process.

1. In Solution Explorer, select **Show all files**.
2. In the project folder, expand *bin/Release/net6.0/publish*.



As the image shows, the published output includes the following files:

- *HelloWorld.deps.json*

This is the application's runtime dependencies file. It defines the .NET components and the libraries (including the dynamic link library that contains your application) needed to run the app. For more information, see [Runtime configuration files](#).

- *HelloWorld.dll*

This is the [framework-dependent deployment](#) version of the application. To execute this dynamic link library, enter `dotnet HelloWorld.dll` at a command prompt. This method of running the app works on any platform that has the .NET runtime installed.

- *HelloWorld.exe*

This is the [framework-dependent executable](#) version of the application. To run it, enter `HelloWorld.exe` at a command prompt. The file is operating-system-specific.

- *HelloWorld.pdb* (optional for deployment)

This is the debug symbols file. You aren't required to deploy this file along with your application, although you should save it in the event that you need to debug the published version of your application.

- *HelloWorld.runtimeconfig.json*

This is the application's runtime configuration file. It identifies the version of .NET that your application was built to run on. You can also add configuration options to it. For more information, see [.NET runtime configuration settings](#).

Run the published app

1. In **Solution Explorer**, right-click the *publish* folder, and select **Copy Full Path**.
2. Open a command prompt and navigate to the *publish* folder. To do that, enter `cd` and then paste the full path. For example:

```
cd C:\Projects\HelloWorld\bin\Release\net6.0\publish\
```
3. Run the app by using the executable:
 - a. Enter `HelloWorld.exe` and press **Enter**.
 - b. Enter a name in response to the prompt, and press any key to exit.
4. Run the app by using the `dotnet` command:
 - a. Enter `dotnet HelloWorld.dll` and press **Enter**.
 - b. Enter a name in response to the prompt, and press any key to exit.

Additional resources

- [.NET application deployment](#)

Next steps

In this tutorial, you published a console app. In the next tutorial, you create a class library.

[Create a .NET class library using Visual Studio](#)

This tutorial shows how to publish a console app so that other users can run it. Publishing creates the set of files that are needed to run your application. To deploy the files, copy them to the target machine.

Prerequisites

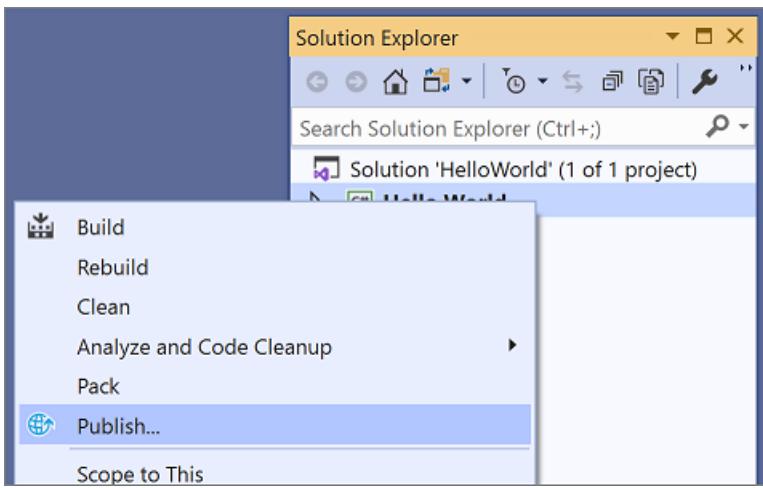
- This tutorial works with the console app that you create in [Create a .NET console application using Visual Studio](#).

Publish the app

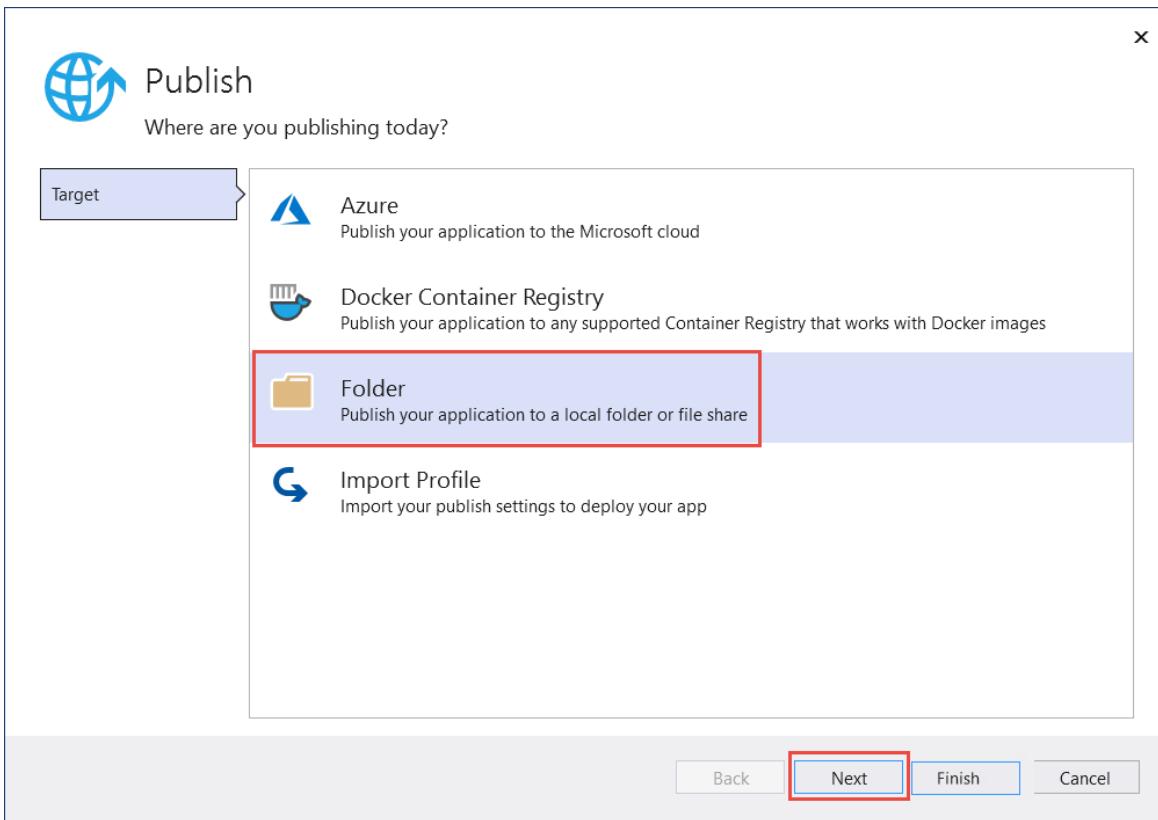
1. Start Visual Studio.
2. Open the *HelloWorld* project that you created in [Create a .NET console application using Visual Studio](#).
3. Make sure that Visual Studio is using the Release build configuration. If necessary, change the build configuration setting on the toolbar from **Debug** to **Release**.



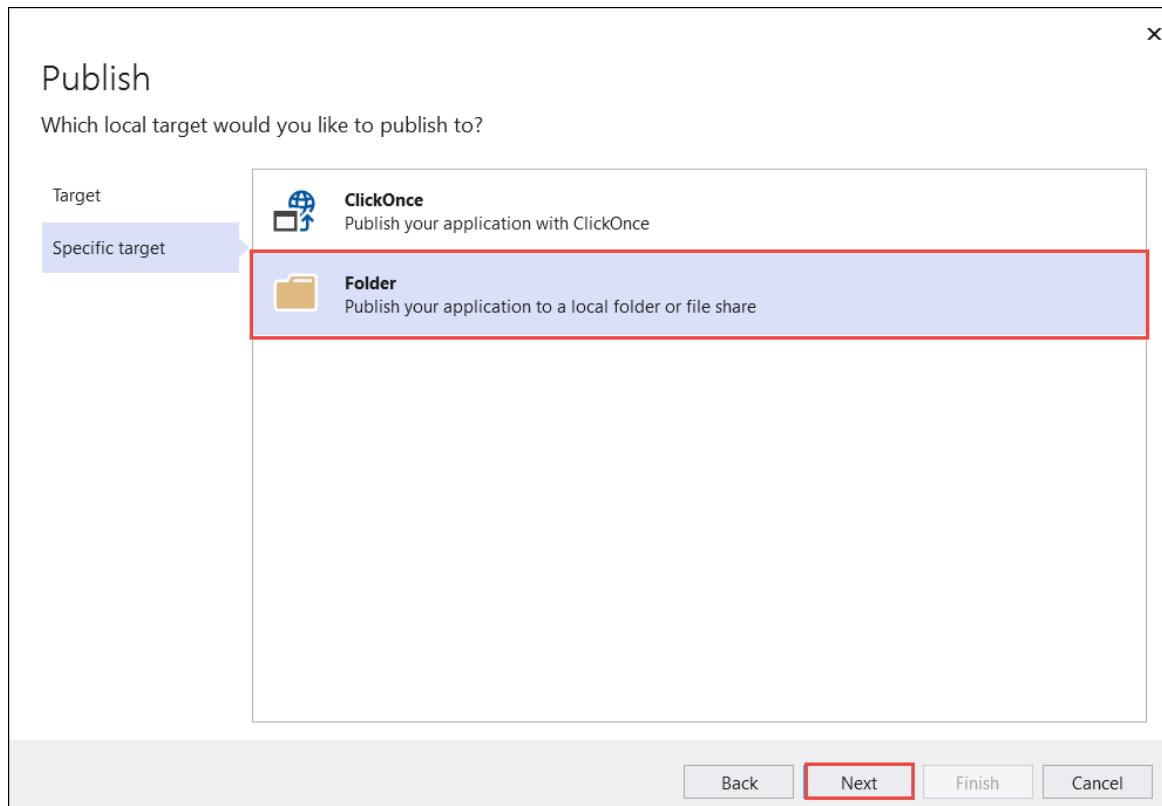
4. Right-click on the **HelloWorld** project (not the *HelloWorld* solution) and select **Publish** from the menu.



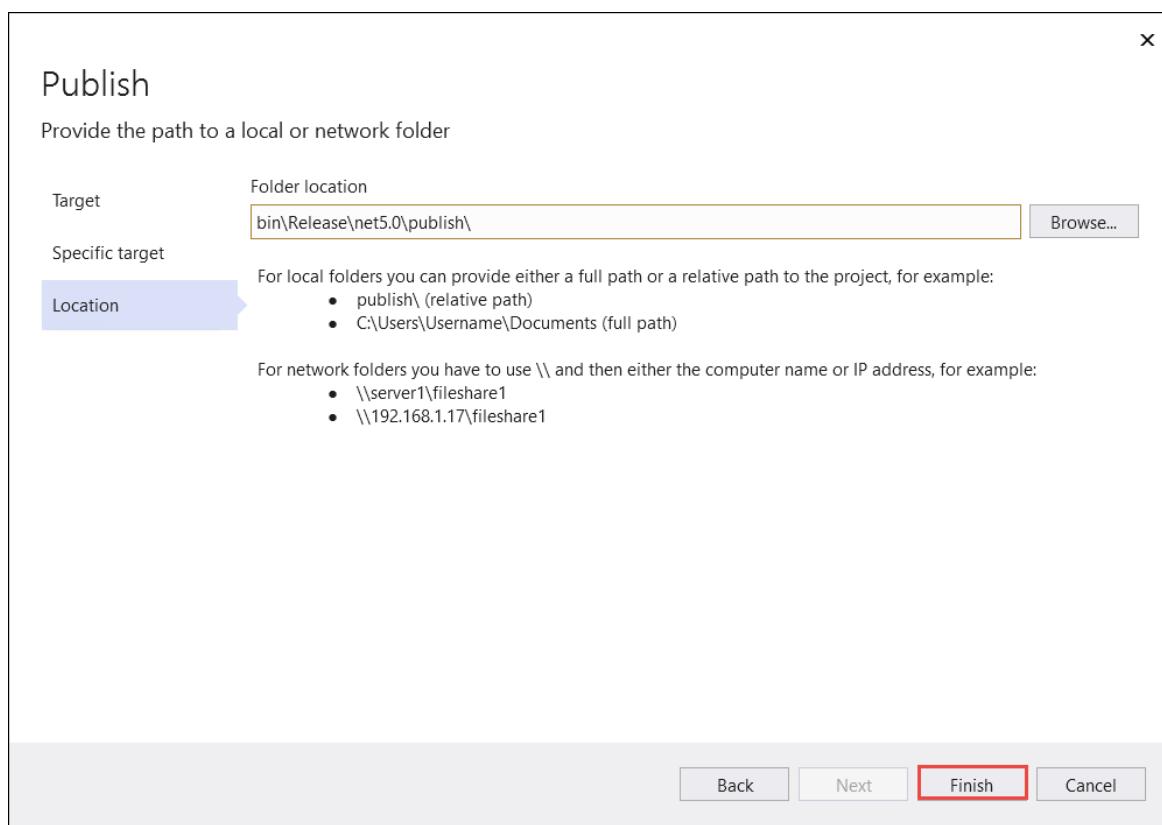
5. On the **Target** tab of the **Publish** page, select **Folder**, and then select **Next**.



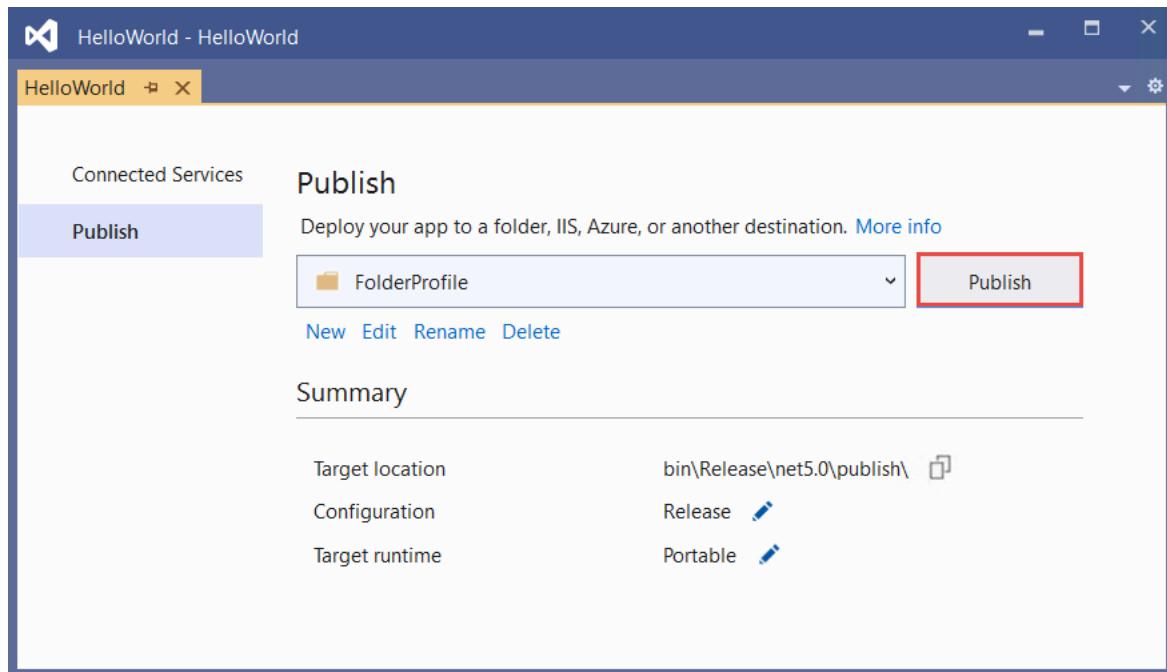
6. On the **Specific Target** tab of the **Publish** page, select **Folder**, and then select **Next**.



7. On the **Location** tab of the **Publish** page, select **Finish**.



8. On the **Publish** tab of the **Publish** window, select **Publish**.

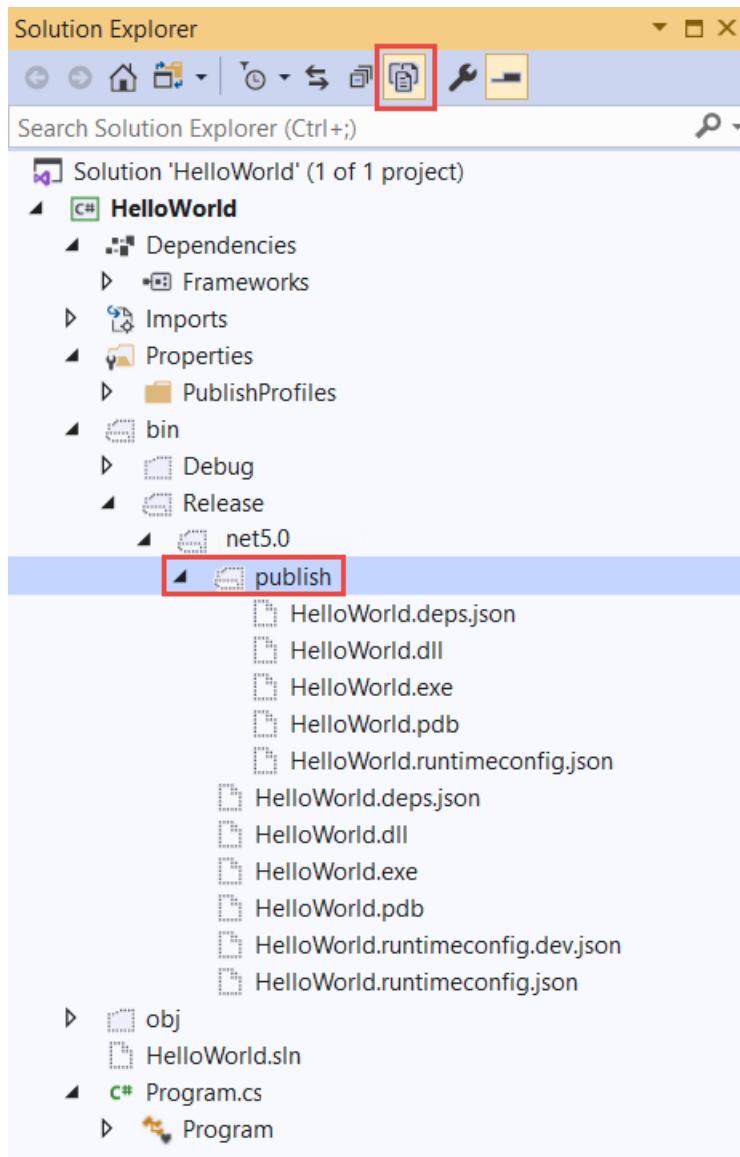


Inspect the files

By default, the publishing process creates a framework-dependent deployment, which is a type of deployment where the published application runs on machine that has the .NET runtime installed. Users can run the published app by double-clicking the executable or issuing the `dotnet HelloWorld.dll` command from a command prompt.

In the following steps, you'll look at the files created by the publish process.

1. In **Solution Explorer**, select **Show all files**.
2. In the project folder, expand *bin/Release/net5.0/publish*.



As the image shows, the published output includes the following files:

- *HelloWorld.deps.json*

This is the application's runtime dependencies file. It defines the .NET components and the libraries (including the dynamic link library that contains your application) needed to run the app. For more information, see [Runtime configuration files](#).

- *HelloWorld.dll*

This is the [framework-dependent deployment](#) version of the application. To execute this dynamic link library, enter `dotnet HelloWorld.dll` at a command prompt. This method of running the app works on any platform that has the .NET runtime installed.

- *HelloWorld.exe*

This is the [framework-dependent executable](#) version of the application. To run it, enter `HelloWorld.exe` at a command prompt. The file is operating-system-specific.

- *HelloWorld.pdb* (optional for deployment)

This is the debug symbols file. You aren't required to deploy this file along with your application, although you should save it in the event that you need to debug the published version of your application.

- *HelloWorld.runtimeconfig.json*

This is the application's runtime configuration file. It identifies the version of .NET that your application was built to run on. You can also add configuration options to it. For more information, see [.NET runtime configuration settings](#).

Run the published app

1. In **Solution Explorer**, right-click the *publish* folder, and select **Copy Full Path**.
2. Open a command prompt and navigate to the *publish* folder. To do that, enter `cd` and then paste the full path. For example:

```
cd C:\Projects\HelloWorld\bin\Release\net5.0\publish\
```

3. Run the app by using the executable:
 - a. Enter `HelloWorld.exe` and press **Enter**.
 - b. Enter a name in response to the prompt, and press any key to exit.
4. Run the app by using the `dotnet` command:
 - a. Enter `dotnet HelloWorld.dll` and press **Enter**.
 - b. Enter a name in response to the prompt, and press any key to exit.

Additional resources

- [.NET application deployment](#)

Next steps

In this tutorial, you published a console app. In the next tutorial, you create a class library.

[Create a .NET class library using Visual Studio](#)

This tutorial is only available for .NET 5 and .NET 6. Select one of those options at the top of the page.

Tutorial: Create a .NET class library using Visual Studio

9/20/2022 • 12 minutes to read • [Edit Online](#)

In this tutorial, you create a simple class library that contains a single string-handling method.

A *class library* defines types and methods that are called by an application. If the library targets .NET Standard 2.0, it can be called by any .NET implementation (including .NET Framework) that supports .NET Standard 2.0. If the library targets .NET 6, it can be called by any application that targets .NET 6. This tutorial shows how to target .NET 6.

When you create a class library, you can distribute it as a NuGet package or as a component bundled with the application that uses it.

Prerequisites

- [Visual Studio 2022 version 17.0.0 Preview](#) with the **.NET desktop development** workload installed. The .NET 6 SDK is automatically installed when you select this workload.

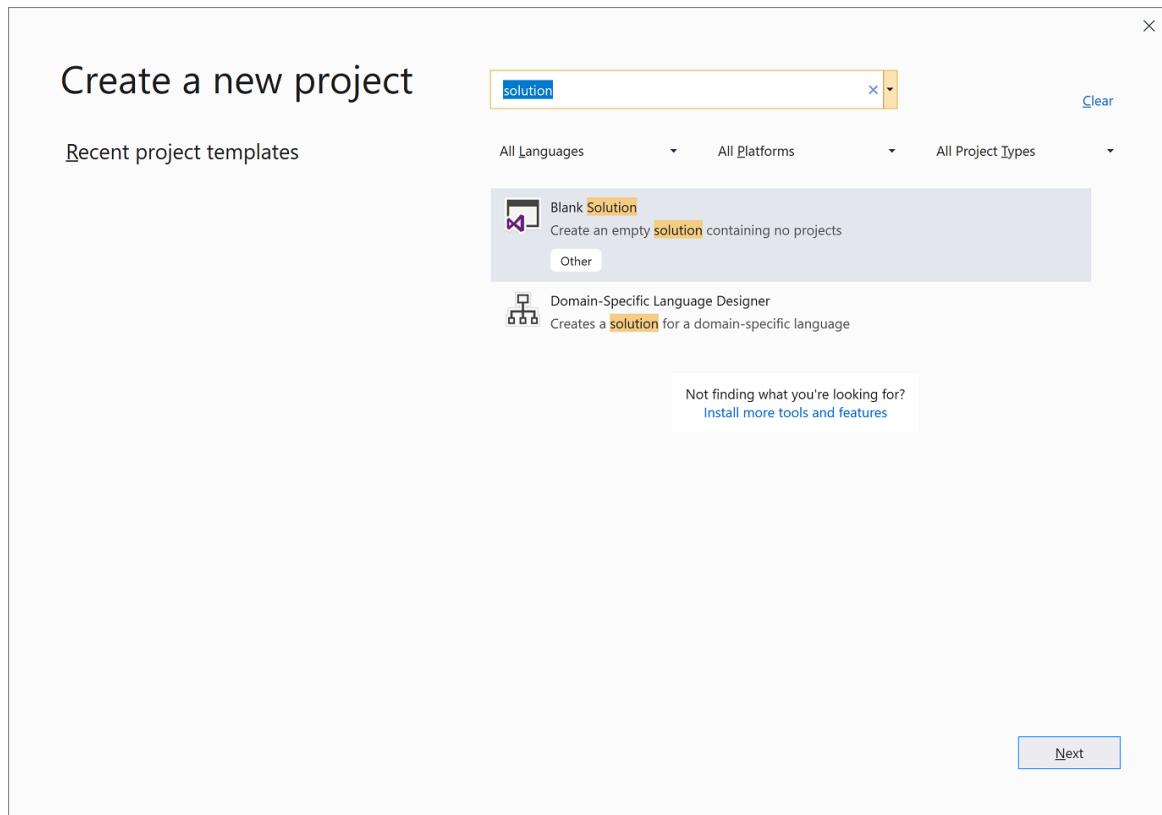
For more information, see [Install the .NET SDK with Visual Studio](#).

Create a solution

Start by creating a blank solution to put the class library project in. A Visual Studio solution serves as a container for one or more projects. You'll add additional, related projects to the same solution.

To create the blank solution:

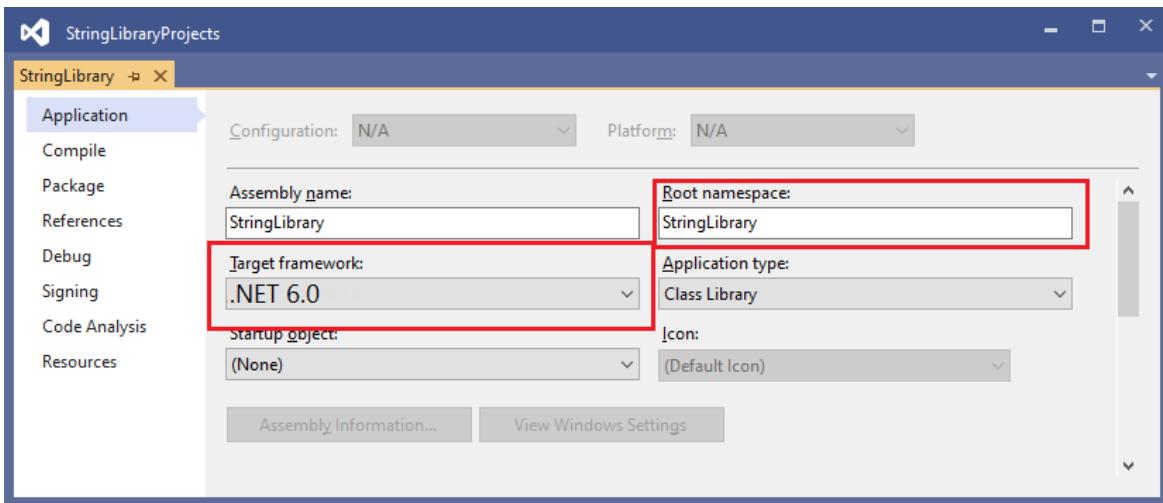
1. Start Visual Studio.
2. On the start window, choose **Create a new project**.
3. On the **Create a new project** page, enter **solution** in the search box. Choose the **Blank Solution** template, and then choose **Next**.



4. On the **Configure your new project** page, enter **ClassLibraryProjects** in the **Solution name** box. Then choose **Create**.

Create a class library project

1. Add a new .NET class library project named "StringLibrary" to the solution.
 - a. Right-click on the solution in **Solution Explorer** and select **Add > New Project**.
 - b. On the **Add a new project** page, enter **library** in the search box. Choose **C# or Visual Basic** from the **Language** list, and then choose **All platforms** from the **Platform** list. Choose the **Class Library** template, and then choose **Next**.
 - c. On the **Configure your new project** page, enter **StringLibrary** in the **Project name** box, and then choose **Next**.
 - d. On the **Additional information** page, select **.NET 6 (Long-term support)**, and then choose **Create**.
2. Check to make sure that the library targets the correct version of .NET. Right-click on the library project in **Solution Explorer**, and then select **Properties**. The **Target Framework** text box shows that the project targets .NET 6.0.
3. If you're using Visual Basic, clear the text in the **Root namespace** text box.



For each project, Visual Basic automatically creates a namespace that corresponds to the project name. In this tutorial, you define a top-level namespace by using the `namespace` keyword in the code file.

4. Replace the code in the code window for *Class1.cs* or *Class1.vb* with the following code, and save the file. If the language you want to use isn't shown, change the language selector at the top of the page.

```
namespace UtilityLibraries;

public static class StringLibrary
{
    public static bool StartsWithUpper(this string? str)
    {
        if (string.IsNullOrWhiteSpace(str))
            return false;

        char ch = str[0];
        return char.IsUpper(ch);
    }
}
```

```
Imports System.Runtime.CompilerServices

Namespace UtilityLibraries
    Public Module StringLibrary
        <Extension>
        Public Function StartsWithUpper(str As String) As Boolean
            If String.IsNullOrWhiteSpace(str) Then
                Return False
            End If

            Dim ch As Char = str(0)
            Return Char.IsUpper(ch)
        End Function
    End Module
End Namespace
```

The class library, `UtilityLibraries.StringLibrary`, contains a method named `StartsWithUpper`. This method returns a `Boolean` value that indicates whether the current string instance begins with an uppercase character. The Unicode standard distinguishes uppercase characters from lowercase characters. The `Char.IsUpper(Char)` method returns `true` if a character is uppercase.

`StartsWithUpper` is implemented as an `extension method` so that you can call it as if it were a member of the `String` class. The question mark (?) after `string` in the C# code indicates that the string may be null.

5. On the menu bar, select **Build > Build Solution** or press **Ctrl+Shift+B** to verify that the project

compiles without error.

Add a console app to the solution

Add a console application that uses the class library. The app will prompt the user to enter a string and report whether the string begins with an uppercase character.

1. Add a new .NET console application named "ShowCase" to the solution.
 - a. Right-click on the solution in **Solution Explorer** and select **Add > New project**.
 - b. On the **Add a new project** page, enter **console** in the search box. Choose **C# or Visual Basic** from the Language list, and then choose **All platforms** from the Platform list.
 - c. Choose the **Console Application** template, and then choose **Next**.
 - d. On the **Configure your new project** page, enter **ShowCase** in the **Project name** box. Then choose **Next**.
 - e. On the **Additional information** page, select **.NET 6 (Long-term support)** in the **Framework** box. Then choose **Create**.

2. In the code window for the *Program.cs* or *Program.vb* file, replace all of the code with the following code.

```
using UtilityLibraries;

class Program
{
    static void Main(string[] args)
    {
        int row = 0;

        do
        {
            if (row == 0 || row >= 25)
                ResetConsole();

            string? input = Console.ReadLine();
            if (string.IsNullOrEmpty(input)) break;
            Console.WriteLine($"Input: {input}");
            Console.WriteLine("Begins with uppercase? " +
                $"{(input.StartsWithUpper() ? "Yes" : "No")}");
            Console.WriteLine();
            row += 4;
        } while (true);
        return;

        // Declare a ResetConsole local method
        void ResetConsole()
        {
            if (row > 0)
            {
                Console.WriteLine("Press any key to continue...");
                Console.ReadKey();
            }
            Console.Clear();
            Console.WriteLine($"{Environment.NewLine}Press <Enter> only to exit; otherwise, enter a
string and press <Enter>:{Environment.NewLine}");
            row = 3;
        }
    }
}
```

```

Imports UtilityLibraries

Module Program
    Dim row As Integer = 0

    Sub Main()
        Do
            If row = 0 OrElse row >= 25 Then ResetConsole()

            Dim input As String = Console.ReadLine()
            If String.IsNullOrEmpty(input) Then Return

            Console.WriteLine($"Input: {input} {"Begins with uppercase? ",30}: " +
                $"{If(input.StartsWithUpper(), "Yes", "No")}{Environment.NewLine}")
            row += 3
        Loop While True
    End Sub

    Private Sub ResetConsole()
        If row > 0 Then
            Console.WriteLine("Press any key to continue...")
            Console.ReadKey()
        End If
        Console.Clear()
        Console.WriteLine($"{Environment.NewLine}Press <Enter> only to exit; otherwise, enter a
string and press <Enter>:{Environment.NewLine}")
        row = 3
    End Sub
End Module

```

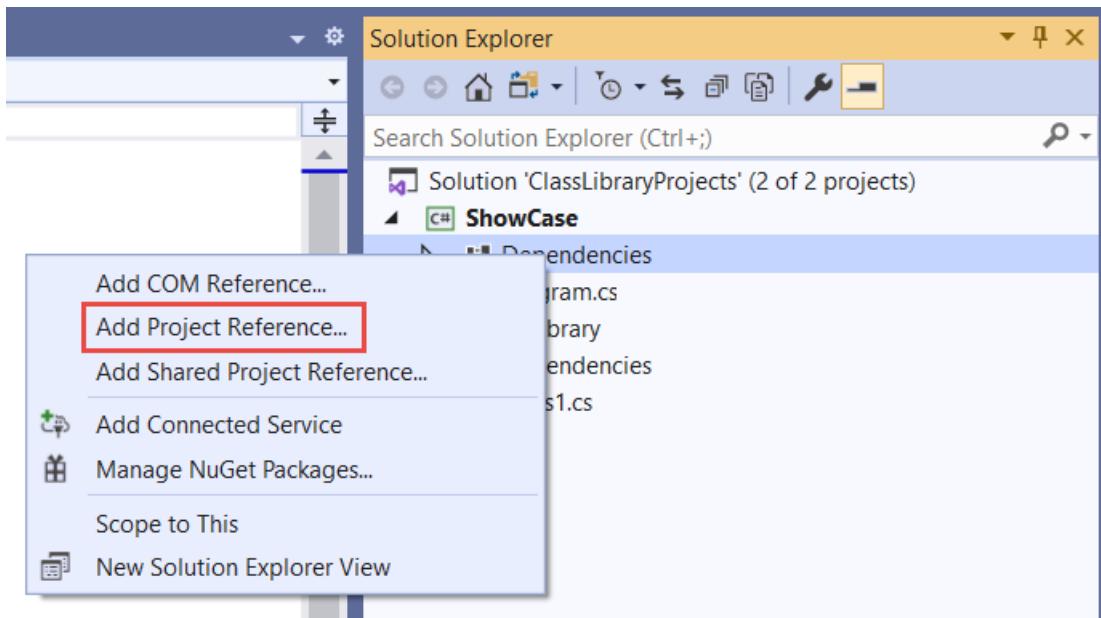
The code uses the `row` variable to maintain a count of the number of rows of data written to the console window. Whenever it's greater than or equal to 25, the code clears the console window and displays a message to the user.

The program prompts the user to enter a string. It indicates whether the string starts with an uppercase character. If the user presses the Enter key without entering a string, the application ends, and the console window closes.

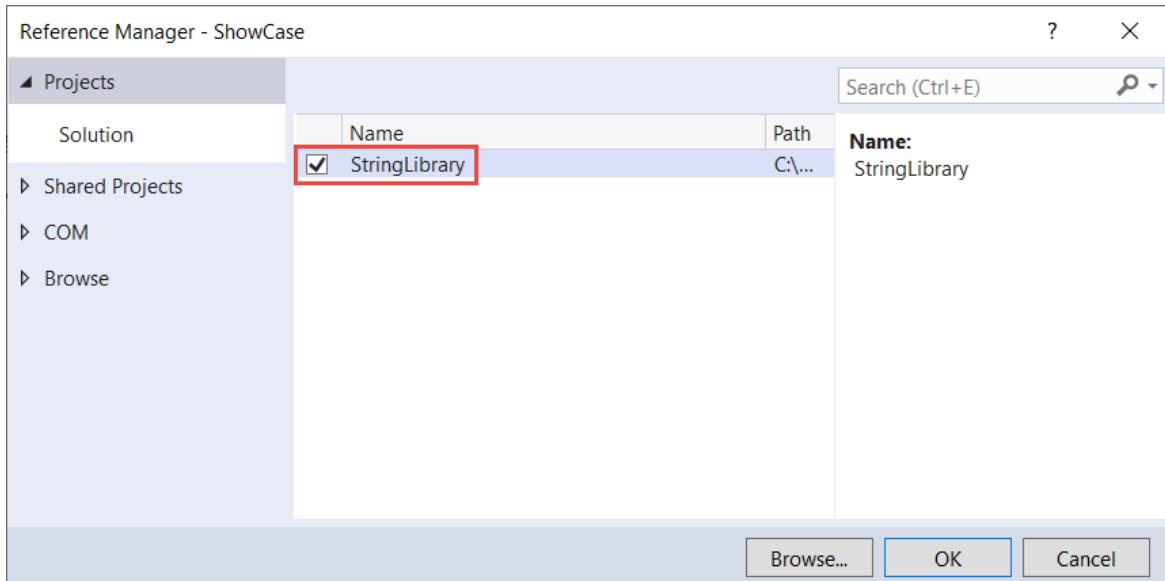
Add a project reference

Initially, the new console app project doesn't have access to the class library. To allow it to call methods in the class library, create a project reference to the class library project.

1. In **Solution Explorer**, right-click the `ShowCase` project's **Dependencies** node, and select **Add Project Reference**.

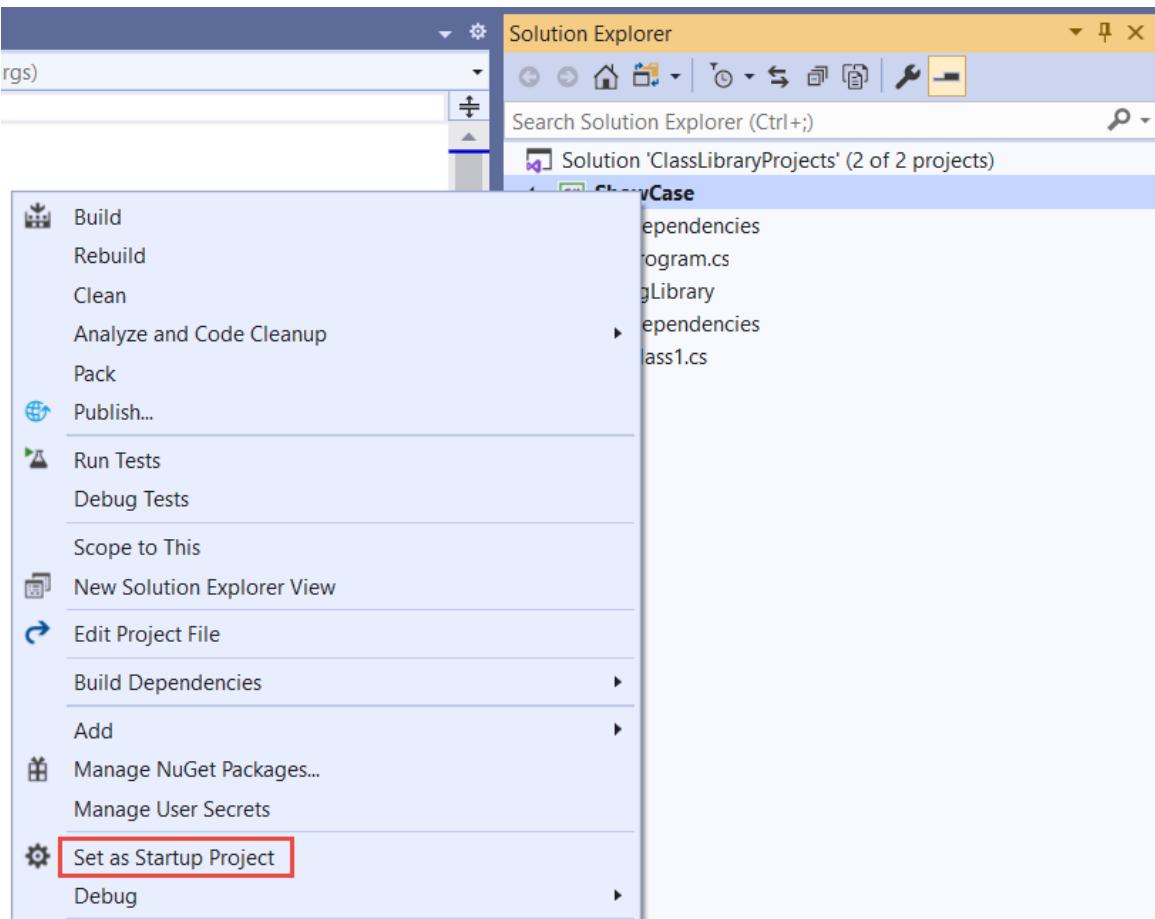


2. In the **Reference Manager** dialog, select the **StringLibrary** project, and select **OK**.

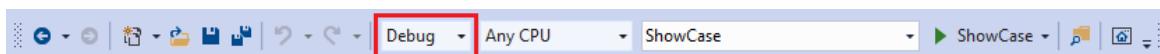


Run the app

1. In **Solution Explorer**, right-click the **ShowCase** project and select **Set as StartUp Project** in the context menu.



2. Press Ctrl+F5 to compile and run the program without debugging.



3. Try out the program by entering strings and pressing Enter, then press Enter to exit.

```
Press <Enter> only to exit; otherwise, enter a string and press <Enter>:  
  
Hello  
Input: Hello  
Begins with uppercase? Yes  
  
hello  
Input: hello  
Begins with uppercase? No
```

Additional resources

- [Develop libraries with the .NET CLI](#)
- [.NET Standard versions and the platforms they support](#).

Next steps

In this tutorial, you created a class library. In the next tutorial, you learn how to unit test the class library.

[Unit test a .NET class library using Visual Studio](#)

Or you can skip automated unit testing and learn how to share the library by creating a NuGet package:

[Create and publish a package using Visual Studio](#)

Or learn how to publish a console app. If you publish the console app from the solution you created in this tutorial, the class library goes with it as a `.dll` file.

Publish a .NET console application using Visual Studio

In this tutorial, you create a simple class library that contains a single string-handling method.

A *class library* defines types and methods that are called by an application. If the library targets .NET Standard 2.0, it can be called by any .NET implementation (including .NET Framework) that supports .NET Standard 2.0. If the library targets .NET 5, it can be called by any application that targets .NET 5. This tutorial shows how to target .NET 5.

When you create a class library, you can distribute it as a NuGet package or as a component bundled with the application that uses it.

Prerequisites

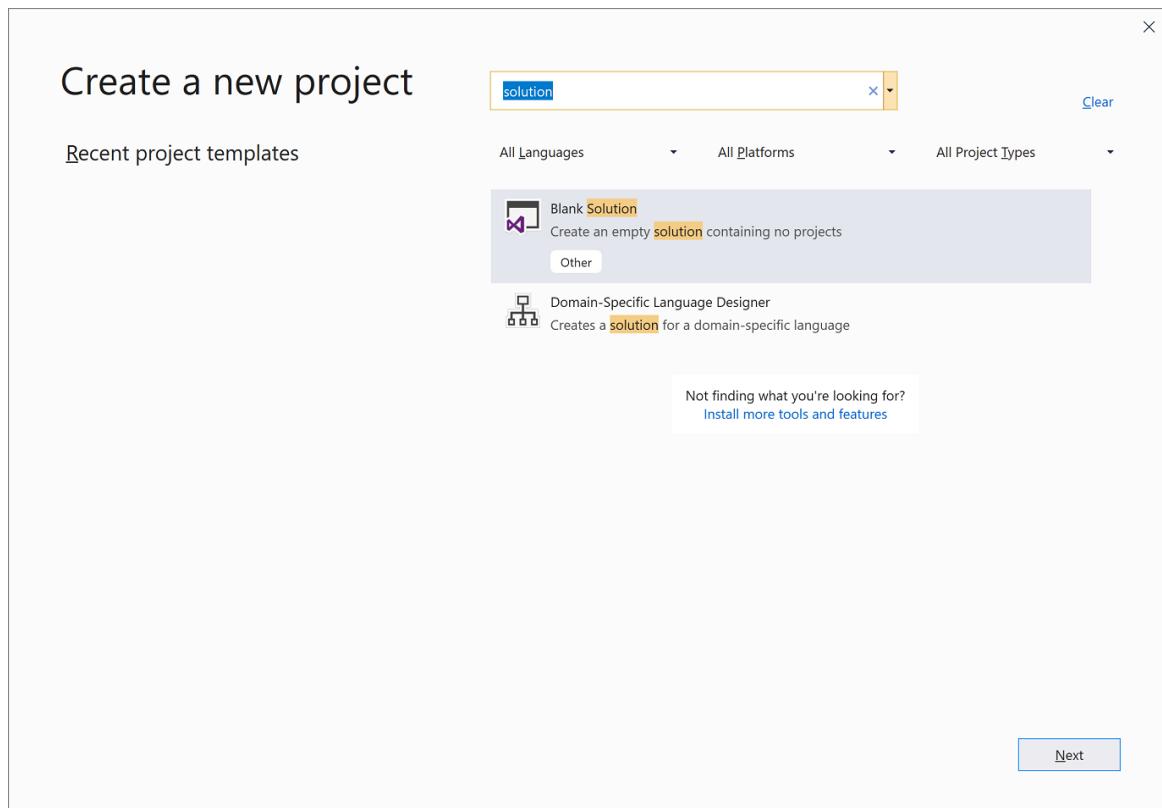
- [Visual Studio 2019 version 16.8 or a later version](#) with the **.NET Core cross-platform development** workload installed. The .NET 5.0 SDK is automatically installed when you select this workload. This tutorial assumes you have enabled **Show all .NET Core templates in the New project**, as shown in [Tutorial: Create a .NET console application using Visual Studio](#).

Create a solution

Start by creating a blank solution to put the class library project in. A Visual Studio solution serves as a container for one or more projects. You'll add additional, related projects to the same solution.

To create the blank solution:

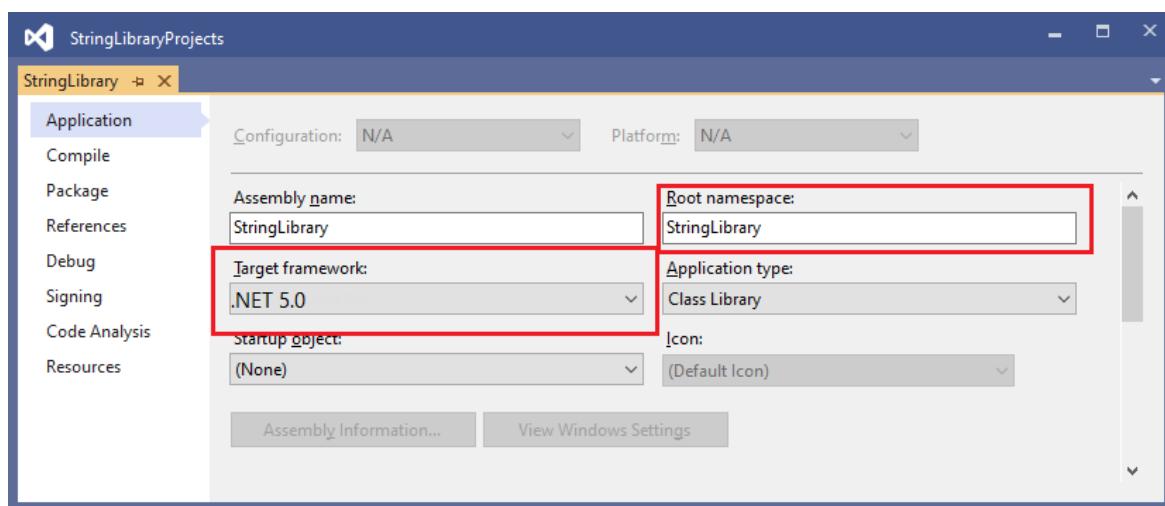
1. Start Visual Studio.
2. On the start window, choose **Create a new project**.
3. On the **Create a new project** page, enter `solution` in the search box. Choose the **Blank Solution** template, and then choose **Next**.



4. On the **Configure your new project** page, enter **ClassLibraryProjects** in the **Project name** box. Then choose **Create**.

Create a class library project

1. Add a new .NET class library project named "StringLibrary" to the solution.
 - a. Right-click on the solution in **Solution Explorer** and select **Add > New Project**.
 - b. On the **Add a new project** page, enter **library** in the search box. Choose **C# or Visual Basic** from the Language list, and then choose **All platforms** from the Platform list. Choose the **Class Library** template, and then choose **Next**.
 - c. On the **Configure your new project** page, enter **StringLibrary** in the **Project name** box, and then choose **Next**.
 - d. On the **Additional information** page, select **.NET 5.0 (Current)**, and then choose **Create**.
2. Check to make sure that the library targets the correct version of .NET. Right-click on the library project in **Solution Explorer**, and then select **Properties**. The **Target Framework** text box shows that the project targets .NET 5.0.
3. If you're using Visual Basic, clear the text in the **Root namespace** text box.



For each project, Visual Basic automatically creates a namespace that corresponds to the project name. In this tutorial, you define a top-level namespace by using the `namespace` keyword in the code file.

4. Replace the code in the code window for *Class1.cs* or *Class1.vb* with the following code, and save the file. If the language you want to use is not shown, change the language selector at the top of the page.

```

using System;

namespace UtilityLibraries
{
    public static class StringLibrary
    {
        public static bool StartsWithUpper(this string str)
        {
            if (string.IsNullOrWhiteSpace(str))
                return false;

            char ch = str[0];
            return char.IsUpper(ch);
        }
    }
}

```

```

Imports System.Runtime.CompilerServices

Namespace UtilityLibraries
    Public Module StringLibrary
        <Extension>
        Public Function StartsWithUpper(str As String) As Boolean
            If String.IsNullOrWhiteSpace(str) Then
                Return False
            End If

            Dim ch As Char = str(0)
            Return Char.IsUpper(ch)
        End Function
    End Module
End Namespace

```

The class library, `UtilityLibraries.StringLibrary`, contains a method named `StartsWithUpper`. This method returns a `Boolean` value that indicates whether the current string instance begins with an uppercase character. The Unicode standard distinguishes uppercase characters from lowercase characters. The `Char.ToUpper(Char)` method returns `true` if a character is uppercase.

`StartsWithUpper` is implemented as an [extension method](#) so that you can call it as if it were a member of the `String` class.

5. On the menu bar, select **Build > Build Solution** or press **Ctrl+Shift+B** to verify that the project compiles without error.

Add a console app to the solution

Add a console application that uses the class library. The app will prompt the user to enter a string and report whether the string begins with an uppercase character.

1. Add a new .NET console application named "ShowCase" to the solution.
 - a. Right-click on the solution in **Solution Explorer** and select **Add > New project**.
 - b. On the **Add a new project** page, enter **console** in the search box. Choose **C#** or **Visual Basic** from the Language list, and then choose **All platforms** from the Platform list.
 - c. Choose the **Console Application** template, and then choose **Next**.
 - d. On the **Configure your new project** page, enter **ShowCase** in the **Project name** box. Then choose **Next**.

- e. On the **Additional information** page, select **.NET 5.0 (Current)** in the **Target Framework** box. Then choose **Create**.
2. In the code window for the *Program.cs* or *Program.vb* file, replace all of the code with the following code.

```
using System;
using UtilityLibraries;

class Program
{
    static void Main(string[] args)
    {
        int row = 0;

        do
        {
            if (row == 0 || row >= 25)
                ResetConsole();

            string? input = Console.ReadLine();
            if (string.IsNullOrEmpty(input)) break;
            Console.WriteLine($"Input: {input} {"Begins with uppercase? ",30}: " +
                $"{{(input.StartsWithUpper() ? "Yes" : "No")}{Environment.NewLine}}");

            row += 3;
        } while (true);
        return;

        // Declare a ResetConsole local method
        void ResetConsole()
        {
            if (row > 0)
            {
                Console.WriteLine("Press any key to continue...");
                Console.ReadKey();
            }
            Console.Clear();
            Console.WriteLine($"{Environment.NewLine}Press <Enter> only to exit; otherwise, enter a
string and press <Enter>:{Environment.NewLine}");
            row = 3;
        }
    }
}
```

```

Imports UtilityLibraries

Module Program
    Dim row As Integer = 0

    Sub Main()
        Do
            If row = 0 OrElse row >= 25 Then ResetConsole()

            Dim input As String = Console.ReadLine()
            If String.IsNullOrEmpty(input) Then Return

            Console.WriteLine($"Input: {input} {"Begins with uppercase? ",30}: " +
                $"{If(input.StartsWithUpper(), "Yes", "No")}{Environment.NewLine}")
            row += 3
        Loop While True
    End Sub

    Private Sub ResetConsole()
        If row > 0 Then
            Console.WriteLine("Press any key to continue...")
            Console.ReadKey()
        End If
        Console.Clear()
        Console.WriteLine($"{Environment.NewLine}Press <Enter> only to exit; otherwise, enter a
string and press <Enter>:{Environment.NewLine}")
        row = 3
    End Sub
End Module

```

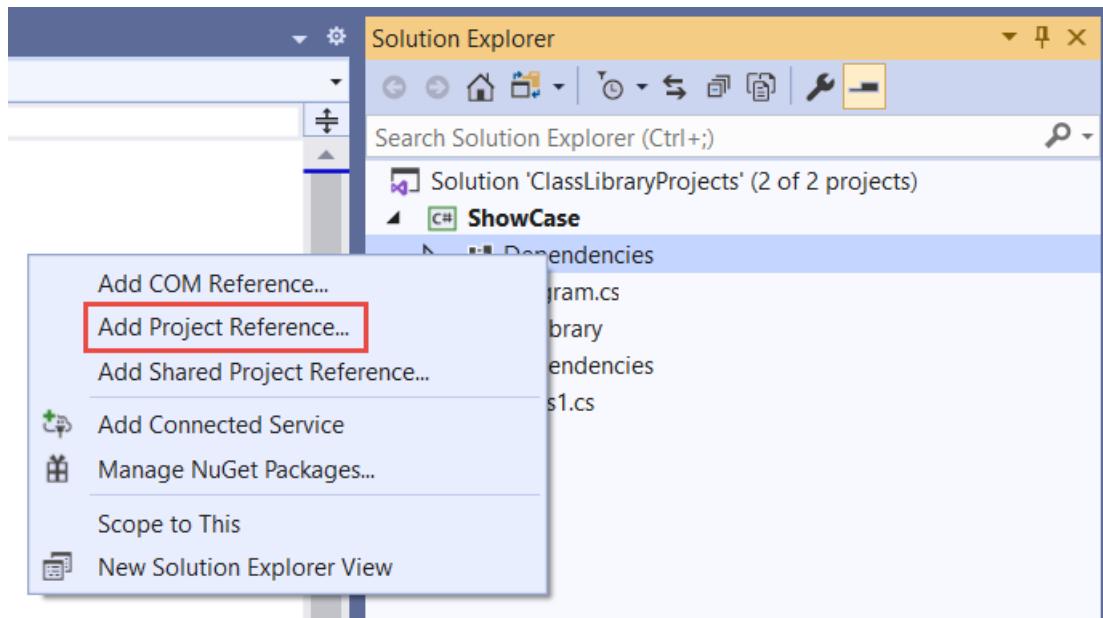
The code uses the `row` variable to maintain a count of the number of rows of data written to the console window. Whenever it's greater than or equal to 25, the code clears the console window and displays a message to the user.

The program prompts the user to enter a string. It indicates whether the string starts with an uppercase character. If the user presses the Enter key without entering a string, the application ends, and the console window closes.

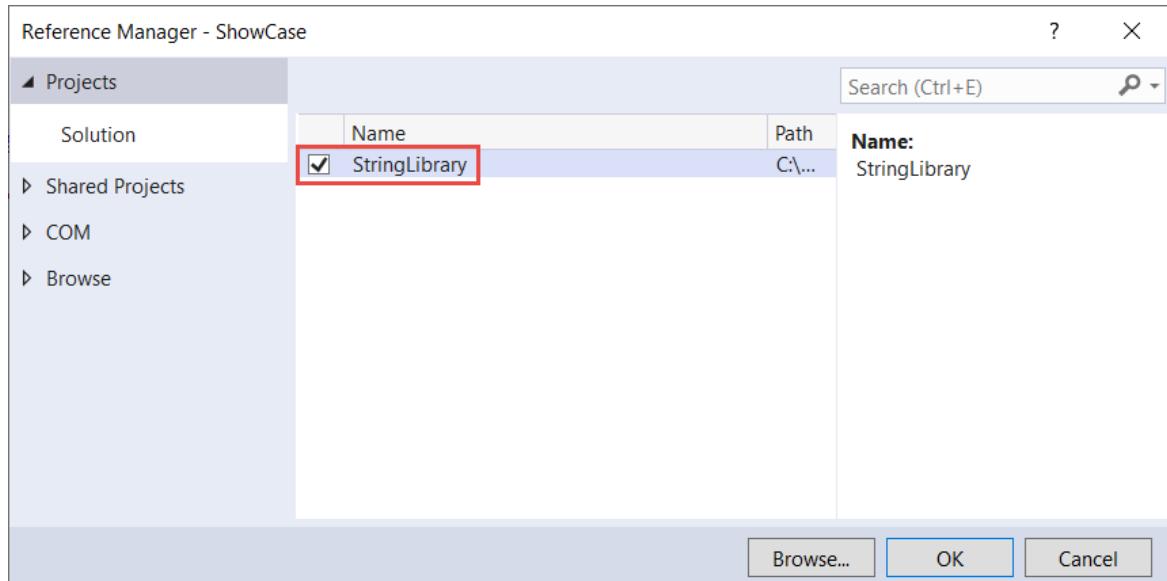
Add a project reference

Initially, the new console app project doesn't have access to the class library. To allow it to call methods in the class library, create a project reference to the class library project.

1. In **Solution Explorer**, right-click the `ShowCase` project's **Dependencies** node, and select **Add Project Reference**.

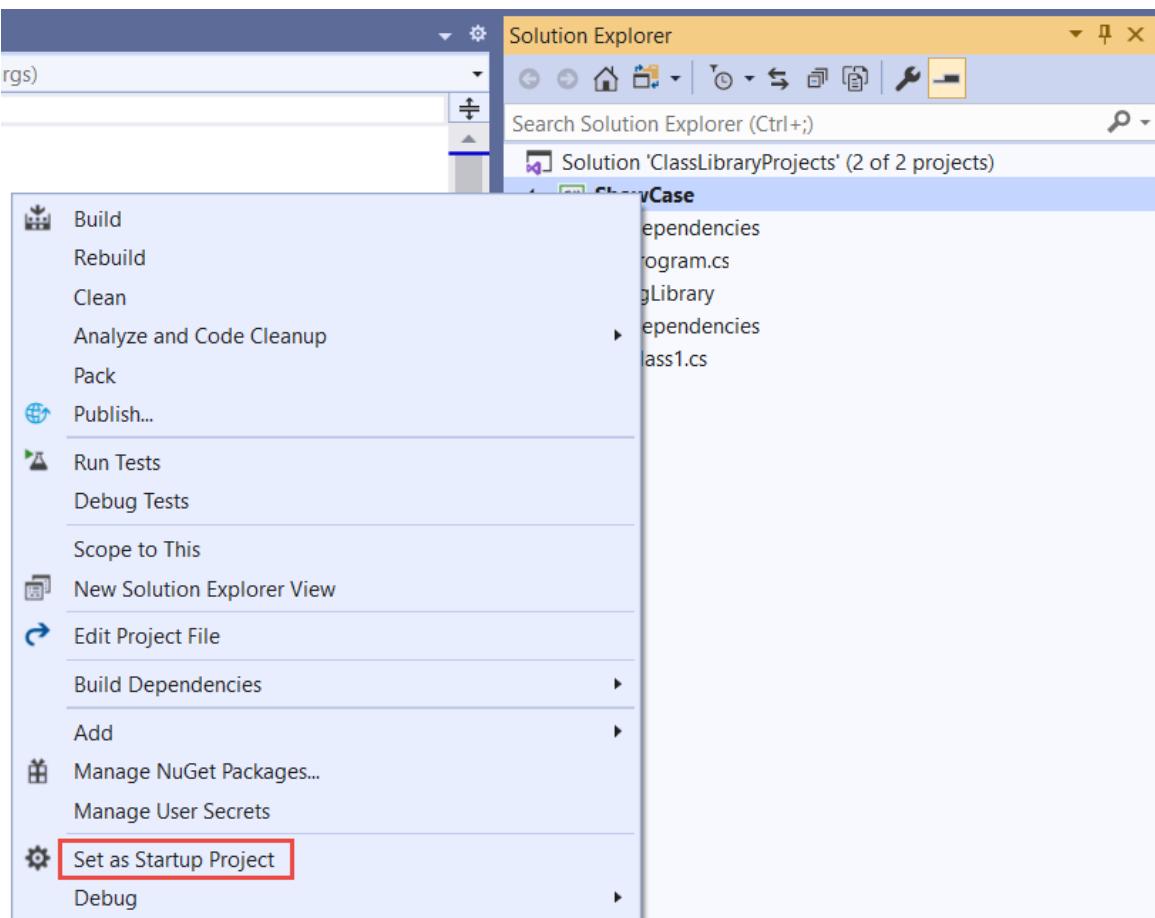


2. In the **Reference Manager** dialog, select the **StringLibrary** project, and select **OK**.

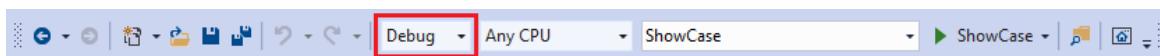


Run the app

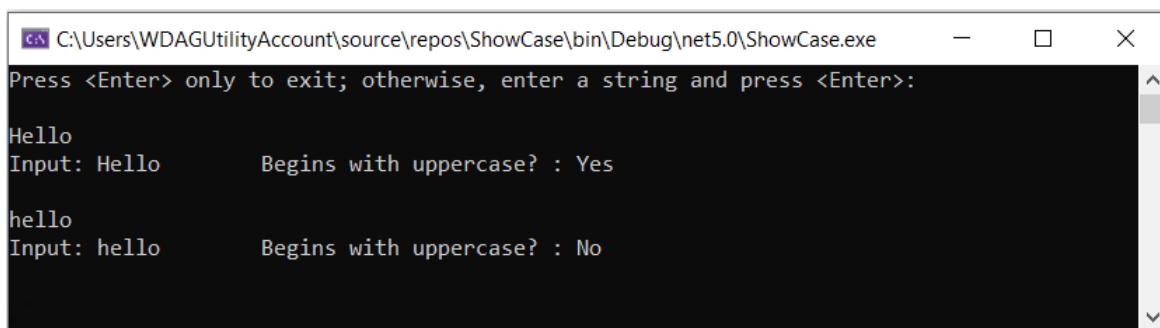
1. In **Solution Explorer**, right-click the **ShowCase** project and select **Set as StartUp Project** in the context menu.



2. Press Ctrl+F5 to compile and run the program without debugging.



3. Try out the program by entering strings and pressing Enter, then press Enter to exit.



Additional resources

- [Develop libraries with the .NET CLI](#)
- [.NET Standard versions and the platforms they support.](#)

Next steps

In this tutorial, you created a class library. In the next tutorial, you learn how to unit test the class library.

[Unit test a .NET class library using Visual Studio](#)

Or you can skip automated unit testing and learn how to share the library by creating a NuGet package:

[Create and publish a package using Visual Studio](#)

Or learn how to publish a console app. If you publish the console app from the solution you created in this

tutorial, the class library goes with it as a *.dll* file.

[Publish a .NET console application using Visual Studio](#)

This tutorial is only available for .NET 5 and .NET 6. Select one of those options at the top of the page.

Tutorial: Test a .NET class library with .NET using Visual Studio

9/20/2022 • 18 minutes to read • [Edit Online](#)

This tutorial shows how to automate unit testing by adding a test project to a solution.

Prerequisites

- This tutorial works with the solution that you create in [Create a .NET class library using Visual Studio](#).

Create a unit test project

Unit tests provide automated software testing during your development and publishing. [MSTest](#) is one of three test frameworks you can choose from. The others are [xUnit](#) and [nUnit](#).

1. Start Visual Studio.
2. Open the `ClassLibraryProjects` solution you created in [Create a .NET class library using Visual Studio](#).
3. Add a new unit test project named "StringLibraryTest" to the solution.
 - a. Right-click on the solution in **Solution Explorer** and select **Add > New project**.
 - b. On the **Add a new project** page, enter **mstest** in the search box. Choose **C# or Visual Basic** from the Language list, and then choose **All platforms** from the Platform list.
 - c. Choose the **MSTest Test Project** template, and then choose **Next**.
 - d. On the **Configure your new project** page, enter **StringLibraryTest** in the **Project name** box. Then choose **Next**.
 - e. On the **Additional information** page, select **.NET 6 (Long-term support)** in the **Framework** box. Then choose **Create**.
4. Visual Studio creates the project and opens the class file in the code window with the following code. If the language you want to use is not shown, change the language selector at the top of the page.

```
using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace StringLibraryTest
{
    [TestClass]
    public class UnitTest1
    {
        [TestMethod]
        public void TestMethod1()
        {
        }
    }
}
```

```

Imports Microsoft.VisualStudio.TestTools.UnitTesting

Namespace StringLibraryTest
    <TestClass>
    Public Class UnitTest1
        <TestMethod>
        Sub TestSub()

            End Sub
        End Class
    End Namespace

```

The source code created by the unit test template does the following:

- It imports the `Microsoft.VisualStudio.TestTools.UnitTesting` namespace, which contains the types used for unit testing.
- It applies the `TestClassAttribute` attribute to the `UnitTest1` class.
- It applies the `TestMethodAttribute` attribute to define `TestMethod1` in C# or `TestSub` in Visual Basic.

Each method tagged with `[TestMethod]` in a test class tagged with `[TestClass]` is executed automatically when the unit test is run.

Add a project reference

For the test project to work with the `StringLibrary` class, add a reference in the `StringLibraryTest` project to the `StringLibrary` project.

1. In **Solution Explorer**, right-click the **Dependencies** node of the `StringLibraryTest` project and select **Add Project Reference** from the context menu.
2. In the **Reference Manager** dialog, expand the **Projects** node, and select the box next to `StringLibrary`. Adding a reference to the `StringLibrary` assembly allows the compiler to find `StringLibrary` methods while compiling the `StringLibraryTest` project.
3. Select **OK**.

Add and run unit test methods

When Visual Studio runs a unit test, it executes each method that is marked with the `TestMethodAttribute` attribute in a class that is marked with the `TestClassAttribute` attribute. A test method ends when the first failure is found or when all tests contained in the method have succeeded.

The most common tests call members of the `Assert` class. Many assert methods include at least two parameters, one of which is the expected test result and the other of which is the actual test result. Some of the `Assert` class's most frequently called methods are shown in the following table:

ASSERT METHODS	FUNCTION
<code>Assert.AreEqual</code>	Verifies that two values or objects are equal. The assert fails if the values or objects aren't equal.
<code>Assert.AreSame</code>	Verifies that two object variables refer to the same object. The assert fails if the variables refer to different objects.
<code>Assert.IsFalse</code>	Verifies that a condition is <code>false</code> . The assert fails if the condition is <code>true</code> .

ASSERT METHODS	FUNCTION
<code>Assert.IsNotNull</code>	Verifies that an object isn't <code>null</code> . The assert fails if the object is <code>null</code> .

You can also use the `Assert.ThrowsException` method in a test method to indicate the type of exception it's expected to throw. The test fails if the specified exception isn't thrown.

In testing the `StringLibrary.StartsWithUpper` method, you want to provide a number of strings that begin with an uppercase character. You expect the method to return `true` in these cases, so you can call the `Assert.IsTrue` method. Similarly, you want to provide a number of strings that begin with something other than an uppercase character. You expect the method to return `false` in these cases, so you can call the `Assert.IsFalse` method.

Since your library method handles strings, you also want to make sure that it successfully handles an `empty string` (`String.Empty`), a valid string that has no characters and whose `Length` is 0, and a `null` string that hasn't been initialized. You can call `StartsWithUpper` directly as a static method and pass a single `String` argument. Or you can call `StartsWithUpper` as an extension method on a `string` variable assigned to `null`.

You'll define three methods, each of which calls an `Assert` method for each element in a string array. You'll call a method overload that lets you specify an error message to be displayed in case of test failure. The message identifies the string that caused the failure.

To create the test methods:

1. In the `UnitTest1.cs` or `UnitTest1.vb` code window, replace the code with the following code:

```

using Microsoft.VisualStudio.TestTools.UnitTesting;
using UtilityLibraries;

namespace StringLibraryTest
{
    [TestClass]
    public class UnitTest1
    {
        [TestMethod]
        public void TestStartsWithUpper()
        {
            // Tests that we expect to return true.
            string[] words = { "Alphabet", "Zebra", "ABC", "Αθήνα", "Москва" };
            foreach (var word in words)
            {
                bool result = word.StartsWithUpper();
                Assert.IsTrue(result,
                    string.Format("Expected for '{0}': true; Actual: {1}",
                        word, result));
            }
        }

        [TestMethod]
        public void TestDoesNotStartWithUpper()
        {
            // Tests that we expect to return false.
            string[] words = { "alphabet", "zebra", "abc", "αυτοκινητοβιομηχανία", "государство",
                "1234", ".", ";", " " };
            foreach (var word in words)
            {
                bool result = word.StartsWithUpper();
                Assert.IsFalse(result,
                    string.Format("Expected for '{0}': false; Actual: {1}",
                        word, result));
            }
        }

        [TestMethod]
        public void DirectCallWithNullOrEmpty()
        {
            // Tests that we expect to return false.
            string?[] words = { string.Empty, null };
            foreach (var word in words)
            {
                bool result = StringLibrary.StartsWithUpper(word ?? string.Empty);
                Assert.IsFalse(result,
                    string.Format("Expected for '{0}': false; Actual: {1}",
                        word == null ? "<null>" : word, result));
            }
        }
    }
}

```

```

Imports Microsoft.VisualStudio.TestTools.UnitTesting
Imports UtilityLibraries

Namespace StringLibraryTest
    <TestClass>
    Public Class UnitTest1
        <TestMethod>
        Public Sub TestStartsWithUpper()
            ' Tests that we expect to return true.
            Dim words() As String = {"Alphabet", "Zebra", "ABC", "Αθήνα", "Москва"}
            For Each word In words
                Dim result As Boolean = word.StartsWithUpper()
                Assert.IsTrue(result,
                    $"Expected for '{word}': true; Actual: {result}")
            Next
        End Sub

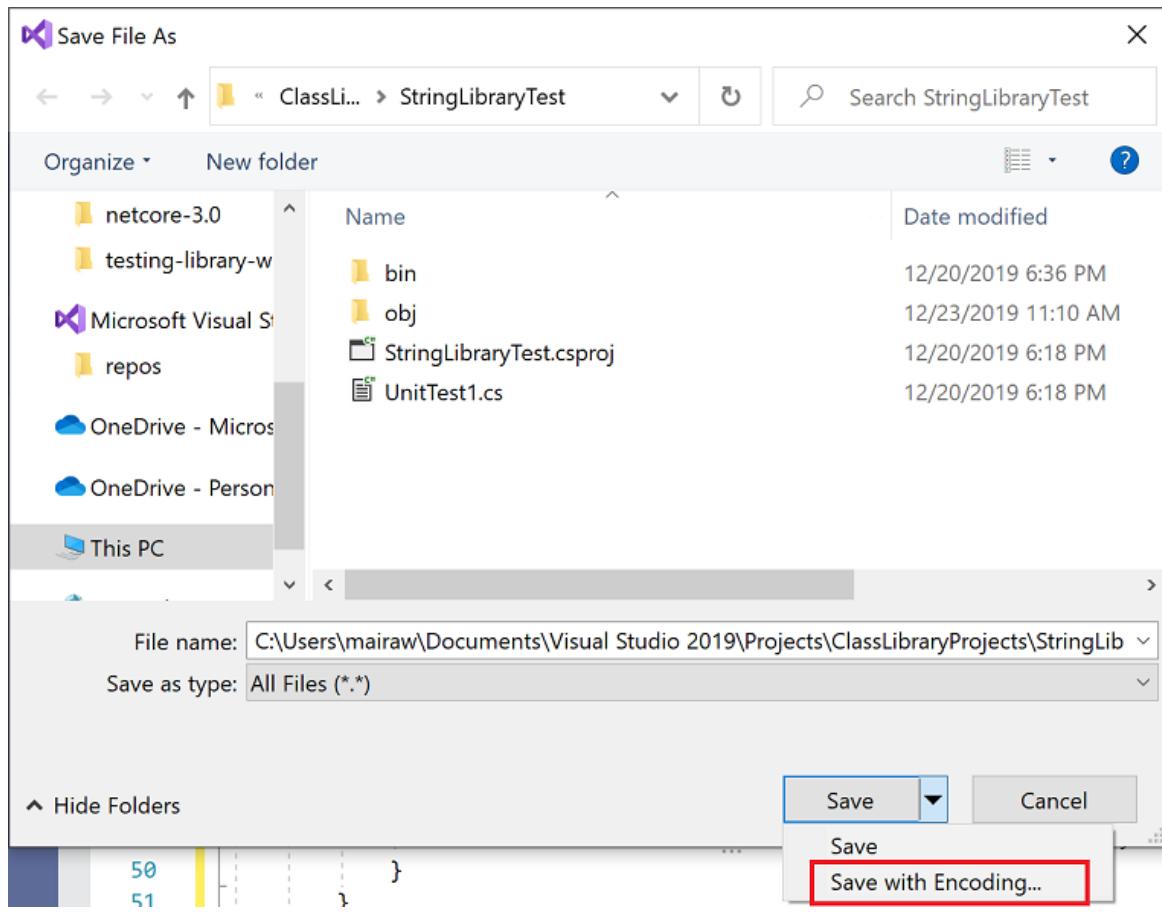
        <TestMethod>
        Public Sub TestDoesNotStartWithUpper()
            ' Tests that we expect to return false.
            Dim words() As String = {"alphabet", "zebra", "abc", "αυτοκινητοβιομηχανία",
"государство",
                "1234", ".", ";", " "}
            For Each word In words
                Dim result As Boolean = word.StartsWithUpper()
                Assert.IsFalse(result,
                    $"Expected for '{word}': false; Actual: {result}")
            Next
        End Sub

        <TestMethod>
        Public Sub DirectCallWithNullOrEmpty()
            ' Tests that we expect to return false.
            Dim words() As String = {String.Empty, Nothing}
            For Each word In words
                Dim result As Boolean = StringLibrary.StartsWithUpper(word)
                Assert.IsFalse(result,
                    $"Expected for '{If(word Is Nothing, "<null>", word)}': false; Actual:
{result}")
            Next
        End Sub
    End Class
End Namespace

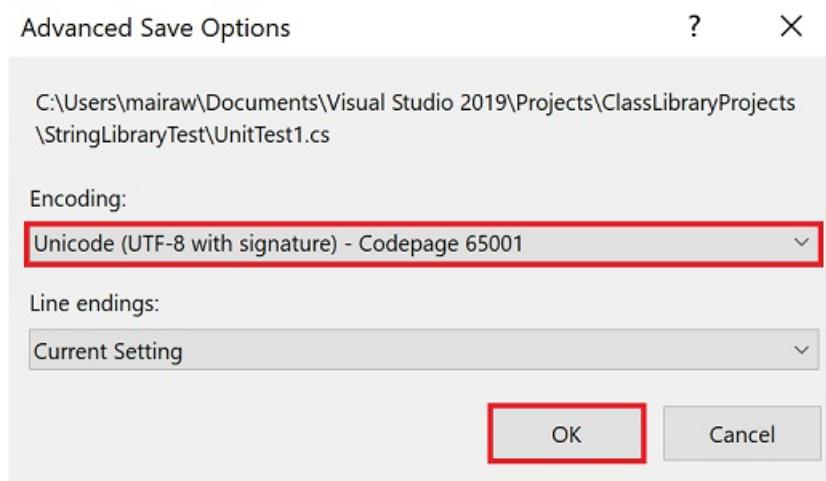
```

The test of uppercase characters in the `TestStartsWithUpper` method includes the Greek capital letter alpha (U+0391) and the Cyrillic capital letter EM (U+041C). The test of lowercase characters in the `TestDoesNotStartWithUpper` method includes the Greek small letter alpha (U+03B1) and the Cyrillic small letter Ghe (U+0433).

2. On the menu bar, select **File > Save UnitTest1.cs As** or **File > Save UnitTest1.vb As**. In the **Save File As** dialog, select the arrow beside the **Save** button, and select **Save with Encoding**.

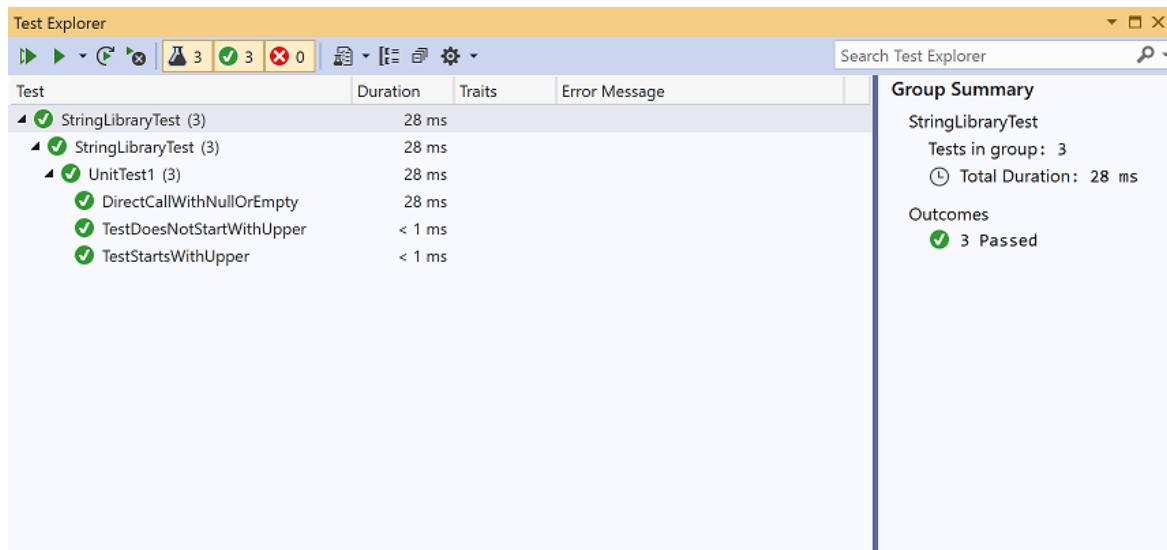


3. In the **Confirm Save As** dialog, select the **Yes** button to save the file.
4. In the **Advanced Save Options** dialog, select **Unicode (UTF-8 with signature) - Codepage 65001** from the **Encoding** drop-down list and select **OK**.



If you fail to save your source code as a UTF8-encoded file, Visual Studio may save it as an ASCII file. When that happens, the runtime doesn't accurately decode the UTF8 characters outside of the ASCII range, and the test results won't be correct.

5. On the menu bar, select **Test > Run All Tests**. If the **Test Explorer** window doesn't open, open it by choosing **Test > Test Explorer**. The three tests are listed in the **Passed Tests** section, and the **Summary** section reports the result of the test run.



Handle test failures

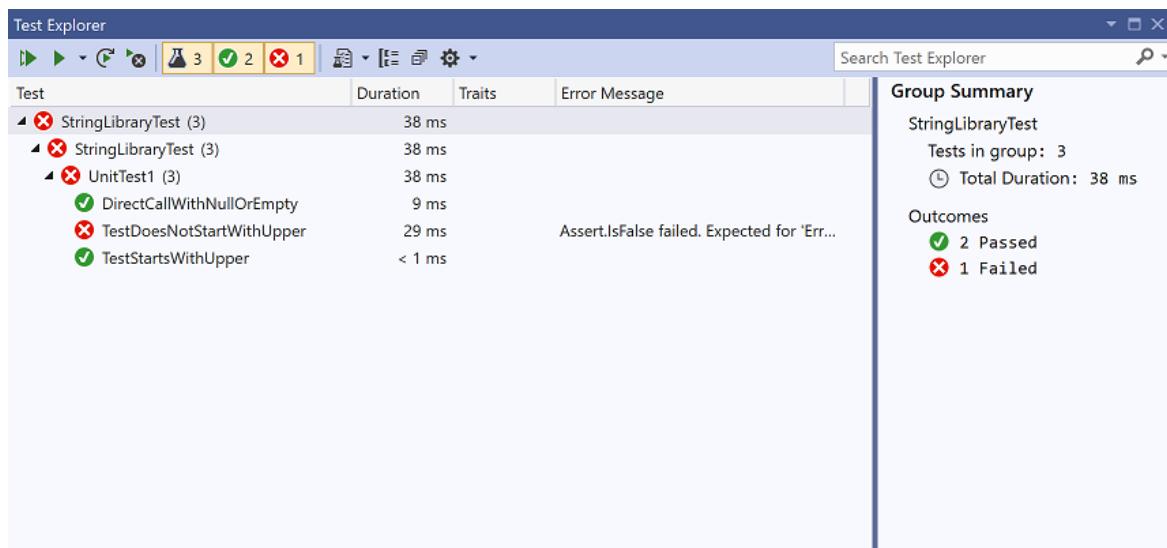
If you're doing test-driven development (TDD), you write tests first and they fail the first time you run them. Then you add code to the app that makes the test succeed. For this tutorial, you created the test after writing the app code that it validates, so you haven't seen the test fail. To validate that a test fails when you expect it to fail, add an invalid value to the test input.

1. Modify the `words` array in the `TestDoesNotStartWithUpper` method to include the string "Error". You don't need to save the file because Visual Studio automatically saves open files when a solution is built to run tests.

```
string[] words = { "alphabet", "Error", "zebra", "abc", "αυτοκινητοβιομηχανία", "государство",
                   "1234", ".", ";" };
```

```
Dim words() As String = { "alphabet", "Error", "zebra", "abc", "αυτοκινητοβιομηχανία", "государство",
                           "1234", ".", ";" };
```

2. Run the test by selecting **Test > Run All Tests** from the menu bar. The **Test Explorer** window indicates that two tests succeeded and one failed.



3. Select the failed test, `TestDoesNotStartWithUpper`.

The **Test Explorer** window displays the message produced by the assert: "Assert.IsFalse failed. Expected

for 'Error': false; actual: True". Because of the failure, no strings in the array after "Error" were tested.

Test Detail Summary

✖ TestDoesNotStartWithUpper

Source: [UnitTest1.cs](#) line 25

Duration: 29 ms

Message:

```
Assert.IsFalse failed. Expected for 'Error': false; Actual: True
```

Stack Trace:

```
UnitTest1.TestDoesNotStartWithUpper\(\) line 34
```

4. Remove the string "Error" that you added in step 1. Rerun the test and the tests pass.

Test the Release version of the library

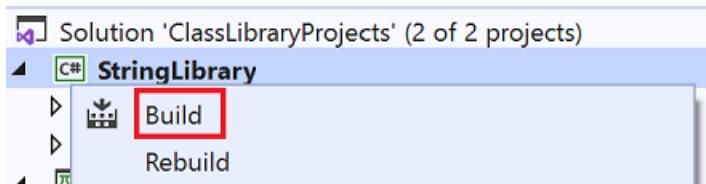
Now that the tests have all passed when running the Debug build of the library, run the tests an additional time against the Release build of the library. A number of factors, including compiler optimizations, can sometimes produce different behavior between Debug and Release builds.

To test the Release build:

1. In the Visual Studio toolbar, change the build configuration from **Debug** to **Release**.



2. In **Solution Explorer**, right-click the **StringLibrary** project and select **Build** from the context menu to recompile the library.



3. Run the unit tests by choosing **Test > Run All Tests** from the menu bar. The tests pass.

Debug tests

If you're using Visual Studio as your IDE, you can use the same process shown in [Tutorial: Debug a .NET console application using Visual Studio](#) to debug code using your unit test project. Instead of starting the *ShowCase* app project, right-click the **StringLibraryTests** project, and select **Debug Tests** from the context menu.

Visual Studio starts the test project with the debugger attached. Execution will stop at any breakpoint you've added to the test project or the underlying library code.

Additional resources

- [Unit test basics - Visual Studio](#)
- [Unit testing in .NET](#)

Next steps

In this tutorial, you unit tested a class library. You can make the library available to others by publishing it to

NuGet as a package. To learn how, follow a NuGet tutorial:

[Create and publish a NuGet package using Visual Studio](#)

If you publish a library as a NuGet package, others can install and use it. To learn how, follow a NuGet tutorial:

[Install and use a package in Visual Studio](#)

A library doesn't have to be distributed as a package. It can be bundled with a console app that uses it. To learn how to publish a console app, see the earlier tutorial in this series:

[Publish a .NET console application using Visual Studio](#)

This tutorial shows how to automate unit testing by adding a test project to a solution.

Prerequisites

- This tutorial works with the solution that you create in [Create a .NET class library using Visual Studio](#).

Create a unit test project

Unit tests provide automated software testing during your development and publishing. [MSTest](#) is one of three test frameworks you can choose from. The others are [xUnit](#) and [nUnit](#).

1. Start Visual Studio.
2. Open the `ClassLibraryProjects` solution you created in [Create a .NET class library using Visual Studio](#).
3. Add a new unit test project named "StringLibraryTest" to the solution.
 - a. Right-click on the solution in **Solution Explorer** and select **Add > New project**.
 - b. On the **Add a new project** page, enter `mstest` in the search box. Choose **C# or Visual Basic** from the Language list, and then choose **All platforms** from the Platform list.
 - c. Choose the **MSTest Test Project** template, and then choose **Next**.
 - d. On the **Configure your new project** page, enter `StringLibraryTest` in the **Project name** box. Then choose **Next**.
 - e. On the **Additional information** page, select **.NET 5.0 (Current)** in the **Target Framework** box. Then choose **Create**.
4. Visual Studio creates the project and opens the class file in the code window with the following code. If the language you want to use is not shown, change the language selector at the top of the page.

```
using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace StringLibraryTest
{
    [TestClass]
    public class UnitTest1
    {
        [TestMethod]
        public void TestMethod1()
        {
        }
    }
}
```

```

Imports Microsoft.VisualStudio.TestTools.UnitTesting

Namespace StringLibraryTest
    <TestClass>
    Public Class UnitTest1
        <TestMethod>
        Sub TestSub()

            End Sub
        End Class
    End Namespace

```

The source code created by the unit test template does the following:

- It imports the `Microsoft.VisualStudio.TestTools.UnitTesting` namespace, which contains the types used for unit testing.
- It applies the `TestClassAttribute` attribute to the `UnitTest1` class.
- It applies the `TestMethodAttribute` attribute to define `TestMethod1` in C# or `TestSub` in Visual Basic.

Each method tagged with `[TestMethod]` in a test class tagged with `[TestClass]` is executed automatically when the unit test is run.

Add a project reference

For the test project to work with the `StringLibrary` class, add a reference in the `StringLibraryTest` project to the `StringLibrary` project.

1. In **Solution Explorer**, right-click the **Dependencies** node of the `StringLibraryTest` project and select **Add Project Reference** from the context menu.
2. In the **Reference Manager** dialog, expand the **Projects** node, and select the box next to `StringLibrary`. Adding a reference to the `StringLibrary` assembly allows the compiler to find `StringLibrary` methods while compiling the `StringLibraryTest` project.
3. Select **OK**.

Add and run unit test methods

When Visual Studio runs a unit test, it executes each method that is marked with the `TestMethodAttribute` attribute in a class that is marked with the `TestClassAttribute` attribute. A test method ends when the first failure is found or when all tests contained in the method have succeeded.

The most common tests call members of the `Assert` class. Many assert methods include at least two parameters, one of which is the expected test result and the other of which is the actual test result. Some of the `Assert` class's most frequently called methods are shown in the following table:

ASSERT METHODS	FUNCTION
<code>Assert.AreEqual</code>	Verifies that two values or objects are equal. The assert fails if the values or objects aren't equal.
<code>Assert.AreSame</code>	Verifies that two object variables refer to the same object. The assert fails if the variables refer to different objects.
<code>Assert.IsFalse</code>	Verifies that a condition is <code>false</code> . The assert fails if the condition is <code>true</code> .

ASSERT METHODS	FUNCTION
<code>Assert.IsNotNull</code>	Verifies that an object isn't <code>null</code> . The assert fails if the object is <code>null</code> .

You can also use the `Assert.ThrowsException` method in a test method to indicate the type of exception it's expected to throw. The test fails if the specified exception isn't thrown.

In testing the `StringLibrary.StartsWithUpper` method, you want to provide a number of strings that begin with an uppercase character. You expect the method to return `true` in these cases, so you can call the `Assert.IsTrue` method. Similarly, you want to provide a number of strings that begin with something other than an uppercase character. You expect the method to return `false` in these cases, so you can call the `Assert.IsFalse` method.

Since your library method handles strings, you also want to make sure that it successfully handles an `empty string` (`String.Empty`), a valid string that has no characters and whose `Length` is 0, and a `null` string that hasn't been initialized. You can call `StartsWithUpper` directly as a static method and pass a single `String` argument. Or you can call `StartsWithUpper` as an extension method on a `string` variable assigned to `null`.

You'll define three methods, each of which calls an `Assert` method for each element in a string array. You'll call a method overload that lets you specify an error message to be displayed in case of test failure. The message identifies the string that caused the failure.

To create the test methods:

1. In the `UnitTest1.cs` or `UnitTest1.vb` code window, replace the code with the following code:

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using UtilityLibraries;

namespace StringLibraryTest
{
    [TestClass]
    public class UnitTest1
    {
        [TestMethod]
        public void TestStartsWithUpper()
        {
            // Tests that we expect to return true.
            string[] words = { "Alphabet", "Zebra", "ABC", "Αθήνα", "Москва" };
            foreach (var word in words)
            {
                bool result = word.StartsWithUpper();
                Assert.IsTrue(result,
                    string.Format("Expected for '{0}': true; Actual: {1}",
                        word, result));
            }
        }

        [TestMethod]
        public void TestDoesNotStartWithUpper()
        {
            // Tests that we expect to return false.
            string[] words = { "alphabet", "zebra", "abc", "αυτοκινητοβιομηχανία", "государство",
                "1234", ".", ";", " " };
            foreach (var word in words)
            {
                bool result = word.StartsWithUpper();
                Assert.IsFalse(result,
                    string.Format("Expected for '{0}': false; Actual: {1}",
                        word, result));
            }
        }

        [TestMethod]
        public void DirectCallWithNullOrEmpty()
        {
            // Tests that we expect to return false.
            string?[] words = { string.Empty, null };
            foreach (var word in words)
            {
                bool result = StringLibrary.StartsWithUpper(word ?? string.Empty);
                Assert.IsFalse(result,
                    string.Format("Expected for '{0}': false; Actual: {1}",
                        word == null ? "<null>" : word, result));
            }
        }
    }
}
```

```

Imports Microsoft.VisualStudio.TestTools.UnitTesting
Imports UtilityLibraries

Namespace StringLibraryTest
    <TestClass>
    Public Class UnitTest1
        <TestMethod>
        Public Sub TestStartsWithUpper()
            ' Tests that we expect to return true.
            Dim words() As String = {"Alphabet", "Zebra", "ABC", "Αθήνα", "Москва"}
            For Each word In words
                Dim result As Boolean = word.StartsWithUpper()
                Assert.IsTrue(result,
                    $"Expected for '{word}': true; Actual: {result}")
            Next
        End Sub

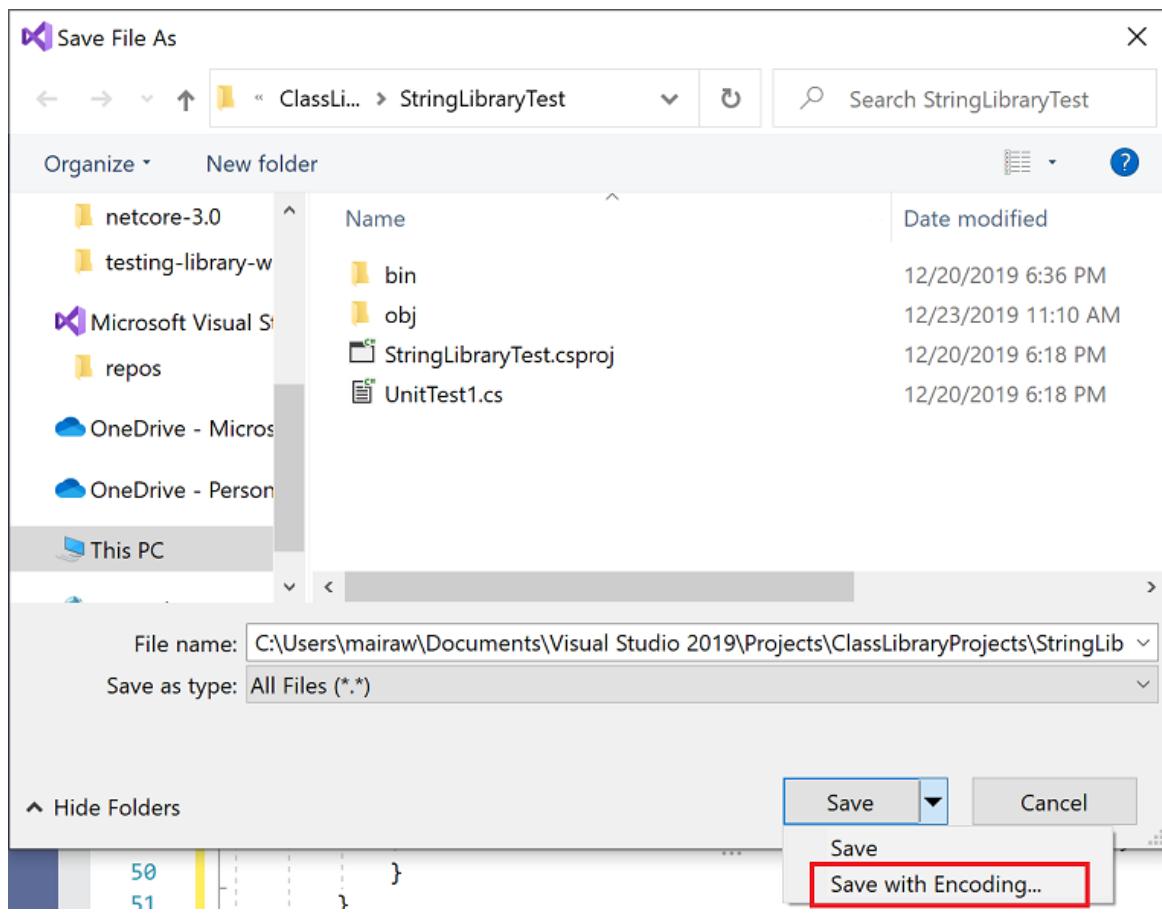
        <TestMethod>
        Public Sub TestDoesNotStartWithUpper()
            ' Tests that we expect to return false.
            Dim words() As String = {"alphabet", "zebra", "abc", "αυτοκινητοβιομηχανία",
"государство",
                "1234", ".", ";", " "}
            For Each word In words
                Dim result As Boolean = word.StartsWithUpper()
                Assert.IsFalse(result,
                    $"Expected for '{word}': false; Actual: {result}")
            Next
        End Sub

        <TestMethod>
        Public Sub DirectCallWithNullOrEmpty()
            ' Tests that we expect to return false.
            Dim words() As String = {String.Empty, Nothing}
            For Each word In words
                Dim result As Boolean = StringLibrary.StartsWithUpper(word)
                Assert.IsFalse(result,
                    $"Expected for '{If(word Is Nothing, "<null>", word)}': false; Actual:
{result}")
            Next
        End Sub
    End Class
End Namespace

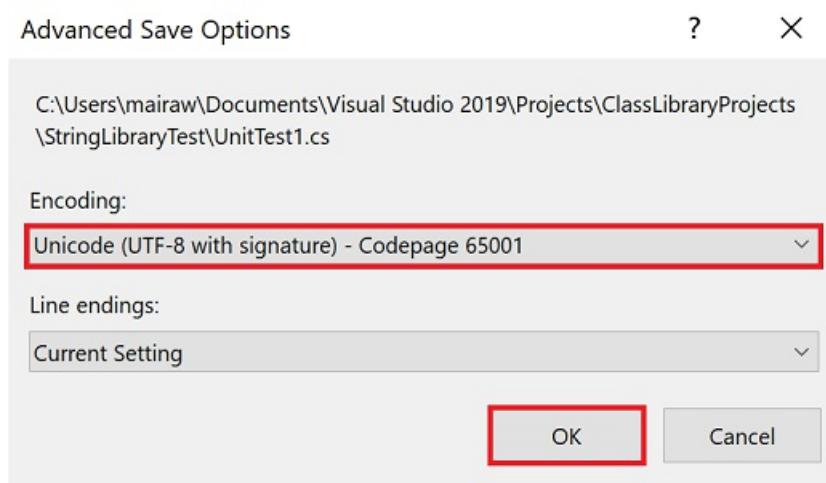
```

The test of uppercase characters in the `TestStartsWithUpper` method includes the Greek capital letter alpha (U+0391) and the Cyrillic capital letter EM (U+041C). The test of lowercase characters in the `TestDoesNotStartWithUpper` method includes the Greek small letter alpha (U+03B1) and the Cyrillic small letter Ghe (U+0433).

2. On the menu bar, select **File > Save UnitTest1.cs As** or **File > Save UnitTest1.vb As**. In the **Save File As** dialog, select the arrow beside the **Save** button, and select **Save with Encoding**.

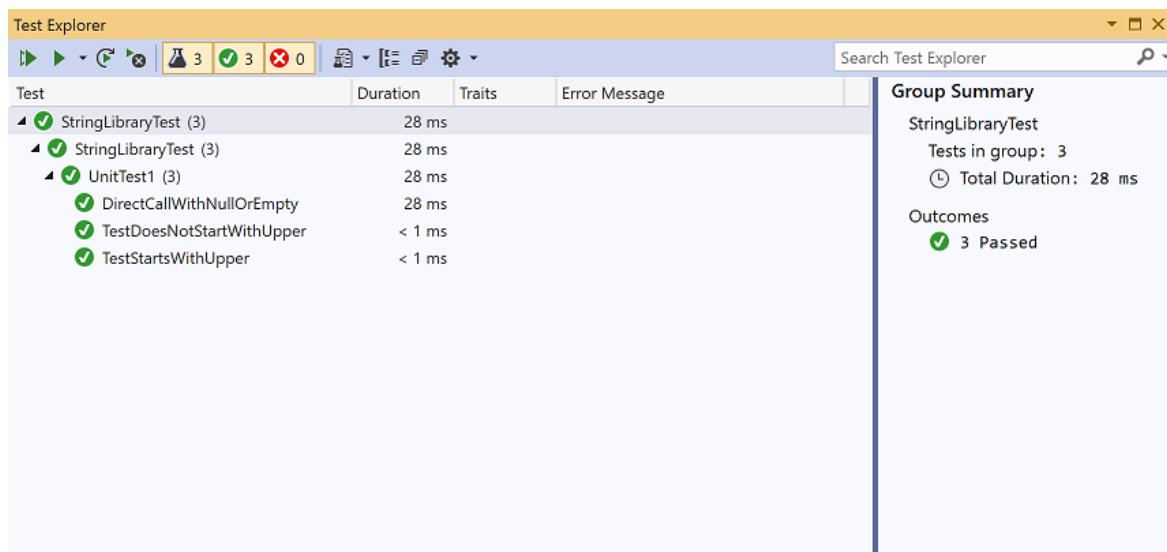


3. In the **Confirm Save As** dialog, select the **Yes** button to save the file.
4. In the **Advanced Save Options** dialog, select **Unicode (UTF-8 with signature) - Codepage 65001** from the **Encoding** drop-down list and select **OK**.



If you fail to save your source code as a UTF8-encoded file, Visual Studio may save it as an ASCII file. When that happens, the runtime doesn't accurately decode the UTF8 characters outside of the ASCII range, and the test results won't be correct.

5. On the menu bar, select **Test > Run All Tests**. If the **Test Explorer** window doesn't open, open it by choosing **Test > Test Explorer**. The three tests are listed in the **Passed Tests** section, and the **Summary** section reports the result of the test run.



Handle test failures

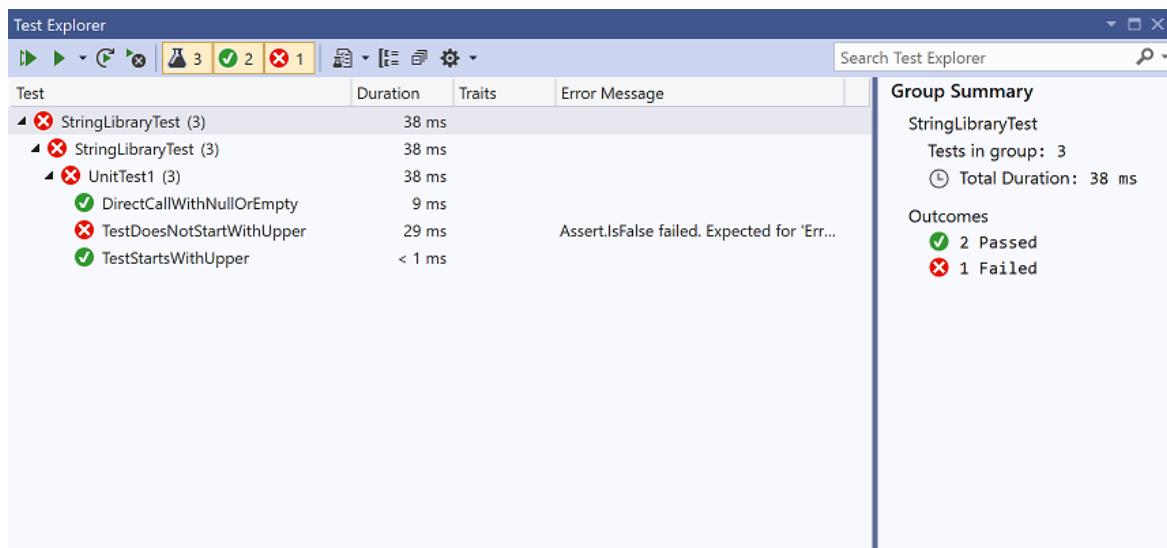
If you're doing test-driven development (TDD), you write tests first and they fail the first time you run them. Then you add code to the app that makes the test succeed. For this tutorial, you created the test after writing the app code that it validates, so you haven't seen the test fail. To validate that a test fails when you expect it to fail, add an invalid value to the test input.

1. Modify the `words` array in the `TestDoesNotStartWithUpper` method to include the string "Error". You don't need to save the file because Visual Studio automatically saves open files when a solution is built to run tests.

```
string[] words = { "alphabet", "Error", "zebra", "abc", "αυτοκινητοβιομηχανία", "государство",
    "1234", ".", ";" };
```

```
Dim words() As String = { "alphabet", "Error", "zebra", "abc", "αυτοκινητοβιομηχανία", "государство",
    "1234", ".", ";" };
```

2. Run the test by selecting **Test > Run All Tests** from the menu bar. The **Test Explorer** window indicates that two tests succeeded and one failed.



3. Select the failed test, `TestDoesNotStartWith`.

The **Test Explorer** window displays the message produced by the assert: "Assert.IsFalse failed. Expected

for 'Error': false; actual: True". Because of the failure, no strings in the array after "Error" were tested.

Test Detail Summary

✖ TestDoesNotStartWithUpper

Source: [UnitTest1.cs](#) line 25

Duration: 29 ms

Message:

 Assert.IsFalse failed. Expected for 'Error': false; Actual: True

Stack Trace:

[UnitTest1.TestDoesNotStartWithUpper\(\)](#) line 34

4. Remove the string "Error" that you added in step 1. Rerun the test and the tests pass.

Test the Release version of the library

Now that the tests have all passed when running the Debug build of the library, run the tests an additional time against the Release build of the library. A number of factors, including compiler optimizations, can sometimes produce different behavior between Debug and Release builds.

To test the Release build:

1. In the Visual Studio toolbar, change the build configuration from **Debug** to **Release**.



2. In **Solution Explorer**, right-click the **StringLibrary** project and select **Build** from the context menu to recompile the library.



3. Run the unit tests by choosing **Test Run > All Tests** from the menu bar. The tests pass.

Debug tests

If you're using Visual Studio as your IDE, you can use the same process shown in [Tutorial: Debug a .NET console application using Visual Studio](#) to debug code using your unit test project. Instead of starting the *ShowCase* app project, right-click the **StringLibraryTests** project, and select **Debug Tests** from the context menu.

Visual Studio starts the test project with the debugger attached. Execution will stop at any breakpoint you've added to the test project or the underlying library code.

Additional resources

- [Unit test basics - Visual Studio](#)
- [Unit testing in .NET](#)

Next steps

In this tutorial, you unit tested a class library. You can make the library available to others by publishing it to

NuGet as a package. To learn how, follow a NuGet tutorial:

[Create and publish a NuGet package using Visual Studio](#)

If you publish a library as a NuGet package, others can install and use it. To learn how, follow a NuGet tutorial:

[Install and use a package in Visual Studio](#)

A library doesn't have to be distributed as a package. It can be bundled with a console app that uses it. To learn how to publish a console app, see the earlier tutorial in this series:

[Publish a .NET console application using Visual Studio](#)

This tutorial is only available for .NET 5 and .NET 6. Select one of those options at the top of the page.

Tutorial: Create a .NET console application using Visual Studio Code

9/20/2022 • 7 minutes to read • [Edit Online](#)

This tutorial shows how to create and run a .NET console application by using Visual Studio Code and the .NET CLI. Project tasks, such as creating, compiling, and running a project are done by using the .NET CLI. You can follow this tutorial with a different code editor and run commands in a terminal if you prefer.

Prerequisites

- [Visual Studio Code](#) with the [C# extension](#) installed. For information about how to install extensions on Visual Studio Code, see [VS Code Extension Marketplace](#).
- The [.NET 6 SDK](#).

Create the app

Create a .NET console app project named "HelloWorld".

1. Start Visual Studio Code.
2. Select **File > Open Folder** (**File > Open...** on macOS) from the main menu.
3. In the **Open Folder** dialog, create a *HelloWorld* folder and select it. Then click **Select Folder** (**Open** on macOS).

The folder name becomes the project name and the namespace name by default. You'll add code later in the tutorial that assumes the project namespace is `HelloWorld`.

4. In the **Do you trust the authors of the files in this folder?** dialog, select **Yes, I trust the authors**.
5. Open the **Terminal** in Visual Studio Code by selecting **View > Terminal** from the main menu.

The **Terminal** opens with the command prompt in the *HelloWorld* folder.

6. In the **Terminal**, enter the following command:

```
dotnet new console --framework net6.0
```

The project template creates a simple application that displays "Hello World" in the console window by calling the [Console.WriteLine\(String\)](#) method in *Program.cs*.

```
Console.WriteLine("Hello, World!");
```

7. Replace the contents of *Program.cs* with the following code:

```
namespace HelloWorld
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

The first time you edit a `.cs` file, Visual Studio Code prompts you to add the missing assets to build and debug your app. Select **Yes**, and Visual Studio Code creates a `.vscode` folder with `launch.json` and `tasks.json` files.

NOTE

If you don't get the prompt, or if you accidentally dismiss it without selecting **Yes**, do the following steps to create `launch.json` and `tasks.json`:

- Select **Run > Add Configuration** from the menu.
- Select **.NET 5+ and .NET Core** at the **Select environment** prompt.

The code defines a class, `Program`, with a single method, `Main`, that takes a `String` array as an argument. `Main` is the application entry point, the method that's called automatically by the runtime when it launches the application. Any command-line arguments supplied when the application is launched are available in the `args` array.

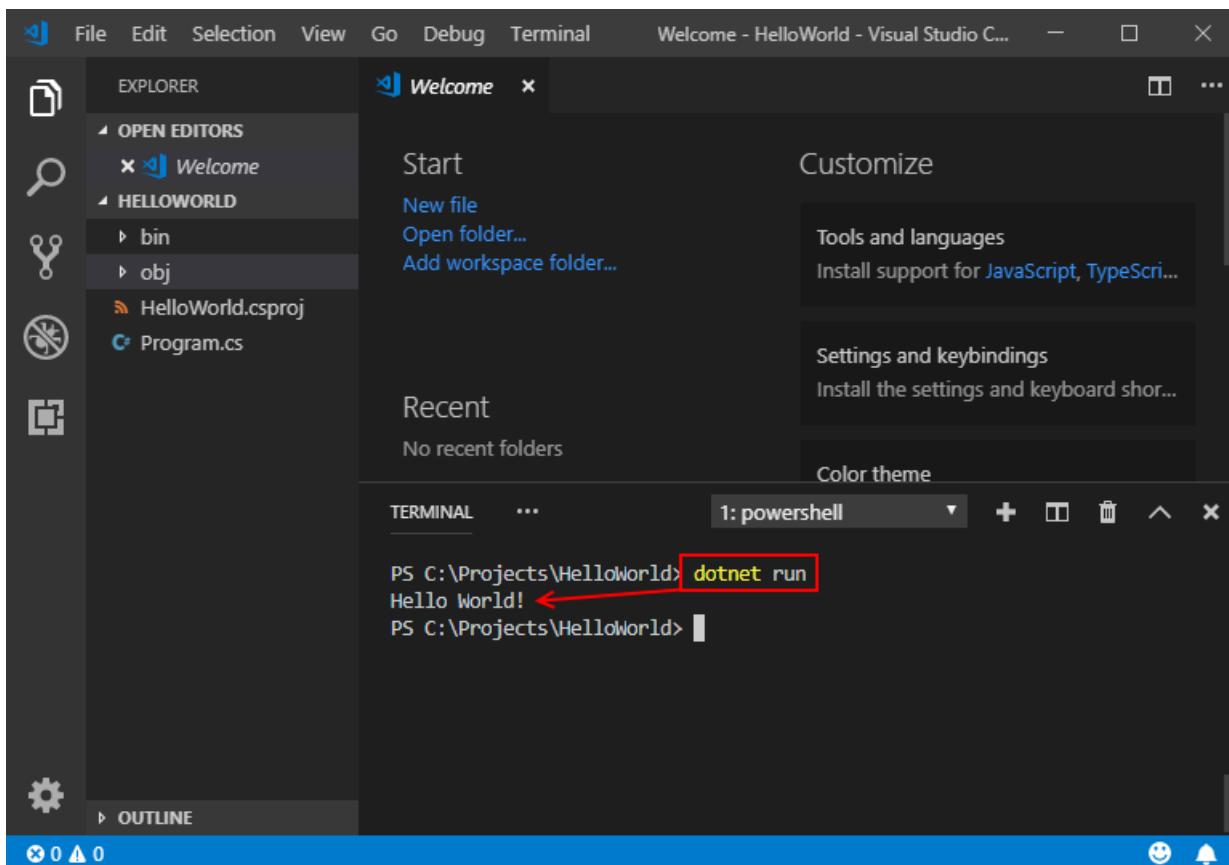
In the latest version of C#, a new feature named [top-level statements](#) lets you omit the `Program` class and the `Main` method. Most existing C# programs don't use top-level statements, so this tutorial doesn't use this new feature. But it's available in C# 10, and whether you use it in your programs is a matter of style preference.

Run the app

Run the following command in the **Terminal**:

```
dotnet run
```

The program displays "Hello World!" and ends.



Enhance the app

Enhance the application to prompt the user for their name and display it along with the date and time.

1. Open *Program.cs*.
2. Replace the contents of the `Main` method in *Program.cs*, which is the line that calls `Console.WriteLine`, with the following code:

```
Console.WriteLine("What is your name?");
var name = Console.ReadLine();
var currentDate = DateTime.Now;
Console.WriteLine($"{Environment.NewLine}Hello, {name}, on {currentDate:d} at {currentDate:t}!");
Console.WriteLine($"{Environment.NewLine}Press any key to exit...");
Console.ReadKey(true);
```

This code displays a prompt in the console window and waits until the user enters a string followed by the Enter key. It stores this string in a variable named `name`. It also retrieves the value of the `DateTime.Now` property, which contains the current local time, and assigns it to a variable named `currentDate`. And it displays these values in the console window. Finally, it displays a prompt in the console window and calls the `Console.ReadKey(Boolean)` method to wait for user input.

`NewLine` is a platform-independent and language-independent way to represent a line break. Alternatives are `\n` in C# and `vbcrlf` in Visual Basic.

The dollar sign (`$`) in front of a string lets you put expressions such as variable names in curly braces in the string. The expression value is inserted into the string in place of the expression. This syntax is referred to as [interpolated strings](#).

3. Save your changes.

IMPORTANT

In Visual Studio Code, you have to explicitly save changes. Unlike Visual Studio, file changes are not automatically saved when you build and run an app.

- Run the program again:

```
dotnet run
```

- Respond to the prompt by entering a name and pressing the Enter key.

The screenshot shows the Visual Studio Code interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Explorer:** Shows the project structure: OPEN EDITORS (Program.cs), CONSOLEAPP5 (vscode, bin, ConsoleApp5, obj, ConsoleApp5.sln, HelloWorld.csproj, Program.cs).
- Code Editor:** Displays the content of Program.cs:

```
1  using System;
2
3  namespace HelloWorld
4  {
5      class Program
6      {
7          static void Main(string[] args)
8          {
9              Console.WriteLine("What is your name?");
10             var name = Console.ReadLine();
11             var currentDate = DateTime.Now;
12             Console.WriteLine($"{Environment.NewLine}Hello, {name},
13             Console.WriteLine($"{Environment.NewLine}Press any key to exit...");
14         }
15     }
}
```
- Terminal:** Shows the command `dotnet run` being run, followed by the user input "Nancy" and the application's response "Hello, Nancy, on 4/27/2021 at 8:52 AM!".
- Status Bar:** Shows the current file is Program.cs, encoding is C#, and the terminal has 2 tabs open.

- Press any key to exit the program.

Additional resources

- [Setting up Visual Studio Code](#)

Next steps

In this tutorial, you created a .NET console application. In the next tutorial, you debug the app.

Debug a .NET console application using Visual Studio Code

This tutorial shows how to create and run a .NET console application by using Visual Studio Code and the .NET CLI. Project tasks, such as creating, compiling, and running a project are done by using the .NET CLI. You can follow this tutorial with a different code editor and run commands in a terminal if you prefer.

Prerequisites

- Visual Studio Code with the [C# extension](#) installed. For information about how to install extensions on Visual Studio Code, see [VS Code Extension Marketplace](#).
- The [.NET 5 SDK](#). If you install the .NET 6 SDK, install the .NET 5 SDK also, or some of the tutorial instructions

won't work. For more information, see [New C# templates generate top-level statements](#).

Create the app

Create a .NET console app project named "HelloWorld".

1. Start Visual Studio Code.
2. Select **File > Open Folder** (**File > Open...** on macOS) from the main menu.
3. In the **Open Folder** dialog, create a *HelloWorld* folder and click **Select Folder** (Open on macOS).

The folder name becomes the project name and the namespace name by default. You'll add code later in the tutorial that assumes the project namespace is `HelloWorld`.

4. Open the **Terminal** in Visual Studio Code by selecting **View > Terminal** from the main menu.

The **Terminal** opens with the command prompt in the *HelloWorld* folder.

5. In the **Terminal**, enter the following command:

```
dotnet new console --framework net5.0
```

The template creates a simple "Hello World" application. It calls the `Console.WriteLine(String)` method to display "Hello World!" in the console window.

The template code defines a class, `Program`, with a single method, `Main`, that takes a `String` array as an argument:

```
using System;

namespace HelloWorld
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

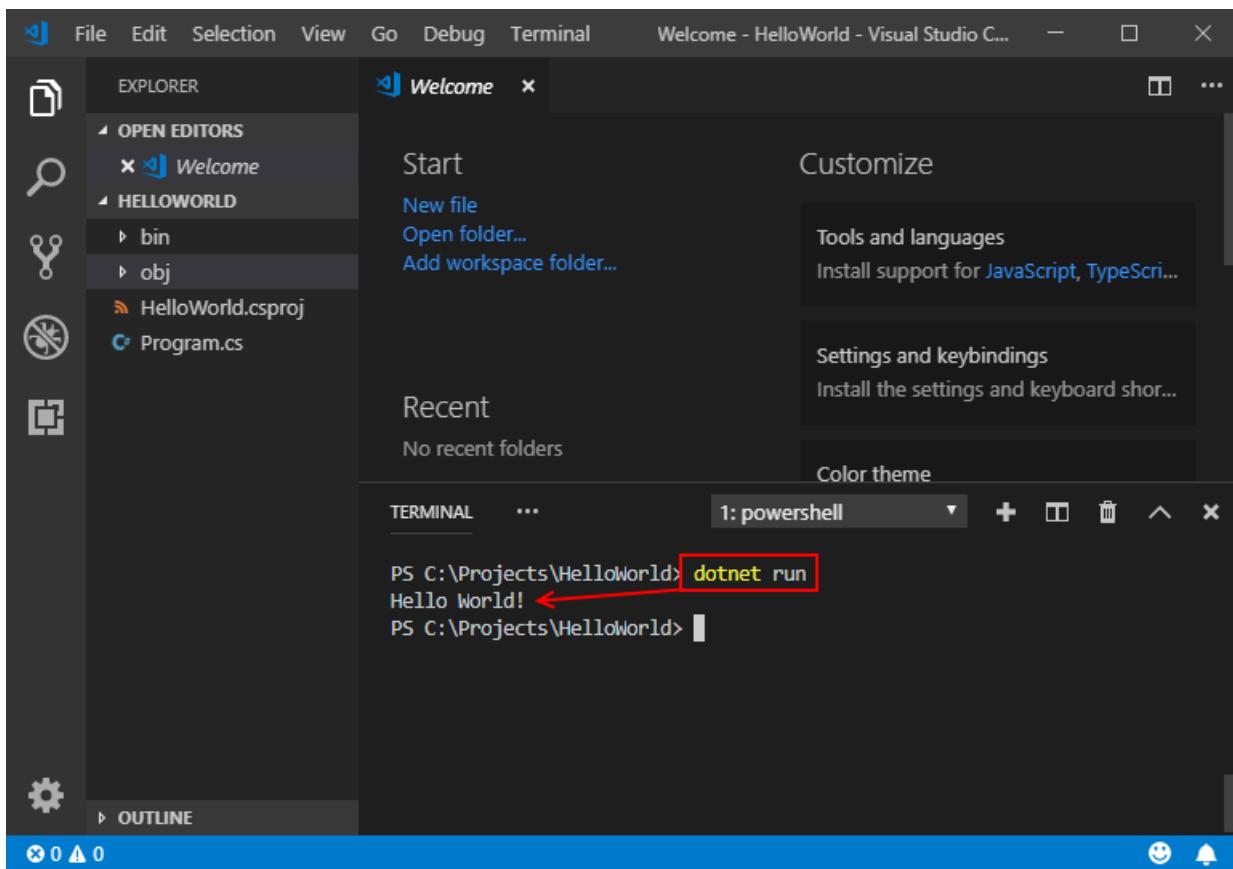
`Main` is the application entry point, the method that's called automatically by the runtime when it launches the application. Any command-line arguments supplied when the application is launched are available in the `args` array.

Run the app

Run the following command in the **Terminal**:

```
dotnet run
```

The program displays "Hello World!" and ends.

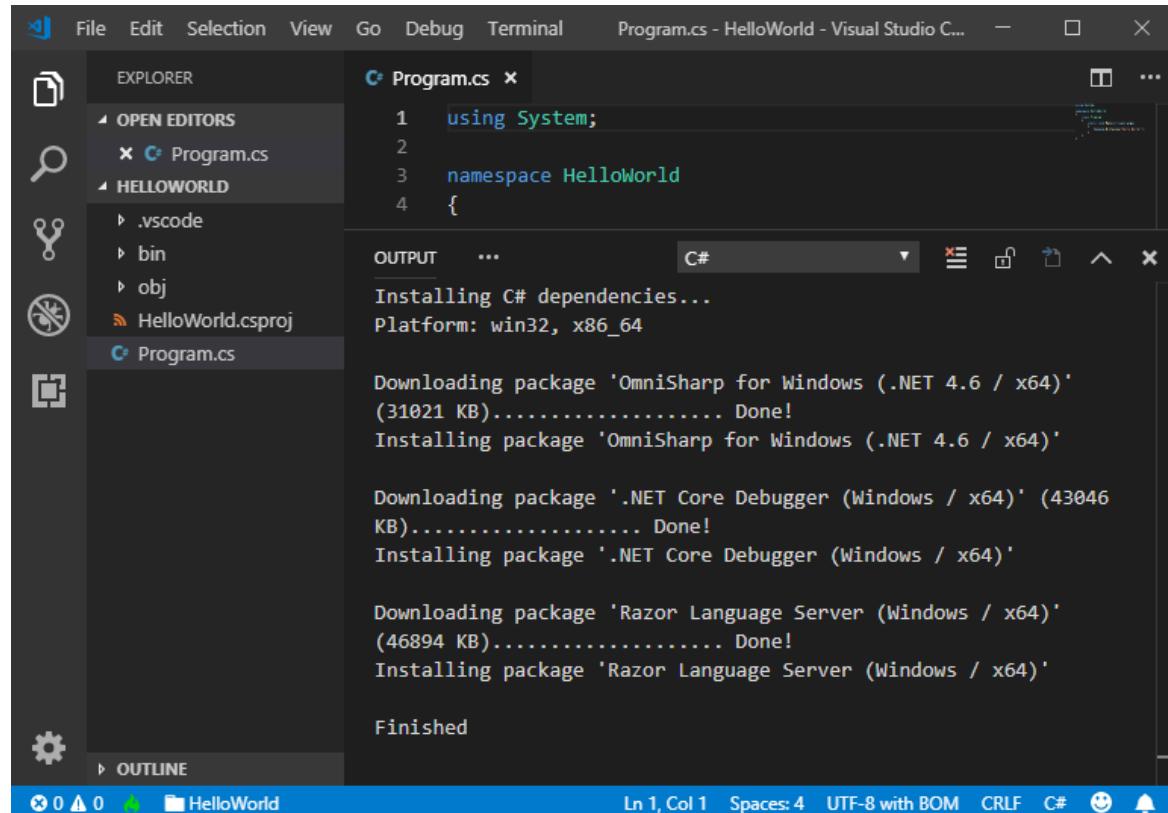


Enhance the app

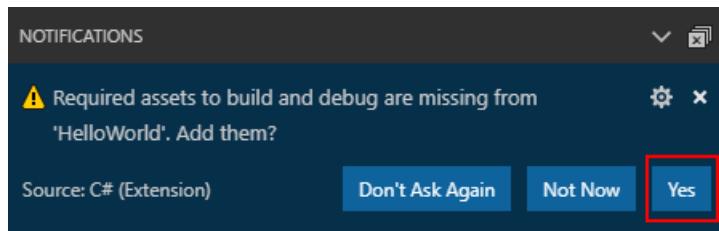
Enhance the application to prompt the user for their name and display it along with the date and time.

1. Open *Program.cs* by clicking on it.

The first time you open a C# file in Visual Studio Code, [OmniSharp](#) loads in the editor.



2. Select Yes when Visual Studio Code prompts you to add the missing assets to build and debug your app.



- Replace the contents of the `Main` method in `Program.cs`, which is the line that calls `Console.WriteLine`, with the following code:

```
Console.WriteLine("What is your name?");
var name = Console.ReadLine();
var currentDate = DateTime.Now;
Console.WriteLine($"{Environment.NewLine}Hello, {name}, on {currentDate:d} at {currentDate:t}!");
Console.WriteLine($"{Environment.NewLine}Press any key to exit...");
Console.ReadKey(true);
```

This code displays a prompt in the console window and waits until the user enters a string followed by the Enter key. It stores this string in a variable named `name`. It also retrieves the value of the `DateTime.Now` property, which contains the current local time, and assigns it to a variable named `currentDate`. And it displays these values in the console window. Finally, it displays a prompt in the console window and calls the `Console.ReadKey(Boolean)` method to wait for user input.

`NewLine` is a platform-independent and language-independent way to represent a line break. Alternatives are `\n` in C# and `vbcrlf` in Visual Basic.

The dollar sign (`$`) in front of a string lets you put expressions such as variable names in curly braces in the string. The expression value is inserted into the string in place of the expression. This syntax is referred to as [interpolated strings](#).

- Save your changes.

IMPORTANT

In Visual Studio Code, you have to explicitly save changes. Unlike Visual Studio, file changes are not automatically saved when you build and run an app.

- Run the program again:

```
dotnet run
```

- Respond to the prompt by entering a name and pressing the Enter key.

The screenshot shows the Visual Studio Code interface with the following details:

- File Menu:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Title Bar:** Program.cs - ConsoleApp5 - Visual Studio Code.
- Sidebar (Left):** Explorer, Open Editors (Program.cs selected), Console Apps (ConsoleApp5 selected), and Solution Explorer (ConsoleApp5.sln selected).
- Editor Area:** Displays the contents of Program.cs:

```
1  using System;
2
3  namespace HelloWorld
4  {
5      // 0 references
6      class Program
7      {
8          // 0 references
9          static void Main(string[] args)
10         {
11             Console.WriteLine("What is your name?");
12             var name = Console.ReadLine();
13             var currentDate = DateTime.Now;
14             Console.WriteLine($"{Environment.NewLine}Hello, {name},");
15             Console.WriteLine($"{Environment.NewLine}Press any key to exit...");
16             Console.ReadKey(true);
17         }
18     }
19 }
```
- Terminal:** Shows the command `dotnet run` being executed in the directory `C:\Users\thruk\source\repos\ConsoleApp5>`. The output shows the application prompting for a name ("What is your name?") and receiving "Nancy". It then outputs "Hello, Nancy, on 4/27/2021 at 8:52 AM!" followed by a prompt to press any key to exit.
- Bottom Status Bar:** Shows the status bar with icons for file operations, .NET Core Launch (console) (ConsoleApp5), Live Share, and file paths (ConsoleApp5.sln, csharp, Program.cs). It also shows CRLF, C#, and a bell icon.

7. Press any key to exit the program.

Additional resources

- Setting up Visual Studio Code

Next steps

In this tutorial, you created a .NET console application. In the next tutorial, you debug the app.

Debug a .NET console application using Visual Studio Code

This tutorial is only available for .NET 5 and .NET 6. Select one of those options at the top of the page.

Tutorial: Debug a .NET console application using Visual Studio Code

9/20/2022 • 13 minutes to read • [Edit Online](#)

This tutorial introduces the debugging tools available in Visual Studio Code for working with .NET apps.

Prerequisites

- This tutorial works with the console app that you create in [Create a .NET console application using Visual Studio Code](#).

Use Debug build configuration

Debug and *Release* are .NET's built-in build configurations. You use the Debug build configuration for debugging and the Release configuration for the final release distribution.

In the Debug configuration, a program compiles with full symbolic debug information and no optimization. Optimization complicates debugging, because the relationship between source code and generated instructions is more complex. The release configuration of a program has no symbolic debug information and is fully optimized.

By default, Visual Studio Code launch settings use the Debug build configuration, so you don't need to change it before debugging.

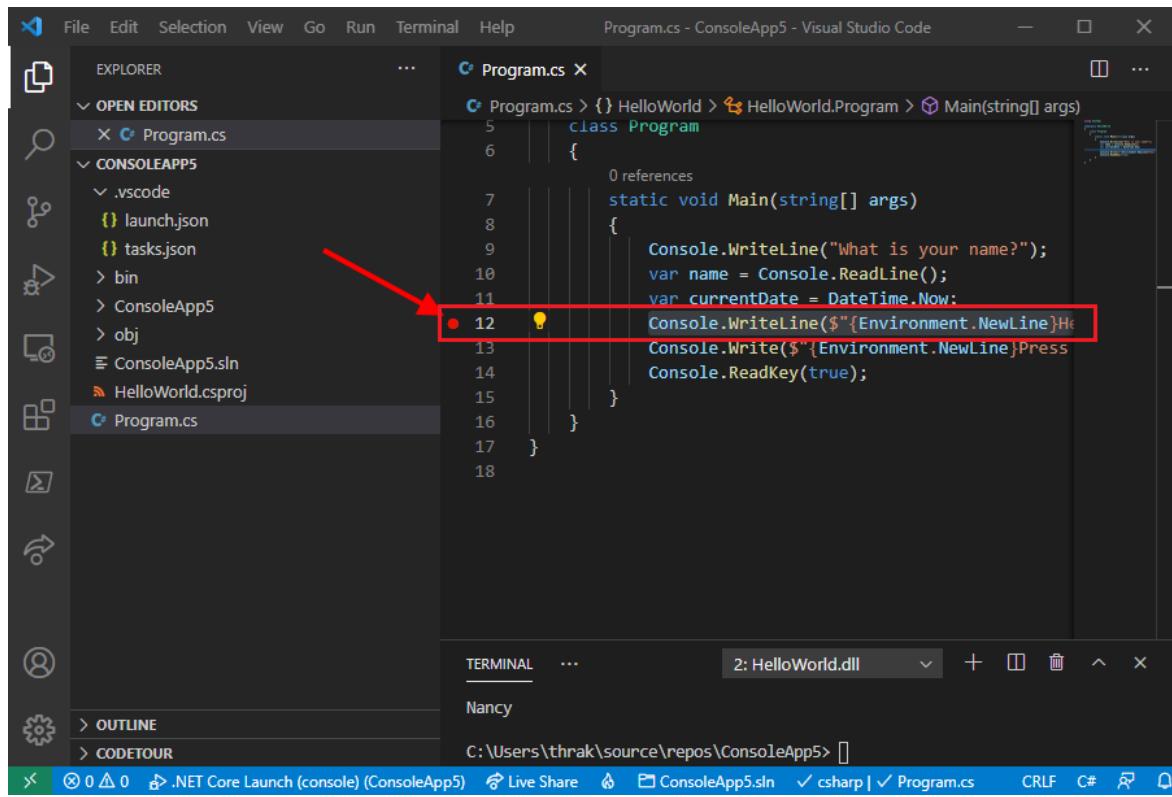
1. Start Visual Studio Code.
2. Open the folder of the project that you created in [Create a .NET console application using Visual Studio Code](#).

Set a breakpoint

A *breakpoint* temporarily interrupts the execution of the application before the line with the breakpoint is run.

1. Open the *Program.cs* file.
2. Set a *breakpoint* on the line that displays the name, date, and time, by clicking in the left margin of the code window. The left margin is to the left of the line numbers. Other ways to set a breakpoint are by pressing F9 or choosing **Run > Toggle Breakpoint** from the menu while the line of code is selected.

Visual Studio Code indicates the line on which the breakpoint is set by displaying a red dot in the left margin.



Set up for terminal input

The breakpoint is located after a `Console.ReadLine()` method call. The **Debug Console** doesn't accept terminal input for a running program. To handle terminal input while debugging, you can use the integrated terminal (one of the Visual Studio Code windows) or an external terminal. For this tutorial, you use the integrated terminal.

1. Open `.vscode/launch.json`.
2. Change the `console` setting from `internalConsole` to `integratedTerminal`:

```
"console": "integratedTerminal",
```

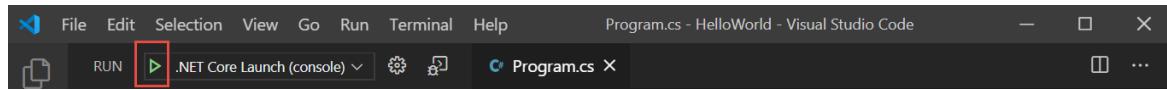
3. Save your changes.

Start debugging

1. Open the Debug view by selecting the Debugging icon on the left side menu.

A screenshot of Visual Studio Code showing the Debug sidebar open. The sidebar includes sections for VARIABLES, WATCH, CALL STACK, and BREAKPOINTS. A red box highlights the green arrow icon at the top of the sidebar, next to the ".NET Core Launch (console)" label. The main editor area shows a C# file named Program.cs with code for a HelloWorld application. The terminal at the bottom shows the path "C:\Users\thruk\source\repos\ConsoleApp5>".

2. Select the green arrow at the top of the pane, next to .NET Core Launch (console). Other ways to start the program in debugging mode are by pressing F5 or choosing Run > Start Debugging from the menu.



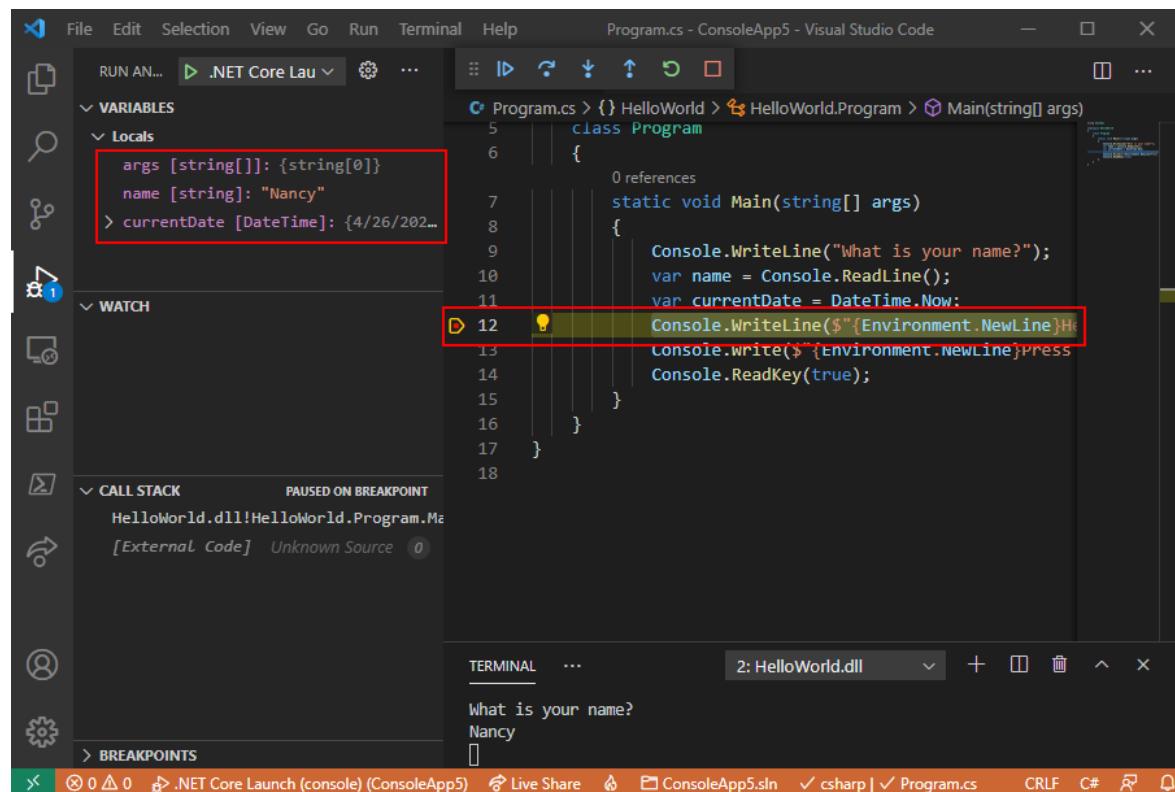
3. Select the Terminal tab to see the "What is your name?" prompt that the program displays before waiting for a response.

A screenshot of Visual Studio Code showing the Terminal tab selected in the bottom navigation bar. The main editor area shows the C# file (Program.cs) with the breakpoint at line 12. The terminal window at the bottom displays the text "What is your name?".

4. Enter a string in the Terminal window in response to the prompt for a name, and then press Enter.

Program execution stops when it reaches the breakpoint and before the `Console.WriteLine` method runs.

The **Locals** section of the **Variables** window displays the values of variables that are defined in the currently running method.



Use the Debug Console

The **Debug Console** window lets you interact with the application you're debugging. You can change the value of variables to see how it affects your program.

1. Select the **Debug Console** tab.
2. Enter `name = "Gracie"` at the prompt at the bottom of the **Debug Console** window and press the Enter key.

A screenshot of Visual Studio Code showing the Debug Console tab selected. The code editor shows the same C# code as before. The terminal window shows the output: "What is your name? Nancy". A red box highlights the `DEBUG CONSOLE` tab. The bottom of the terminal window shows the prompt: "name = "Gracie"".

3. Enter `currentDate = DateTime.Parse("2019-11-16T17:25:00Z").ToUniversalTime()` at the bottom of the **Debug Console** window and press the Enter key.

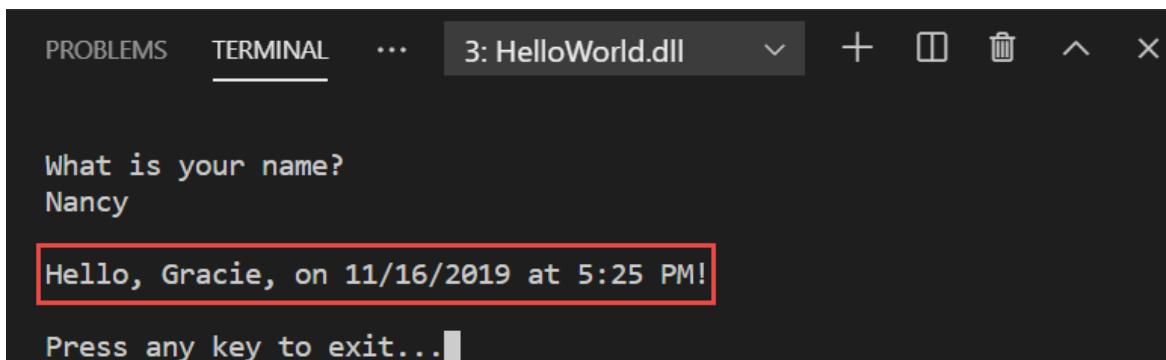
The **Variables** window displays the new values of the `name` and `currentDate` variables.

4. Continue program execution by selecting the **Continue** button in the toolbar. Another way to continue is by pressing F5.



5. Select the **Terminal** tab again.

The values displayed in the console window correspond to the changes you made in the **Debug Console**.

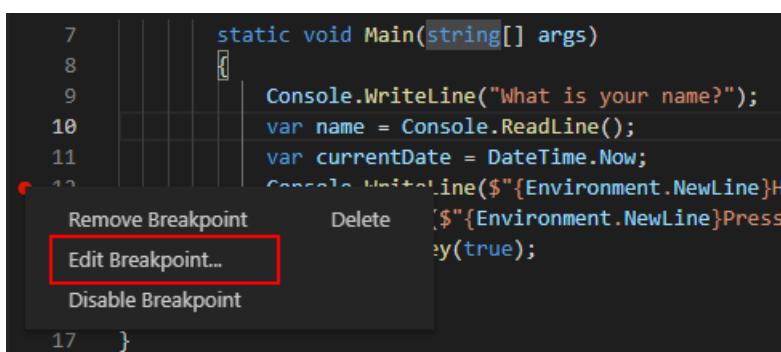


6. Press any key to exit the application and stop debugging.

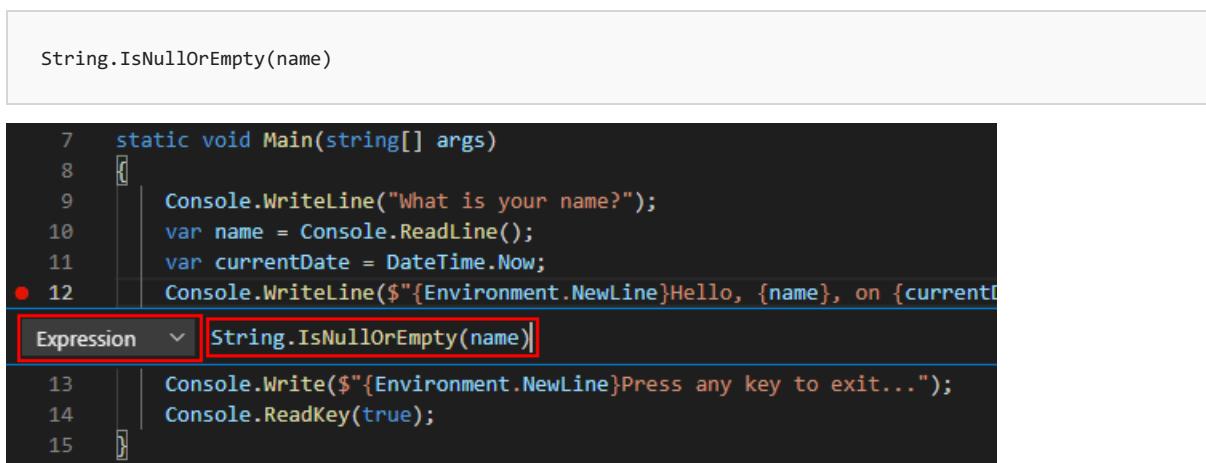
Set a conditional breakpoint

The program displays the string that the user enters. What happens if the user doesn't enter anything? You can test this with a useful debugging feature called a *conditional breakpoint*.

1. Right-click (Ctrl-click on macOS) on the red dot that represents the breakpoint. In the context menu, select **Edit Breakpoint** to open a dialog that lets you enter a conditional expression.



2. Select **Expression** in the drop-down, enter the following conditional expression, and press Enter.



Each time the breakpoint is hit, the debugger calls the `String.IsNullOrEmpty(name)` method, and it breaks

on this line only if the method call returns `true`.

Instead of a conditional expression, you can specify a *hit count*, which interrupts program execution before a statement is run a specified number of times. Another option is to specify a *filter condition*, which interrupts program execution based on such attributes as a thread identifier, process name, or thread name.

3. Start the program with debugging by pressing F5.
4. In the **Terminal** tab, press the Enter key when prompted to enter your name.

Because the condition you specified (`name` is either `null` or `String.Empty`) has been satisfied, program execution stops when it reaches the breakpoint and before the `Console.WriteLine` method runs.

The **Variables** window shows that the value of the `name` variable is `""`, or `String.Empty`.

5. Confirm the value is an empty string by entering the following statement at the **Debug Console** prompt and pressing Enter. The result is `true`.

```
name == String.Empty
```

6. Select the **Continue** button on the toolbar to continue program execution.
7. Select the **Terminal** tab, and press any key to exit the program and stop debugging.
8. Clear the breakpoint by clicking on the dot in the left margin of the code window. Other ways to clear a breakpoint are by pressing F9 or choosing **Run > Toggle Breakpoint** from the menu while the line of code is selected.
9. If you get a warning that the breakpoint condition will be lost, select **Remove Breakpoint**.

Step through a program

Visual Studio Code also allows you to step line by line through a program and monitor its execution. Ordinarily, you'd set a breakpoint and follow program flow through a small part of your program code. Since this program is small, you can step through the entire program.

1. Set a breakpoint on the opening curly brace of the `Main` method.
2. Press F5 to start debugging.

Visual Studio Code highlights the breakpoint line.

At this point, the **Variables** window shows that the `args` array is empty, and `name` and `currentDate` have default values.

3. Select **Run > Step Into** or press F11.



Visual Studio Code highlights the next line.

4. Select **Run > Step Into** or press F11.

Visual Studio Code runs the `Console.WriteLine` for the name prompt and highlights the next line of execution. The next line is the `Console.ReadLine` for the `name`. The **Variables** window is unchanged, and the **Terminal** tab shows the "What is your name?" prompt.

5. Select **Run > Step Into** or press F11.

Visual Studio highlights the `name` variable assignment. The **Variables** window shows that `name` is still `null`.

6. Respond to the prompt by entering a string in the Terminal tab and pressing Enter.

The **Terminal** tab might not display the string you enter while you're entering it, but the `Console.ReadLine` method will capture your input.

7. Select **Run > Step Into** or press F11.

Visual Studio Code highlights the `currentDate` variable assignment. The **Variables** window shows the value returned by the call to the `Console.ReadLine` method. The **Terminal** tab displays the string you entered at the prompt.

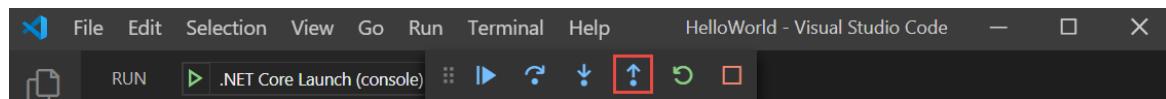
8. Select **Run > Step Into** or press F11.

The **Variables** window shows the value of the `currentDate` variable after the assignment from the `DateTime.Now` property.

9. Select **Run > Step Into** or press F11.

Visual Studio Code calls the `Console.WriteLine(String, Object, Object)` method. The console window displays the formatted string.

10. Select **Run > Step Out** or press Shift+F11.



11. Select the **Terminal** tab.

The terminal displays "Press any key to exit..."

12. Press any key to exit the program.

Use Release build configuration

Once you've tested the Debug version of your application, you should also compile and test the Release version. The Release version incorporates compiler optimizations that can affect the behavior of an application. For example, compiler optimizations that are designed to improve performance can create race conditions in multithreaded applications.

To build and test the Release version of your console application, open the **Terminal** and run the following command:

```
dotnet run --configuration Release
```

Additional resources

- [Debugging in Visual Studio Code](#)

Next steps

In this tutorial, you used Visual Studio Code debugging tools. In the next tutorial, you publish a deployable version of the app.

Publish a .NET console application using Visual Studio Code

This tutorial introduces the debugging tools available in Visual Studio Code for working with .NET apps.

Prerequisites

- This tutorial works with the console app that you create in [Create a .NET console application using Visual Studio Code](#).

Use Debug build configuration

Debug and *Release* are .NET Core's built-in build configurations. You use the Debug build configuration for debugging and the Release configuration for the final release distribution.

In the Debug configuration, a program compiles with full symbolic debug information and no optimization. Optimization complicates debugging, because the relationship between source code and generated instructions is more complex. The release configuration of a program has no symbolic debug information and is fully optimized.

By default, Visual Studio Code launch settings use the Debug build configuration, so you don't need to change it before debugging.

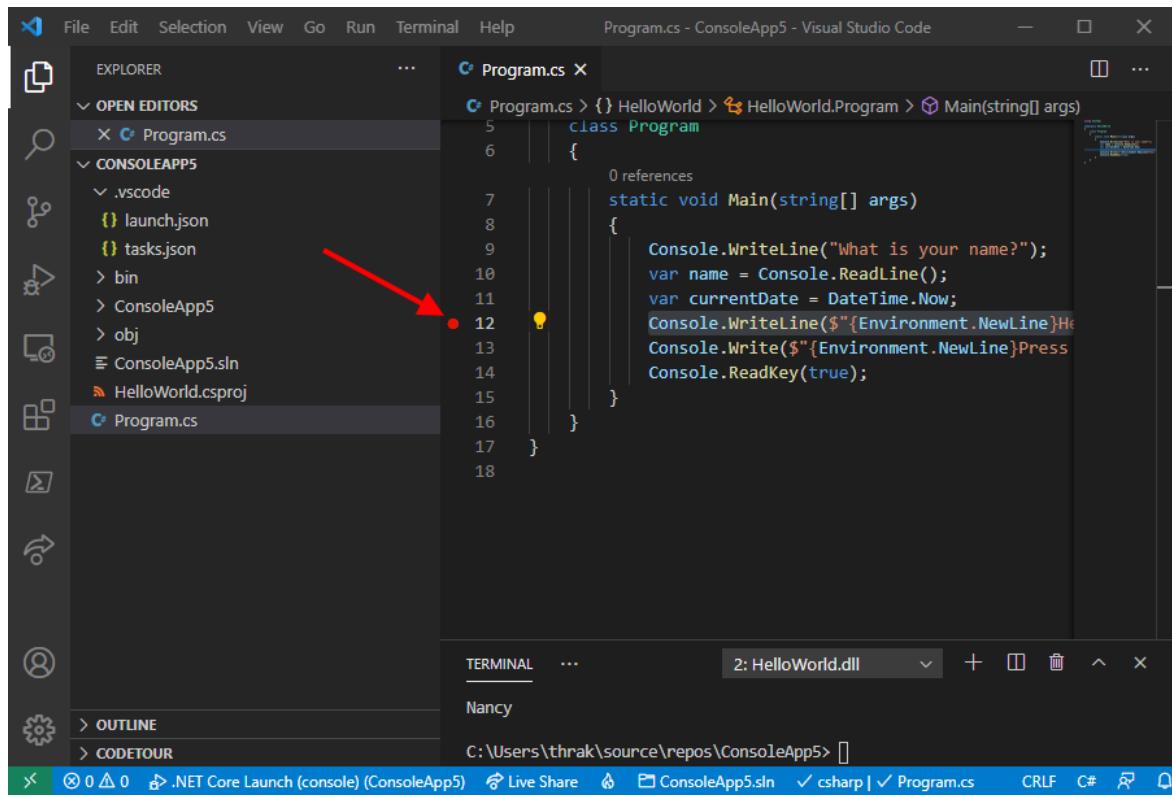
1. Start Visual Studio Code.
2. Open the folder of the project that you created in [Create a .NET console application using Visual Studio Code](#).

Set a breakpoint

A *breakpoint* temporarily interrupts the execution of the application before the line with the breakpoint is executed.

1. Open the *Program.cs* file.
2. Set a *breakpoint* on the line that displays the name, date, and time, by clicking in the left margin of the code window. The left margin is to the left of the line numbers. Other ways to set a breakpoint are by pressing F9 or choosing **Run > Toggle Breakpoint** from the menu while the line of code is selected.

Visual Studio Code indicates the line on which the breakpoint is set by displaying a red dot in the left margin.



Set up for terminal input

The breakpoint is located after a `Console.ReadLine()` method call. The **Debug** Console doesn't accept terminal input for a running program. To handle terminal input while debugging, you can use the integrated terminal (one of the Visual Studio Code windows) or an external terminal. For this tutorial, you use the integrated terminal.

1. Open `.vscode/launch.json`.
2. Change the `console` setting from `internalConsole` to `integratedTerminal`:

```
"console": "integratedTerminal",
```

3. Save your changes.

Start debugging

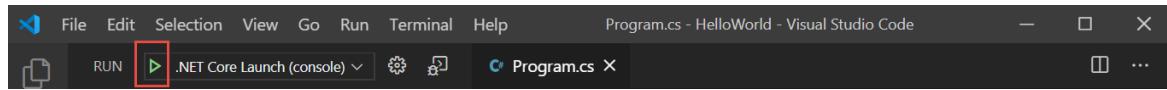
1. Open the Debug view by selecting the Debugging icon on the left side menu.

A screenshot of Visual Studio Code showing the Debug sidebar open. The sidebar includes sections for VARIABLES, WATCH, CALL STACK, and BREAKPOINTS. A red box highlights the green arrow icon at the top of the sidebar, next to the ".NET Core Launch (console)" button. The code editor shows a C# file named Program.cs with the following code:

```
1 using System;
2
3 namespace HelloWorld
4 {
5     0 references
6     class Program
7     {
8         0 references
9         static void Main(string[] args)
10        {
11            Console.WriteLine("What is your name?");
12            var name = Console.ReadLine();
13            var currentDate = DateTime.Now;
14            Console.WriteLine($"{Environment.NewLine}Hello, {name},");
15            Console.Write($"{Environment.NewLine}Press any key to exit...");
16            Console.ReadKey(true);
17        }
18    }
19 }
```

The terminal tab shows the path "C:\Users\thruk\source\repos\ConsoleApp5>".

2. Select the green arrow at the top of the pane, next to **.NET Core Launch (console)**. Other ways to start the program in debugging mode are by pressing F5 or choosing **Run > Start Debugging** from the menu.

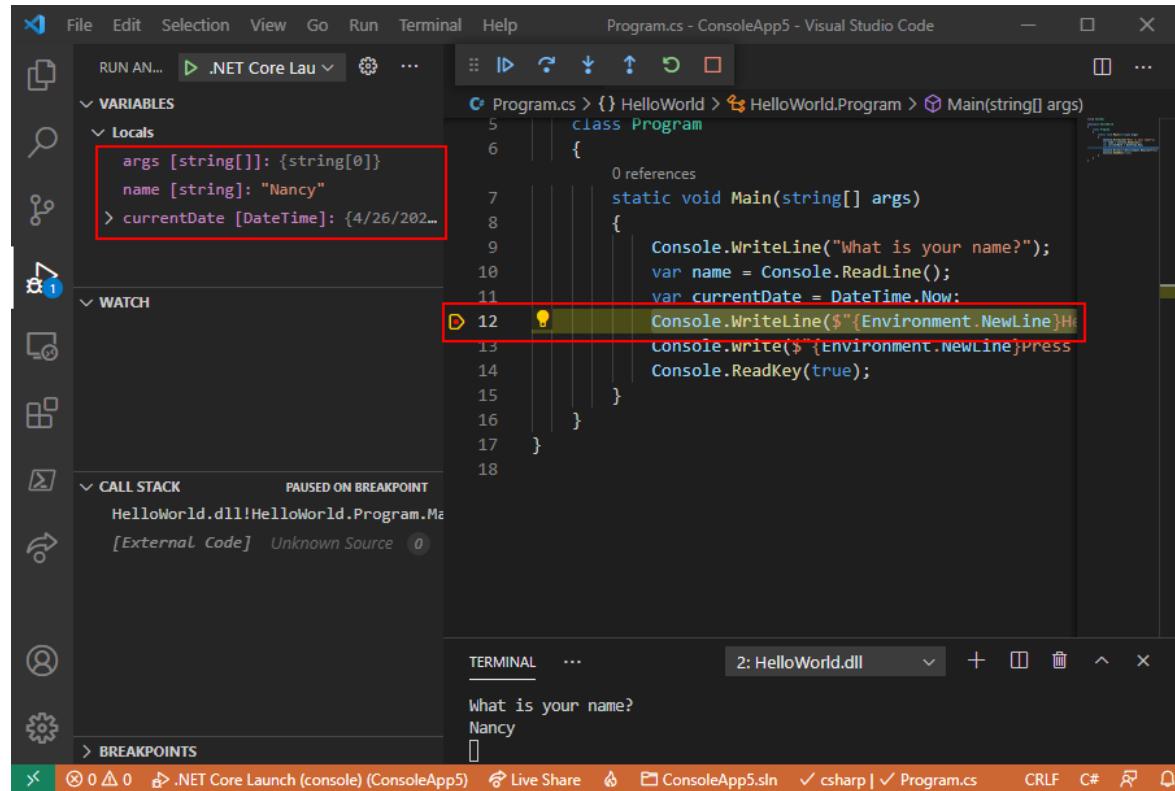


3. Select the **Terminal** tab to see the "What is your name?" prompt that the program displays before waiting for a response.

A screenshot of Visual Studio Code showing the program running. The terminal tab is active, displaying the message "What is your name?". The code editor shows the same C# code as the previous screenshot. The status bar at the bottom indicates "RUNNING" and shows the path "C:\Users\thruk\source\repos\ConsoleApp5>".

4. Enter a string in the **Terminal** window in response to the prompt for a name, and then press Enter.

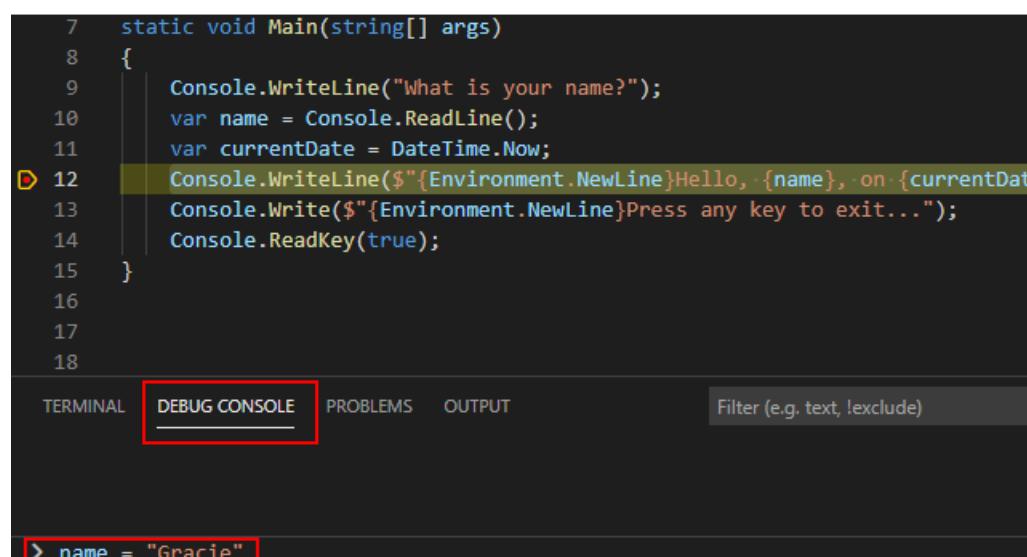
Program execution stops when it reaches the breakpoint and before the `Console.WriteLine` method executes. The **Locals** section of the **Variables** window displays the values of variables that are defined in the currently executing method.



Use the Debug Console

The **Debug Console** window lets you interact with the application you're debugging. You can change the value of variables to see how it affects your program.

1. Select the **Debug Console** tab.
2. Enter `name = "Gracie"` at the prompt at the bottom of the **Debug Console** window and press the Enter key.



3. Enter `currentDate = DateTime.Parse("2019-11-16T17:25:00Z").ToUniversalTime()` at the bottom of the **Debug Console** window and press the Enter key.

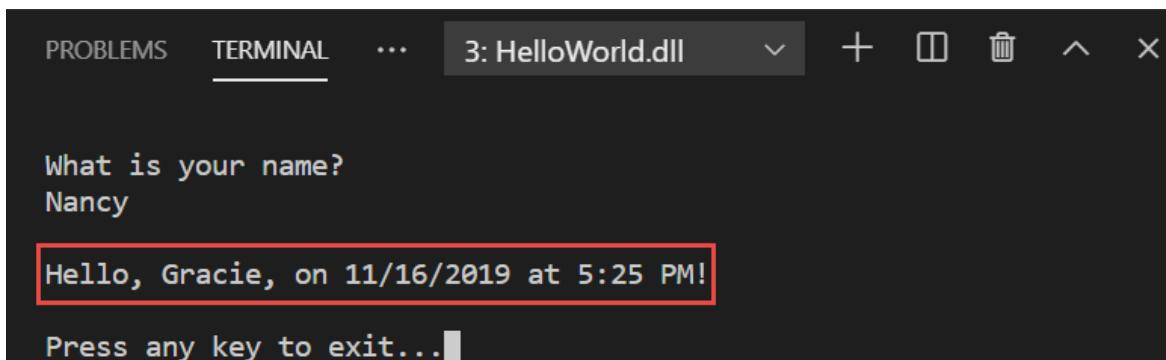
The **Variables** window displays the new values of the `name` and `currentDate` variables.

4. Continue program execution by selecting the **Continue** button in the toolbar. Another way to continue is by pressing F5.



5. Select the **Terminal** tab again.

The values displayed in the console window correspond to the changes you made in the **Debug Console**.

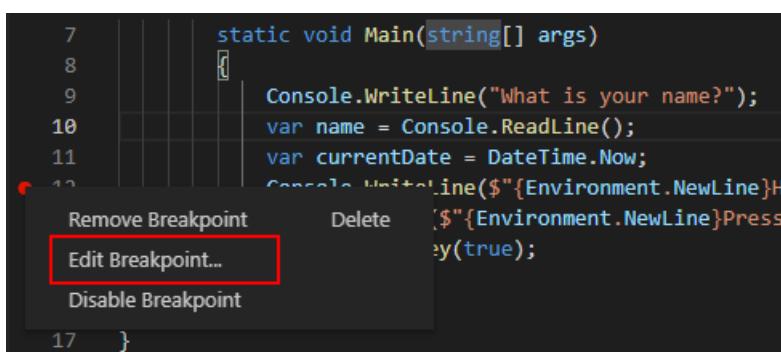


6. Press any key to exit the application and stop debugging.

Set a conditional breakpoint

The program displays the string that the user enters. What happens if the user doesn't enter anything? You can test this with a useful debugging feature called a *conditional breakpoint*.

1. Right-click (Ctrl-click on macOS) on the red dot that represents the breakpoint. In the context menu, select **Edit Breakpoint** to open a dialog that lets you enter a conditional expression.



2. Select **Expression** in the drop-down, enter the following conditional expression, and press Enter.



Each time the breakpoint is hit, the debugger calls the `String.IsNullOrEmpty(name)` method, and it breaks

on this line only if the method call returns `true`.

Instead of a conditional expression, you can specify a *hit count*, which interrupts program execution before a statement is executed a specified number of times. Another option is to specify a *filter condition*, which interrupts program execution based on such attributes as a thread identifier, process name, or thread name.

3. Start the program with debugging by pressing F5.
4. In the **Terminal** tab, press the Enter key when prompted to enter your name.

Because the condition you specified (`name` is either `null` or `String.Empty`) has been satisfied, program execution stops when it reaches the breakpoint and before the `Console.WriteLine` method executes.

The **Variables** window shows that the value of the `name` variable is `""`, or `String.Empty`.

5. Confirm the value is an empty string by entering the following statement at the **Debug Console** prompt and pressing Enter. The result is `true`.

A screenshot of the Visual Studio Code interface. The code editor shows a C# file with the following code:

```
7 static void Main(string[] args)
8 {
9     Console.WriteLine("What is your name?");
10    var name = Console.ReadLine();
11    var currentDate = DateTime.Now;
12    Console.WriteLine($"{Environment.NewLine}Hello, {name}, on {currentDate}");
13    if (String.IsNullOrEmpty(name))
14        Console.WriteLine($"{Environment.NewLine}Press any key to exit..");
15    Console.ReadKey(true);
16 }
```

The line `12` is highlighted with a yellow background, indicating it is the current line of execution. The **DEBUG CONSOLE** tab is selected. The output pane shows the following:

```
→ name == String.Empty
true
```

The output `name == String.Empty` is highlighted with a red box.

6. Select the **Continue** button on the toolbar to continue program execution.
7. Select the **Terminal** tab, and press any key to exit the program and stop debugging.
8. Clear the breakpoint by clicking on the dot in the left margin of the code window. Other ways to clear a breakpoint are by pressing F9 or choosing **Run > Toggle Breakpoint** from the menu while the line of code is selected.
9. If you get a warning that the breakpoint condition will be lost, select **Remove Breakpoint**.

Step through a program

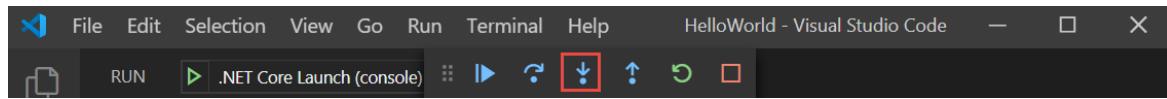
Visual Studio Code also allows you to step line by line through a program and monitor its execution. Ordinarily, you'd set a breakpoint and follow program flow through a small part of your program code. Since this program is small, you can step through the entire program.

1. Set a breakpoint on the opening curly brace of the `Main` method.
2. Press F5 to start debugging.

Visual Studio Code highlights the breakpoint line.

At this point, the **Variables** window shows that the `args` array is empty, and `name` and `currentDate` have default values.

3. Select **Run > Step Into** or press F11.



Visual Studio Code highlights the next line.

4. Select **Run > Step Into** or press F11.

Visual Studio Code executes the `Console.WriteLine` for the name prompt and highlights the next line of execution. The next line is the `Console.ReadLine` for the `name`. The **Variables** window is unchanged, and the **Terminal** tab shows the "What is your name?" prompt.

5. Select **Run > Step Into** or press F11.

Visual Studio highlights the `name` variable assignment. The **Variables** window shows that `name` is still `null`.

6. Respond to the prompt by entering a string in the Terminal tab and pressing Enter.

The **Terminal** tab might not display the string you enter while you're entering it, but the `Console.ReadLine` method will capture your input.

7. Select **Run > Step Into** or press F11.

Visual Studio Code highlights the `currentDate` variable assignment. The **Variables** window shows the value returned by the call to the `Console.ReadLine` method. The **Terminal** tab displays the string you entered at the prompt.

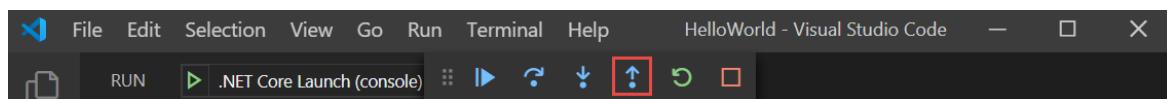
8. Select **Run > Step Into** or press F11.

The **Variables** window shows the value of the `currentDate` variable after the assignment from the `DateTime.Now` property.

9. Select **Run > Step Into** or press F11.

Visual Studio Code calls the `Console.WriteLine(String, Object, Object)` method. The console window displays the formatted string.

10. Select **Run > Step Out** or press Shift+F11.



11. Select the **Terminal** tab.

The terminal displays "Press any key to exit..."

12. Press any key to exit the program.

Use Release build configuration

Once you've tested the Debug version of your application, you should also compile and test the Release version. The Release version incorporates compiler optimizations that can affect the behavior of an application. For example, compiler optimizations that are designed to improve performance can create race conditions in multithreaded applications.

To build and test the Release version of your console application, open the **Terminal** and run the following command:

```
dotnet run --configuration Release
```

Additional resources

- [Debugging in Visual Studio Code](#)

Next steps

In this tutorial, you used Visual Studio Code debugging tools. In the next tutorial, you publish a deployable version of the app.

[Publish a .NET console application using Visual Studio Code](#)

This tutorial is only available for .NET 5 and .NET 6. Select one of those options at the top of the page.

Tutorial: Publish a .NET console application using Visual Studio Code

9/20/2022 • 6 minutes to read • [Edit Online](#)

This tutorial shows how to publish a console app so that other users can run it. Publishing creates the set of files that are needed to run an application. To deploy the files, copy them to the target machine.

The .NET CLI is used to publish the app, so you can follow this tutorial with a code editor other than Visual Studio Code if you prefer.

Prerequisites

- This tutorial works with the console app that you create in [Create a .NET console application using Visual Studio Code](#).

Publish the app

1. Start Visual Studio Code.
2. Open the *HelloWorld* project folder that you created in [Create a .NET console application using Visual Studio Code](#).
3. Choose **View > Terminal** from the main menu.

The terminal opens in the *HelloWorld* folder.

4. Run the following command:

```
dotnet publish --configuration Release
```

The default build configuration is *Debug*, so this command specifies the *Release* build configuration. The output from the *Release* build configuration has minimal symbolic debug information and is fully optimized.

The command output is similar to the following example:

```
Microsoft (R) Build Engine version 16.7.0+b89cb5fde for .NET
Copyright (C) Microsoft Corporation. All rights reserved.

Determining projects to restore...
All projects are up-to-date for restore.
HelloWorld -> C:\Projects\HelloWorld\bin\Release\net6.0\HelloWorld.dll
HelloWorld -> C:\Projects\HelloWorld\bin\Release\net6.0\publish\
```

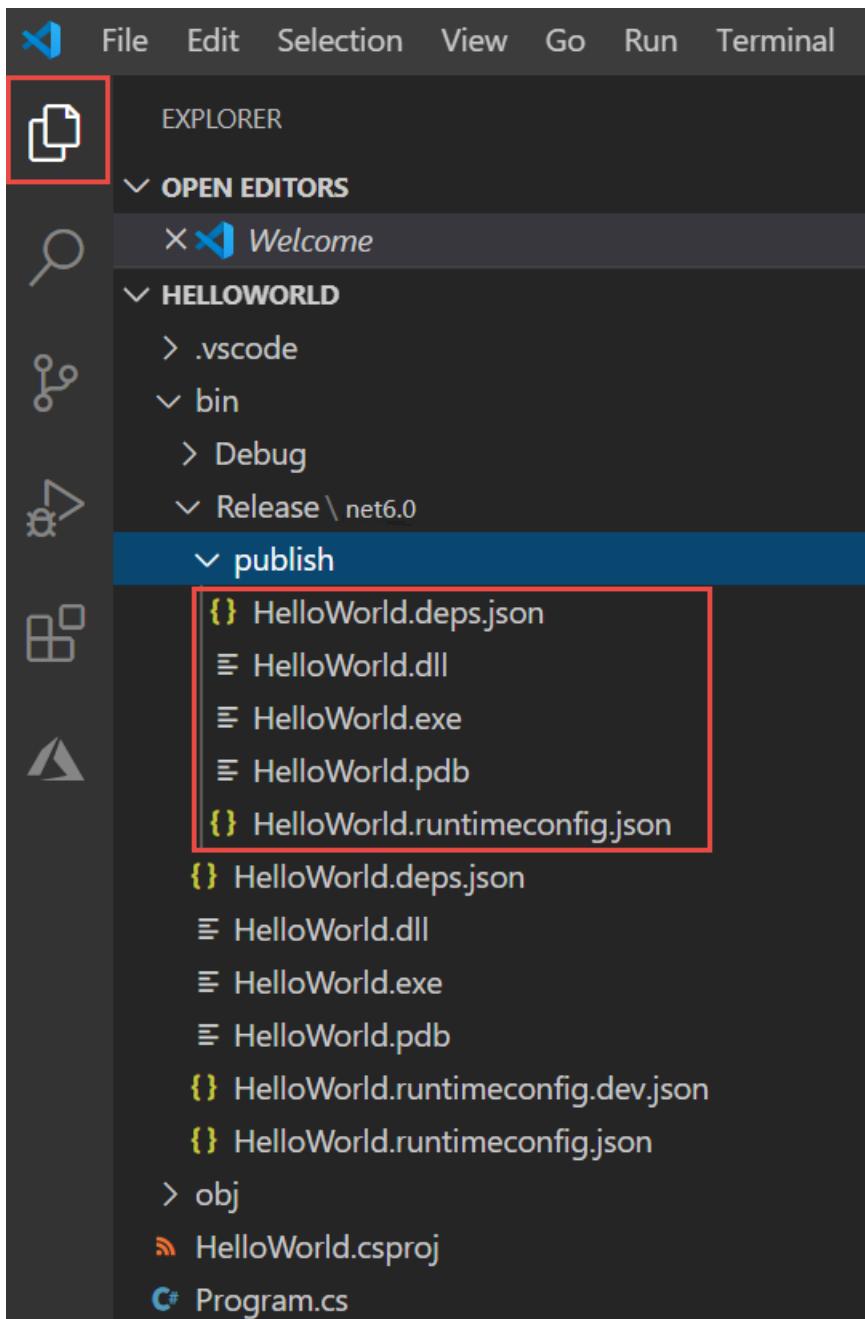
Inspect the files

By default, the publishing process creates a framework-dependent deployment, which is a type of deployment where the published application runs on a machine that has the .NET runtime installed. To run the published app you can use the executable file or run the `dotnet HelloWorld.dll` command from a command prompt.

In the following steps, you'll look at the files created by the publish process.

1. Select the **Explorer** in the left navigation bar.

2. Expand `bin/Release/net6.0/publish`.



As the image shows, the published output includes the following files:

- `HelloWorld.deps.json`

This is the application's runtime dependencies file. It defines the .NET components and the libraries (including the dynamic link library that contains your application) needed to run the app. For more information, see [Runtime configuration files](#).

- `HelloWorld.dll`

This is the [framework-dependent deployment](#) version of the application. To run this dynamic link library, enter `dotnet HelloWorld.dll` at a command prompt. This method of running the app works on any platform that has the .NET runtime installed.

- `HelloWorld.exe` (`HelloWorld` on Linux, not created on macOS.)

This is the [framework-dependent executable](#) version of the application. The file is operating-system-specific.

- *HelloWorld.pdb* (optional for deployment)

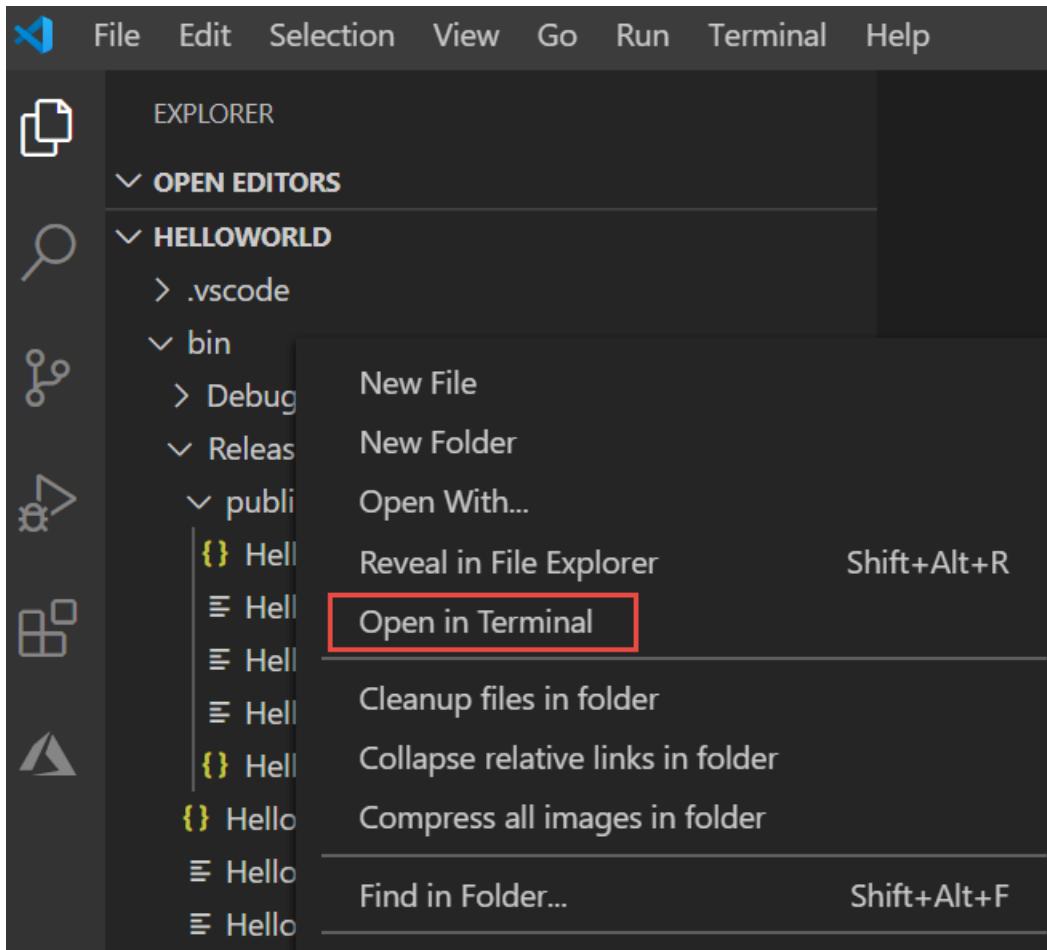
This is the debug symbols file. You aren't required to deploy this file along with your application, although you should save it in the event that you need to debug the published version of your application.

- *HelloWorld.runtimeconfig.json*

This is the application's runtime configuration file. It identifies the version of .NET that your application was built to run on. You can also add configuration options to it. For more information, see [.NET runtime configuration settings](#).

Run the published app

1. In Explorer, right-click the *publish* folder (Ctrl-click on macOS), and select **Open in Terminal**.



2. On Windows or Linux, run the app by using the executable.

- a. On Windows, enter `.\HelloWorld.exe` and press Enter.
- b. On Linux, enter `./HelloWorld` and press Enter.
- c. Enter a name in response to the prompt, and press any key to exit.

3. On any platform, run the app by using the `dotnet` command:

- a. Enter `dotnet HelloWorld.dll` and press Enter.
- b. Enter a name in response to the prompt, and press any key to exit.

Additional resources

- .NET application deployment

Next steps

In this tutorial, you published a console app. In the next tutorial, you create a class library.

[Create a .NET class library using Visual Studio Code](#)

This tutorial shows how to publish a console app so that other users can run it. Publishing creates the set of files that are needed to run an application. To deploy the files, copy them to the target machine.

The .NET CLI is used to publish the app, so you can follow this tutorial with a code editor other than Visual Studio Code if you prefer.

Prerequisites

- This tutorial works with the console app that you create in [Create a .NET console application using Visual Studio Code](#).

Publish the app

1. Start Visual Studio Code.
2. Open the *HelloWorld* project folder that you created in [Create a .NET console application using Visual Studio Code](#).
3. Choose **View > Terminal** from the main menu.

The terminal opens in the *HelloWorld* folder.

4. Run the following command:

```
dotnet publish --configuration Release
```

The default build configuration is *Debug*, so this command specifies the *Release* build configuration. The output from the *Release* build configuration has minimal symbolic debug information and is fully optimized.

The command output is similar to the following example:

```
Microsoft (R) Build Engine version 16.7.0+b89cb5fde for .NET
Copyright (C) Microsoft Corporation. All rights reserved.

Determining projects to restore...
All projects are up-to-date for restore.
HelloWorld -> C:\Projects\HelloWorld\bin\Release\net5.0\HelloWorld.dll
HelloWorld -> C:\Projects\HelloWorld\bin\Release\net5.0\publish\
```

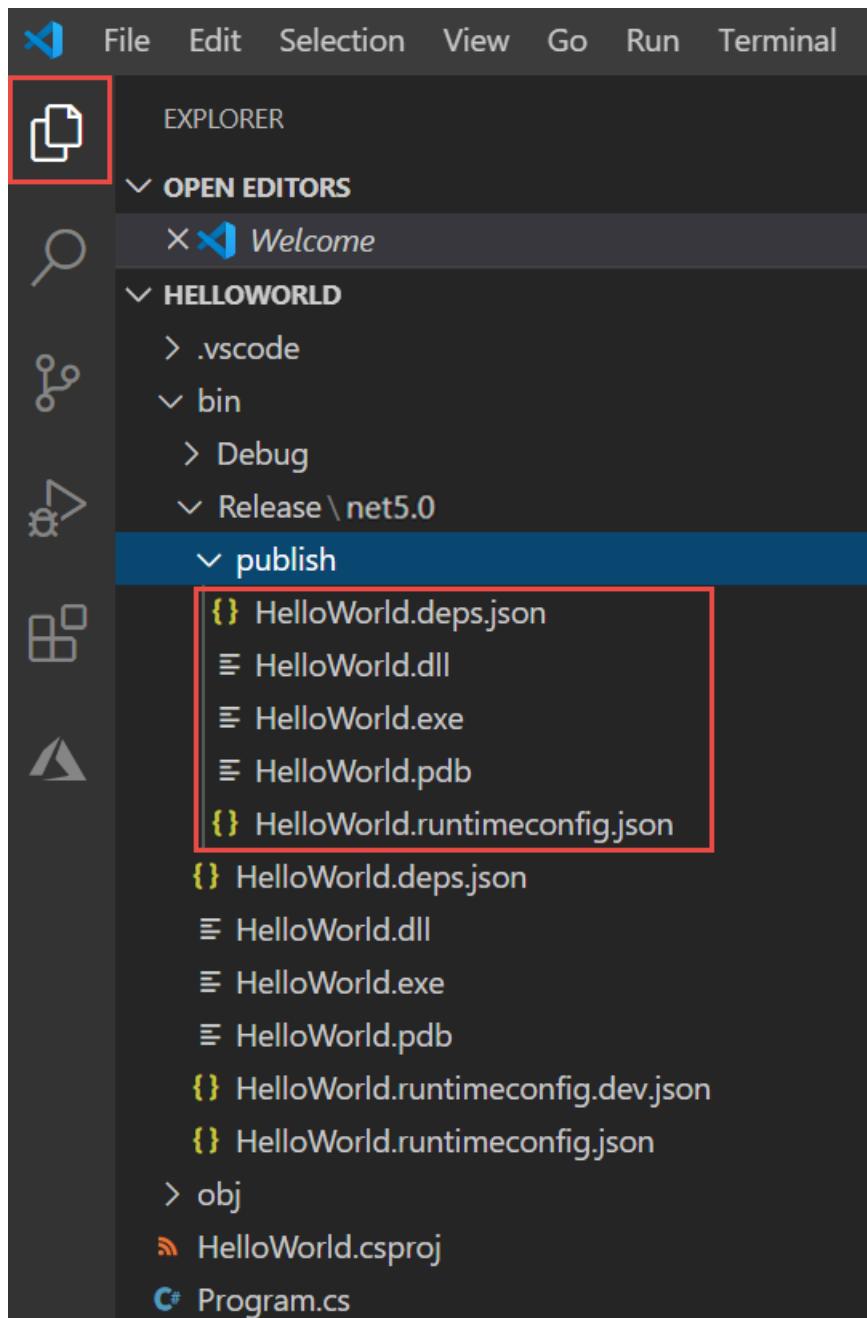
Inspect the files

By default, the publishing process creates a framework-dependent deployment, which is a type of deployment where the published application runs on a machine that has the .NET runtime installed. To run the published app you can use the executable file or run the `dotnet HelloWorld.dll` command from a command prompt.

In the following steps, you'll look at the files created by the publish process.

1. Select the **Explorer** in the left navigation bar.

2. Expand `bin/Release/net5.0/publish`.



As the image shows, the published output includes the following files:

- *HelloWorld.deps.json*

This is the application's runtime dependencies file. It defines the .NET components and the libraries (including the dynamic link library that contains your application) needed to run the app. For more information, see [Runtime configuration files](#).

- *HelloWorld.dll*

This is the [framework-dependent deployment](#) version of the application. To execute this dynamic link library, enter `dotnet HelloWorld.dll` at a command prompt. This method of running the app works on any platform that has the .NET runtime installed.

- *HelloWorld.exe* (*HelloWorld* on Linux, not created on macOS.)

This is the [framework-dependent executable](#) version of the application. The file is operating-system-specific.

- *HelloWorld.pdb* (optional for deployment)

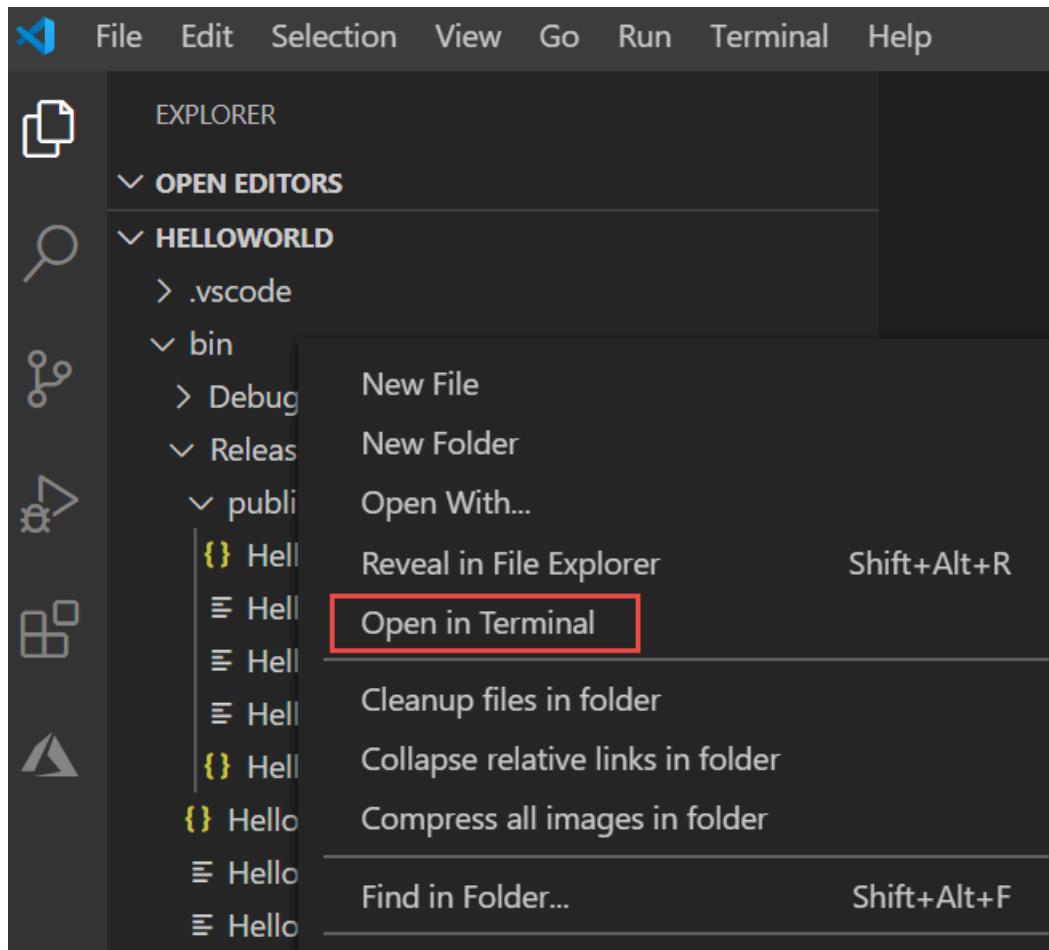
This is the debug symbols file. You aren't required to deploy this file along with your application, although you should save it in the event that you need to debug the published version of your application.

- *HelloWorld.runtimeconfig.json*

This is the application's runtime configuration file. It identifies the version of .NET that your application was built to run on. You can also add configuration options to it. For more information, see [.NET runtime configuration settings](#).

Run the published app

1. In Explorer, right-click the *publish* folder (Ctrl-click on macOS), and select Open in Integrated Terminal.



2. On Windows or Linux, run the app by using the executable.

- a. On Windows, enter `.\\HelloWorld.exe` and press Enter.
- b. On Linux, enter `./HelloWorld` and press Enter.
- c. Enter a name in response to the prompt, and press any key to exit.

3. On any platform, run the app by using the `dotnet` command:

- a. Enter `dotnet HelloWorld.dll` and press Enter.
- b. Enter a name in response to the prompt, and press any key to exit.

Additional resources

- [.NET application deployment](#)

Next steps

In this tutorial, you published a console app. In the next tutorial, you create a class library.

[Create a .NET class library using Visual Studio Code](#)

This tutorial is only available for .NET 5 and .NET 6. Select one of those options at the top of the page.

Tutorial: Create a .NET class library using Visual Studio Code

9/20/2022 • 11 minutes to read • [Edit Online](#)

In this tutorial, you create a simple utility library that contains a single string-handling method.

A *class library* defines types and methods that are called by an application. If the library targets .NET Standard 2.0, it can be called by any .NET implementation (including .NET Framework) that supports .NET Standard 2.0. If the library targets .NET 6, it can be called by any application that targets .NET 6. This tutorial shows how to target .NET 6.

When you create a class library, you can distribute it as a third-party component or as a bundled component with one or more applications.

Prerequisites

- [Visual Studio Code](#) with the [C# extension](#) installed. For information about how to install extensions on Visual Studio Code, see [VS Code Extension Marketplace](#).
- The [.NET 6 SDK](#).

Create a solution

Start by creating a blank solution to put the class library project in. A solution serves as a container for one or more projects. You'll add additional, related projects to the same solution.

1. Start Visual Studio Code.
2. Select **File > Open Folder (Open... on macOS)** from the main menu
3. In the **Open Folder** dialog, create a *ClassLibraryProjects* folder and click **Select Folder (Open on macOS)**.
4. Open the **Terminal** in Visual Studio Code by selecting **View > Terminal** from the main menu.

The **Terminal** opens with the command prompt in the *ClassLibraryProjects* folder.

5. In the **Terminal**, enter the following command:

```
dotnet new sln
```

The terminal output looks like the following example:

```
The template "Solution File" was created successfully.
```

Create a class library project

Add a new .NET class library project named "StringLibrary" to the solution.

1. In the terminal, run the following command to create the library project:

```
dotnet new classlib -o StringLibrary
```

The `-o` or `--output` command specifies the location to place the generated output.

The terminal output looks like the following example:

```
The template "Class library" was created successfully.  
Processing post-creation actions...  
Running 'dotnet restore' on StringLibrary/StringLibrary.csproj...  
Determining projects to restore...  
Restored C:\Projects\ClassLibraryProjects\StringLibrary\StringLibrary.csproj (in 328 ms).  
Restore succeeded.
```

- Run the following command to add the library project to the solution:

```
dotnet sln add StringLibrary/StringLibrary.csproj
```

The terminal output looks like the following example:

```
Project `StringLibrary\StringLibrary.csproj` added to the solution.
```

- Check to make sure that the library targets .NET 6. In **Explorer**, open *StringLibrary/StringLibrary.csproj*.

The `TargetFramework` element shows that the project targets .NET 6.0.

```
<Project Sdk="Microsoft.NET.Sdk">  
  
<PropertyGroup>  
    <TargetFramework>net6.0</TargetFramework>  
</PropertyGroup>  
  
</Project>
```

- Open *Class1.cs* and replace the code with the following code.

```
namespace UtilityLibraries;  
  
public static class StringLibrary  
{  
    public static bool StartsToUpper(this string? str)  
    {  
        if (string.IsNullOrWhiteSpace(str))  
            return false;  
  
        char ch = str[0];  
        return char.IsUpper(ch);  
    }  
}
```

The class library, `UtilityLibraries.StringLibrary`, contains a method named `StartsToUpper`. This method returns a `Boolean` value that indicates whether the current string instance begins with an uppercase character. The Unicode standard distinguishes uppercase characters from lowercase characters. The `Char.IsUpper(Char)` method returns `true` if a character is uppercase.

`StartsToUpper` is implemented as an [extension method](#) so that you can call it as if it were a member of the `String` class.

5. Save the file.
6. Run the following command to build the solution and verify that the project compiles without error.

```
dotnet build
```

The terminal output looks like the following example:

```
Microsoft (R) Build Engine version 16.7.0+b89cb5fde for .NET
Copyright (C) Microsoft Corporation. All rights reserved.

Determining projects to restore...
All projects are up-to-date for restore.
StringLibrary -> C:\Projects\ClassLibraryProjects\StringLibrary\bin\Debug\net6.0\StringLibrary.dll
Build succeeded.

0 Warning(s)
0 Error(s)

Time Elapsed 00:00:02.78
```

Add a console app to the solution

Add a console application that uses the class library. The app will prompt the user to enter a string and report whether the string begins with an uppercase character.

1. In the terminal, run the following command to create the console app project:

```
dotnet new console -o ShowCase
```

The terminal output looks like the following example:

```
The template "Console Application" was created successfully.
Processing post-creation actions...
Running 'dotnet restore' on ShowCase>ShowCase.csproj...
Determining projects to restore...
Restored C:\Projects\ClassLibraryProjects>ShowCase>ShowCase.csproj (in 210 ms).
Restore succeeded.
```

2. Run the following command to add the console app project to the solution:

```
dotnet sln add ShowCase>ShowCase.csproj
```

The terminal output looks like the following example:

```
Project `ShowCase>ShowCase.csproj` added to the solution.
```

3. Open *ShowCase/Program.cs* and replace all of the code with the following code.

```

using UtilityLibraries;

class Program
{
    static void Main(string[] args)
    {
        int row = 0;

        do
        {
            if (row == 0 || row >= 25)
                ResetConsole();

            string? input = Console.ReadLine();
            if (string.IsNullOrEmpty(input)) break;
            Console.WriteLine($"Input: {input}");
            Console.WriteLine("Begins with uppercase? " +
                $"{(input.StartsWithUpper() ? "Yes" : "No")}");
            Console.WriteLine();
            row += 4;
        } while (true);
        return;
    }

    // Declare a ResetConsole local method
    void ResetConsole()
    {
        if (row > 0)
        {
            Console.WriteLine("Press any key to continue...");
            Console.ReadKey();
        }
        Console.Clear();
        Console.WriteLine($"{Environment.NewLine}Press <Enter> only to exit; otherwise, enter a
string and press <Enter>:{Environment.NewLine}");
        row = 3;
    }
}

```

The code uses the `row` variable to maintain a count of the number of rows of data written to the console window. Whenever it's greater than or equal to 25, the code clears the console window and displays a message to the user.

The program prompts the user to enter a string. It indicates whether the string starts with an uppercase character. If the user presses the Enter key without entering a string, the application ends, and the console window closes.

4. Save your changes.

Add a project reference

Initially, the new console app project doesn't have access to the class library. To allow it to call methods in the class library, create a project reference to the class library project.

1. Run the following command:

```
dotnet add ShowCase/ShowCase.csproj reference StringLibrary/StringLibrary.csproj
```

The terminal output looks like the following example:

```
Reference `..\StringLibrary\StringLibrary.csproj` added to the project.
```

Run the app

1. Run the following command in the terminal:

```
dotnet run --project ShowCase>ShowCase.csproj
```

2. Try out the program by entering strings and pressing Enter, then press Enter to exit.

The terminal output looks like the following example:

```
Press <Enter> only to exit; otherwise, enter a string and press <Enter>:  
  
A string that starts with an uppercase letter  
Input: A string that starts with an uppercase letter  
Begins with uppercase? : Yes  
  
a string that starts with a lowercase letter  
Input: a string that starts with a lowercase letter  
Begins with uppercase? : No
```

Additional resources

- [Develop libraries with the .NET CLI](#)
- [.NET Standard versions and the platforms they support](#).

Next steps

In this tutorial, you created a solution, added a library project, and added a console app project that uses the library. In the next tutorial, you add a unit test project to the solution.

[Test a .NET class library with .NET using Visual Studio Code](#)

In this tutorial, you create a simple utility library that contains a single string-handling method.

A *class library* defines types and methods that are called by an application. If the library targets .NET Standard 2.0, it can be called by any .NET implementation (including .NET Framework) that supports .NET Standard 2.0. If the library targets .NET 5, it can be called by any application that targets .NET 5. This tutorial shows how to target .NET 5.

When you create a class library, you can distribute it as a third-party component or as a bundled component with one or more applications.

Prerequisites

1. [Visual Studio Code](#) with the [C# extension](#) installed. For information about how to install extensions on Visual Studio Code, see [VS Code Extension Marketplace](#).
2. The [.NET 5.0 SDK or later](#)

Create a solution

Start by creating a blank solution to put the class library project in. A solution serves as a container for one or more projects. You'll add additional, related projects to the same solution.

1. Start Visual Studio Code.
2. Select **File > Open Folder** (**Open...** on macOS) from the main menu
3. In the **Open Folder** dialog, create a *ClassLibraryProjects* folder and click **Select Folder** (**Open** on macOS).
4. Open the **Terminal** in Visual Studio Code by selecting **View > Terminal** from the main menu.

The **Terminal** opens with the command prompt in the *ClassLibraryProjects* folder.

5. In the **Terminal**, enter the following command:

```
dotnet new sln
```

The terminal output looks like the following example:

```
The template "Solution File" was created successfully.
```

Create a class library project

Add a new .NET class library project named "StringLibrary" to the solution.

1. In the terminal, run the following command to create the library project:

```
dotnet new classlib -o StringLibrary
```

The `-o` or `--output` command specifies the location to place the generated output.

The terminal output looks like the following example:

```
The template "Class library" was created successfully.  
Processing post-creation actions...  
Running 'dotnet restore' on StringLibrary/StringLibrary.csproj...  
Determining projects to restore...  
Restored C:\Projects\ClassLibraryProjects\StringLibrary\StringLibrary.csproj (in 328 ms).  
Restore succeeded.
```

2. Run the following command to add the library project to the solution:

```
dotnet sln add StringLibrary/StringLibrary.csproj
```

The terminal output looks like the following example:

```
Project `StringLibrary\StringLibrary.csproj` added to the solution.
```

3. Check to make sure that the library targets .NET 5. In **Explorer**, open *StringLibrary/StringLibrary.csproj*.

The `TargetFramework` element shows that the project targets .NET 5.0.

```
<Project Sdk="Microsoft.NET.Sdk">

<PropertyGroup>
<TargetFramework>net5.0</TargetFramework>
</PropertyGroup>

</Project>
```

4. Open *Class1.cs* and replace the code with the following code.

```
using System;

namespace UtilityLibraries
{
    public static class StringLibrary
    {
        public static bool StartsWithUpper(this string str)
        {
            if (string.IsNullOrWhiteSpace(str))
                return false;

            char ch = str[0];
            return char.IsUpper(ch);
        }
    }
}
```

The class library, `UtilityLibraries.StringLibrary`, contains a method named `StartsWithUpper`. This method returns a `Boolean` value that indicates whether the current string instance begins with an uppercase character. The Unicode standard distinguishes uppercase characters from lowercase characters. The `Char.IsUpper(Char)` method returns `true` if a character is uppercase.

`StartsWithUpper` is implemented as an [extension method](#) so that you can call it as if it were a member of the `String` class. The question mark (?) after `string` indicates that the string may be null.

5. Save the file.
6. Run the following command to build the solution and verify that the project compiles without error.

```
dotnet build
```

The terminal output looks like the following example:

```
Microsoft (R) Build Engine version 16.7.0+b89cb5fde for .NET
Copyright (C) Microsoft Corporation. All rights reserved.
Determining projects to restore...
All projects are up-to-date for restore.
StringLibrary -> C:\Projects\ClassLibraryProjects\StringLibrary\bin\Debug\net5.0\StringLibrary.dll
Build succeeded.
0 Warning(s)
0 Error(s)
Time Elapsed 00:00:02.78
```

Add a console app to the solution

Add a console application that uses the class library. The app will prompt the user to enter a string and report whether the string begins with an uppercase character.

1. In the terminal, run the following command to create the console app project:

```
dotnet new console -o ShowCase
```

The terminal output looks like the following example:

```
The template "Console Application" was created successfully.  
Processing post-creation actions...  
Running 'dotnet restore' on ShowCase>ShowCase.csproj...  
Determining projects to restore...  
Restored C:\Projects\ClassLibraryProjects>ShowCase>ShowCase.csproj (in 210 ms).  
Restore succeeded.
```

2. Run the following command to add the console app project to the solution:

```
dotnet sln add ShowCase>ShowCase.csproj
```

The terminal output looks like the following example:

```
Project `ShowCase>ShowCase.csproj` added to the solution.
```

3. Open *ShowCase/Program.cs* and replace all of the code with the following code.

```

using System;
using UtilityLibraries;

class Program
{
    static void Main(string[] args)
    {
        int row = 0;

        do
        {
            if (row == 0 || row >= 25)
                ResetConsole();

            string? input = Console.ReadLine();
            if (string.IsNullOrEmpty(input)) break;
            Console.WriteLine($"Input: {input} {"Begins with uppercase? ",30}: " +
                $"{{(input.StartsWithUpper() ? "Yes" : "No")}{Environment.NewLine}}");
            row += 3;
        } while (true);
        return;
    }

    // Declare a ResetConsole local method
    void ResetConsole()
    {
        if (row > 0)
        {
            Console.WriteLine("Press any key to continue...");
            Console.ReadKey();
        }
        Console.Clear();
        Console.WriteLine($"{Environment.NewLine}Press <Enter> only to exit; otherwise, enter a
string and press <Enter>:{Environment.NewLine}");
        row = 3;
    }
}

```

The code uses the `row` variable to maintain a count of the number of rows of data written to the console window. Whenever it's greater than or equal to 25, the code clears the console window and displays a message to the user.

The program prompts the user to enter a string. It indicates whether the string starts with an uppercase character. If the user presses the Enter key without entering a string, the application ends, and the console window closes.

4. Save your changes.

Add a project reference

Initially, the new console app project doesn't have access to the class library. To allow it to call methods in the class library, create a project reference to the class library project.

1. Run the following command:

```
dotnet add ShowCase/ShowCase.csproj reference StringLibrary/StringLibrary.csproj
```

The terminal output looks like the following example:

```
Reference `..\StringLibrary\StringLibrary.csproj` added to the project.
```

Run the app

1. Run the following command in the terminal:

```
dotnet run --project ShowCase>ShowCase.csproj
```

2. Try out the program by entering strings and pressing Enter, then press Enter to exit.

The terminal output looks like the following example:

```
Press <Enter> only to exit; otherwise, enter a string and press <Enter>:  
  
A string that starts with an uppercase letter  
Input: A string that starts with an uppercase letter  
Begins with uppercase? : Yes  
  
a string that starts with a lowercase letter  
Input: a string that starts with a lowercase letter  
Begins with uppercase? : No
```

Additional resources

- [Develop libraries with the .NET CLI](#)
- [.NET Standard versions and the platforms they support.](#)

Next steps

In this tutorial, you created a solution, added a library project, and added a console app project that uses the library. In the next tutorial, you add a unit test project to the solution.

[Test a .NET class library with .NET using Visual Studio Code](#)

This tutorial is only available for .NET 5 and .NET 6. Select one of those options at the top of the page.

Tutorial: Test a .NET class library using Visual Studio Code

9/20/2022 • 14 minutes to read • [Edit Online](#)

This tutorial shows how to automate unit testing by adding a test project to a solution.

Prerequisites

- This tutorial works with the solution that you create in [Create a .NET class library using Visual Studio Code](#).

Create a unit test project

Unit tests provide automated software testing during your development and publishing. The testing framework that you use in this tutorial is MSTest. [MSTest](#) is one of three test frameworks you can choose from. The others are [xUnit](#) and [nUnit](#).

1. Start Visual Studio Code.
2. Open the `ClassLibraryProjects` solution you created in [Create a .NET class library using Visual Studio Code](#).
3. Create a unit test project named "StringLibraryTest".

```
dotnet new mstest -o StringLibraryTest
```

The project template creates a `UnitTest1.cs` file with the following code:

```
using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace StringLibraryTest
{
    [TestClass]
    public class UnitTest1
    {
        [TestMethod]
        public void TestMethod1()
        {
        }
    }
}
```

The source code created by the unit test template does the following:

- It imports the [Microsoft.VisualStudio.TestTools.UnitTesting](#) namespace, which contains the types used for unit testing.
- It applies the [TestClassAttribute](#) attribute to the `UnitTest1` class.
- It applies the [TestMethodAttribute](#) attribute to define `TestMethod1`.

Each method tagged with [\[TestMethod\]](#) in a test class tagged with [\[TestClass\]](#) is run automatically when the unit test is invoked.

4. Add the test project to the solution.

```
dotnet sln add StringLibraryTest/StringLibraryTest.csproj
```

Add a project reference

For the test project to work with the `StringLibrary` class, add a reference in the `StringLibraryTest` project to the `StringLibrary` project.

1. Run the following command:

```
dotnet add StringLibraryTest/StringLibraryTest.csproj reference StringLibrary/StringLibrary.csproj
```

Add and run unit test methods

When Visual Studio invokes a unit test, it runs each method that is marked with the `TestMethodAttribute` attribute in a class that is marked with the `TestClassAttribute` attribute. A test method ends when the first failure is found or when all tests contained in the method have succeeded.

The most common tests call members of the `Assert` class. Many assert methods include at least two parameters, one of which is the expected test result and the other of which is the actual test result. Some of the `Assert` class's most frequently called methods are shown in the following table:

ASSERT METHODS	FUNCTION
<code>Assert.AreEqual</code>	Verifies that two values or objects are equal. The assert fails if the values or objects aren't equal.
<code>Assert.AreSame</code>	Verifies that two object variables refer to the same object. The assert fails if the variables refer to different objects.
<code>Assert.IsFalse</code>	Verifies that a condition is <code>false</code> . The assert fails if the condition is <code>true</code> .
<code>Assert.IsNotNull</code>	Verifies that an object isn't <code>null</code> . The assert fails if the object is <code>null</code> .

You can also use the `Assert.ThrowsException` method in a test method to indicate the type of exception it's expected to throw. The test fails if the specified exception isn't thrown.

In testing the `StringLibrary.StartsWithUpper` method, you want to provide a number of strings that begin with an uppercase character. You expect the method to return `true` in these cases, so you can call the `Assert.IsTrue` method. Similarly, you want to provide a number of strings that begin with something other than an uppercase character. You expect the method to return `false` in these cases, so you can call the `Assert.IsFalse` method.

Since your library method handles strings, you also want to make sure that it successfully handles an `empty string` (`String.Empty`) and a `null` string. An empty string is one that has no characters and whose `Length` is 0. A `null` string is one that hasn't been initialized. You can call `StartsWithUpper` directly as a static method and pass a single `String` argument. Or you can call `StartsWithUpper` as an extension method on a `string` variable assigned to `null`.

You'll define three methods, each of which calls an `Assert` method for each element in a string array. You'll call a method overload that lets you specify an error message to be displayed in case of test failure. The message identifies the string that caused the failure.

To create the test methods:

1. Open *StringLibraryTest/UnitTest1.cs* and replace all of the code with the following code.

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using UtilityLibraries;

namespace StringLibraryTest;

[TestClass]
public class UnitTest1
{
    [TestMethod]
    public void TestStartsWithUpper()
    {
        // Tests that we expect to return true.
        string[] words = { "Alphabet", "Zebra", "ABC", "Αθήνα", "Москва" };
        foreach (var word in words)
        {
            bool result = word.StartsWithUpper();
            Assert.IsTrue(result,
                string.Format("Expected for '{0}': true; Actual: {1}",
                    word, result));
        }
    }

    [TestMethod]
    public void TestDoesNotStartWithUpper()
    {
        // Tests that we expect to return false.
        string[] words = { "alphabet", "zebra", "abc", "αυτοκινητοβιομηχανία", "государство",
            "1234", ".", ";", " " };
        foreach (var word in words)
        {
            bool result = word.StartsWithUpper();
            Assert.IsFalse(result,
                string.Format("Expected for '{0}': false; Actual: {1}",
                    word, result));
        }
    }

    [TestMethod]
    public void DirectCallWithNullOrEmpty()
    {
        // Tests that we expect to return false.
        string?[] words = { string.Empty, null };
        foreach (var word in words)
        {
            bool result = StringLibrary.StartsWithUpper(word);
            Assert.IsFalse(result,
                string.Format("Expected for '{0}': false; Actual: {1}",
                    word == null ? "<null>" : word, result));
        }
    }
}
```

The test of uppercase characters in the `TestStartsWithUpper` method includes the Greek capital letter alpha (U+0391) and the Cyrillic capital letter EM (U+041C). The test of lowercase characters in the `TestDoesNotStartWithUpper` method includes the Greek small letter alpha (U+03B1) and the Cyrillic small letter Ghe (U+0433).

2. Save your changes.

3. Run the tests:

```
dotnet test StringLibraryTest/StringLibraryTest.csproj
```

The terminal output shows that all tests passed.

```
Starting test execution, please wait...
A total of 1 test files matched the specified pattern.

Passed! - Failed: 0, Passed: 3, Skipped: 0, Total: 3, Duration: 3 ms -
StringLibraryTest.dll (net6.0)
```

Handle test failures

If you're doing test-driven development (TDD), you write tests first and they fail the first time you run them. Then you add code to the app that makes the test succeed. For this tutorial, you created the test after writing the app code that it validates, so you haven't seen the test fail. To validate that a test fails when you expect it to fail, add an invalid value to the test input.

1. Modify the `words` array in the `TestDoesNotStartWithUpper` method to include the string "Error".

```
string[] words = { "alphabet", "Error", "zebra", "abc", "αυτοκινητοβιομηχανία", "государство",
    "1234", ".", ";" };
```

2. Run the tests:

```
dotnet test StringLibraryTest/StringLibraryTest.csproj
```

The terminal output shows that one test fails, and it provides an error message for the failed test: "Assert.IsFalse failed. Expected for 'Error': false; actual: True". Because of the failure, no strings in the array after "Error" were tested.

```
Starting test execution, please wait...
A total of 1 test files matched the specified pattern.

Failed TestDoesNotStartWithUpper [28 ms]
Error Message:
  Assert.IsFalse failed. Expected for 'Error': false; Actual: True
Stack Trace:
  at StringLibraryTest.UnitTest1.TestDoesNotStartWithUpper() in
C:\ClassLibraryProjects\StringLibraryTest\UnitTest1.cs:line 33

Failed! - Failed: 1, Passed: 2, Skipped: 0, Total: 3, Duration: 31 ms -
StringLibraryTest.dll (net5.0)
```

3. Remove the string "Error" that you added in step 1. Rerun the test and the tests pass.

Test the Release version of the library

Now that the tests have all passed when running the Debug build of the library, run the tests an additional time against the Release build of the library. A number of factors, including compiler optimizations, can sometimes produce different behavior between Debug and Release builds.

1. Run the tests with the Release build configuration:

```
dotnet test StringLibraryTest/StringLibraryTest.csproj --configuration Release
```

The tests pass.

Debug tests

If you're using Visual Studio Code as your IDE, you can use the same process shown in [Debug a .NET console application using Visual Studio Code](#) to debug code using your unit test project. Instead of starting the *ShowCase* app project, open *StringLibraryTest/UnitTest1.cs*, and select **Debug All Tests** between lines 7 and 8. If you're unable to find it, press **Ctrl+Shift+P** to open the command palette and enter **Reload Window**.

Visual Studio Code starts the test project with the debugger attached. Execution will stop at any breakpoint you've added to the test project or the underlying library code.

Additional resources

- [Unit testing in .NET](#)

Next steps

In this tutorial, you unit tested a class library. You can make the library available to others by publishing it to [NuGet](#) as a package. To learn how, follow a NuGet tutorial:

[Create and publish a package using the dotnet CLI](#)

If you publish a library as a NuGet package, others can install and use it. To learn how, follow a NuGet tutorial:

[Install and use a package using the dotnet CLI](#)

A library doesn't have to be distributed as a package. It can be bundled with a console app that uses it. To learn how to publish a console app, see the earlier tutorial in this series:

[Publish a .NET console application using Visual Studio Code](#)

This tutorial shows how to automate unit testing by adding a test project to a solution.

Prerequisites

- This tutorial works with the solution that you create in [Create a .NET class library using Visual Studio Code](#).

Create a unit test project

Unit tests provide automated software testing during your development and publishing. The testing framework that you use in this tutorial is MSTest. [MSTest](#) is one of three test frameworks you can choose from. The others are [xUnit](#) and [nUnit](#).

1. Start Visual Studio Code.
2. Open the `ClassLibraryProjects` solution you created in [Create a .NET class library using Visual Studio Code](#).
3. Create a unit test project named "StringLibraryTest".

```
dotnet new mstest -o StringLibraryTest
```

The project template creates a `UnitTest1.cs` file with the following code:

```

using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace StringLibraryTest
{
    [TestClass]
    public class UnitTest1
    {
        [TestMethod]
        public void TestMethod1()
        {
        }
    }
}

```

The source code created by the unit test template does the following:

- It imports the `Microsoft.VisualStudio.TestTools.UnitTesting` namespace, which contains the types used for unit testing.
- It applies the `TestClassAttribute` attribute to the `UnitTest1` class.
- It applies the `TestMethodAttribute` attribute to define `TestMethod1`.

Each method tagged with `[TestMethod]` in a test class tagged with `[TestClass]` is executed automatically when the unit test is run.

4. Add the test project to the solution.

```
dotnet sln add StringLibraryTest/StringLibraryTest.csproj
```

Add a project reference

For the test project to work with the `StringLibrary` class, add a reference in the `StringLibraryTest` project to the `StringLibrary` project.

1. Run the following command:

```
dotnet add StringLibraryTest/StringLibraryTest.csproj reference StringLibrary/StringLibrary.csproj
```

Add and run unit test methods

When Visual Studio runs a unit test, it executes each method that is marked with the `TestMethodAttribute` attribute in a class that is marked with the `TestClassAttribute` attribute. A test method ends when the first failure is found or when all tests contained in the method have succeeded.

The most common tests call members of the `Assert` class. Many assert methods include at least two parameters, one of which is the expected test result and the other of which is the actual test result. Some of the `Assert` class's most frequently called methods are shown in the following table:

ASSERT METHODS	FUNCTION
<code>Assert.AreEqual</code>	Verifies that two values or objects are equal. The assert fails if the values or objects aren't equal.
<code>Assert.AreSame</code>	Verifies that two object variables refer to the same object. The assert fails if the variables refer to different objects.

ASSERT METHODS	FUNCTION
<code>Assert.IsFalse</code>	Verifies that a condition is <code>false</code> . The assert fails if the condition is <code>true</code> .
<code>Assert.IsNotNull</code>	Verifies that an object isn't <code>null</code> . The assert fails if the object is <code>null</code> .

You can also use the `Assert.ThrowsException` method in a test method to indicate the type of exception it's expected to throw. The test fails if the specified exception isn't thrown.

In testing the `StringLibrary.StartsWithUpper` method, you want to provide a number of strings that begin with an uppercase character. You expect the method to return `true` in these cases, so you can call the `Assert.IsTrue` method. Similarly, you want to provide a number of strings that begin with something other than an uppercase character. You expect the method to return `false` in these cases, so you can call the `Assert.IsFalse` method.

Since your library method handles strings, you also want to make sure that it successfully handles an `empty string` (`String.Empty`) and a `null` string. An empty string is one that has no characters and whose `Length` is 0. A `null` string is one that hasn't been initialized. You can call `StartsWithUpper` directly as a static method and pass a single `String` argument. Or you can call `StartsWithUpper` as an extension method on a `string` variable assigned to `null`.

You'll define three methods, each of which calls an `Assert` method for each element in a string array. You'll call a method overload that lets you specify an error message to be displayed in case of test failure. The message identifies the string that caused the failure.

To create the test methods:

1. Open `StringLibraryTest/UnitTest1.cs` and replace all of the code with the following code.

```

using Microsoft.VisualStudio.TestTools.UnitTesting;
using UtilityLibraries;

namespace StringLibraryTest
{
    [TestClass]
    public class UnitTest1
    {
        [TestMethod]
        public void TestStartsWithUpper()
        {
            // Tests that we expect to return true.
            string[] words = { "Alphabet", "Zebra", "ABC", "Αθήνα", "Москва" };
            foreach (var word in words)
            {
                bool result = word.StartsWithUpper();
                Assert.IsTrue(result,
                    string.Format("Expected for '{0}': true; Actual: {1}",
                    word, result));
            }
        }

        [TestMethod]
        public void TestDoesNotStartWithUpper()
        {
            // Tests that we expect to return false.
            string[] words = { "alphabet", "zebra", "abc", "αυτοκινητοβιομηχανία", "государство",
                "1234", ".", ";", " " };
            foreach (var word in words)
            {
                bool result = word.StartsWithUpper();
                Assert.IsFalse(result,
                    string.Format("Expected for '{0}': false; Actual: {1}",
                    word, result));
            }
        }

        [TestMethod]
        public void DirectCallWithNullOrEmpty()
        {
            // Tests that we expect to return false.
            string?[] words = { string.Empty, null };
            foreach (var word in words)
            {
                bool result = StringLibrary.StartsWithUpper(word ?? string.Empty);
                Assert.IsFalse(result,
                    string.Format("Expected for '{0}': false; Actual: {1}",
                    word == null ? "<null>" : word, result));
            }
        }
    }
}

```

The test of uppercase characters in the `TestStartsWithUpper` method includes the Greek capital letter alpha (U+0391) and the Cyrillic capital letter EM (U+041C). The test of lowercase characters in the `TestDoesNotStartWithUpper` method includes the Greek small letter alpha (U+03B1) and the Cyrillic small letter Ghe (U+0433).

2. Save your changes.

3. Run the tests:

```
dotnet test StringLibraryTest/StringLibraryTest.csproj
```

The terminal output shows that all tests passed.

```
Starting test execution, please wait...
A total of 1 test files matched the specified pattern.

Passed! - Failed: 0, Passed: 3, Skipped: 0, Total: 3, Duration: 3 ms -
StringLibraryTest.dll (net5.0)
```

Handle test failures

If you're doing test-driven development (TDD), you write tests first and they fail the first time you run them. Then you add code to the app that makes the test succeed. For this tutorial, you created the test after writing the app code that it validates, so you haven't seen the test fail. To validate that a test fails when you expect it to fail, add an invalid value to the test input.

1. Modify the `words` array in the `TestDoesNotStartWithUpper` method to include the string "Error".

```
string[] words = { "alphabet", "Error", "zebra", "abc", "αυτοκινητοβιομηχανία", "государство",
    "1234", ".", ";", " " };
```

2. Run the tests:

```
dotnet test StringLibraryTest/StringLibraryTest.csproj
```

The terminal output shows that one test fails, and it provides an error message for the failed test: "Assert.IsFalse failed. Expected for 'Error': false; actual: True". Because of the failure, no strings in the array after "Error" were tested.

```
Starting test execution, please wait...
A total of 1 test files matched the specified pattern.
Failed TestDoesNotStartWithUpper [28 ms]
Error Message:
  Assert.IsFalse failed. Expected for 'Error': false; Actual: True
Stack Trace:
  at StringLibraryTest.UnitTest1.TestDoesNotStartWithUpper() in
C:\ClassLibraryProjects\StringLibraryTest\UnitTest1.cs:line 33

Failed! - Failed: 1, Passed: 2, Skipped: 0, Total: 3, Duration: 31 ms -
StringLibraryTest.dll (net5.0)
```

3. Remove the string "Error" that you added in step 1. Rerun the test and the tests pass.

Test the Release version of the library

Now that the tests have all passed when running the Debug build of the library, run the tests an additional time against the Release build of the library. A number of factors, including compiler optimizations, can sometimes produce different behavior between Debug and Release builds.

1. Run the tests with the Release build configuration:

```
dotnet test StringLibraryTest/StringLibraryTest.csproj --configuration Release
```

The tests pass.

Debug tests

If you're using Visual Studio Code as your IDE, you can use the same process shown in [Debug a .NET console application using Visual Studio Code](#) to debug code using your unit test project. Instead of starting the *ShowCase* app project, open *StringLibraryTest/UnitTest1.cs*, and select **Debug All Tests** between lines 7 and 8. If you're unable to find it, press **Ctrl+Shift+P** to open the command palette and enter **Reload Window**.

Visual Studio Code starts the test project with the debugger attached. Execution will stop at any breakpoint you've added to the test project or the underlying library code.

Additional resources

- [Unit testing in .NET](#)

Next steps

In this tutorial, you unit tested a class library. You can make the library available to others by publishing it to [NuGet](#) as a package. To learn how, follow a NuGet tutorial:

[Create and publish a package using the dotnet CLI](#)

If you publish a library as a NuGet package, others can install and use it. To learn how, follow a NuGet tutorial:

[Install and use a package using the dotnet CLI](#)

A library doesn't have to be distributed as a package. It can be bundled with a console app that uses it. To learn how to publish a console app, see the earlier tutorial in this series:

[Publish a .NET console application using Visual Studio Code](#)

This tutorial is only available for .NET 5 and .NET 6. Select one of those options at the top of the page.

Tutorial: Create a .NET console application using Visual Studio for Mac

9/20/2022 • 3 minutes to read • [Edit Online](#)

This tutorial shows how to create and run a .NET console application using Visual Studio for Mac.

NOTE

Your feedback is highly valued. There are two ways you can provide feedback to the development team on Visual Studio for Mac:

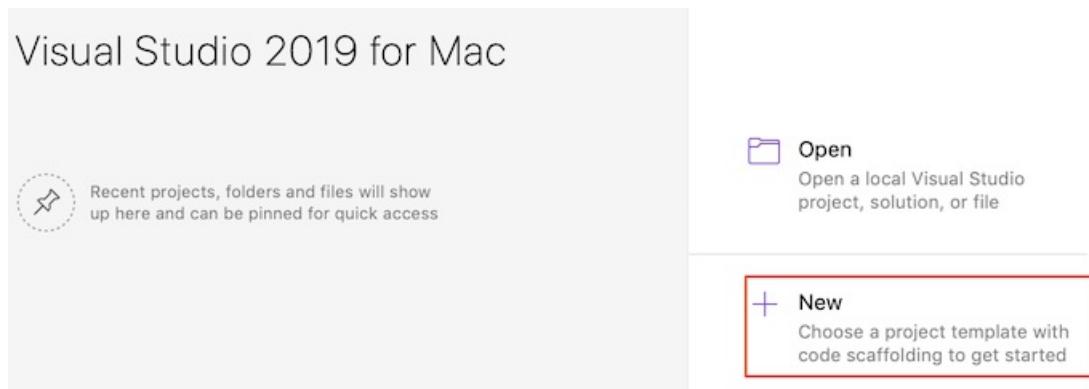
- In Visual Studio for Mac, select **Help > Report a Problem** from the menu or **Report a Problem** from the Welcome screen, which will open a window for filing a bug report. You can track your feedback in the [Developer Community portal](#).
- To make a suggestion, select **Help > Provide a Suggestion** from the menu or **Provide a Suggestion** from the Welcome screen, which will take you to the [Visual Studio for Mac Developer Community webpage](#).

Prerequisites

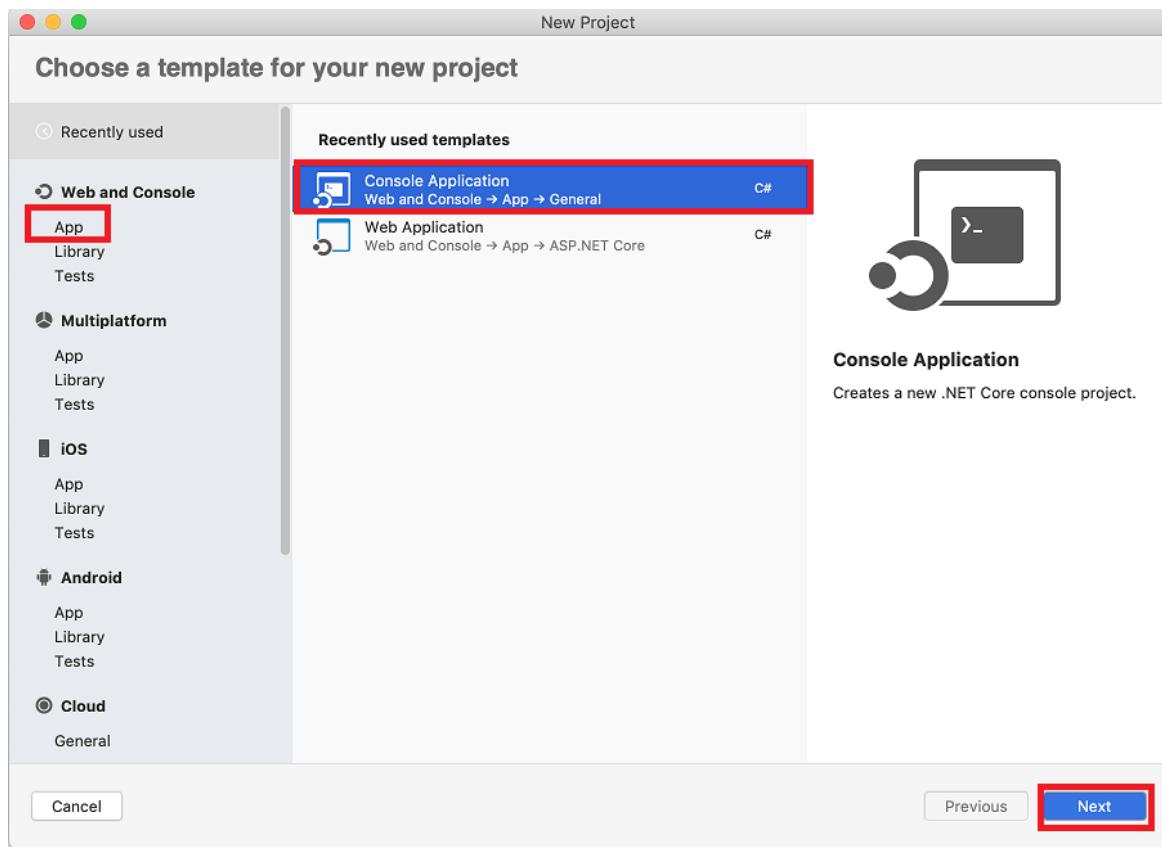
- [Visual Studio for Mac version 8.8 or later](#). Select the option to install .NET Core. Installing Xamarin is optional for .NET development. For more information, see the following resources:
 - [Tutorial: Install Visual Studio for Mac](#).
 - [Supported macOS versions](#).
 - [.NET versions supported by Visual Studio for Mac](#).

Create the app

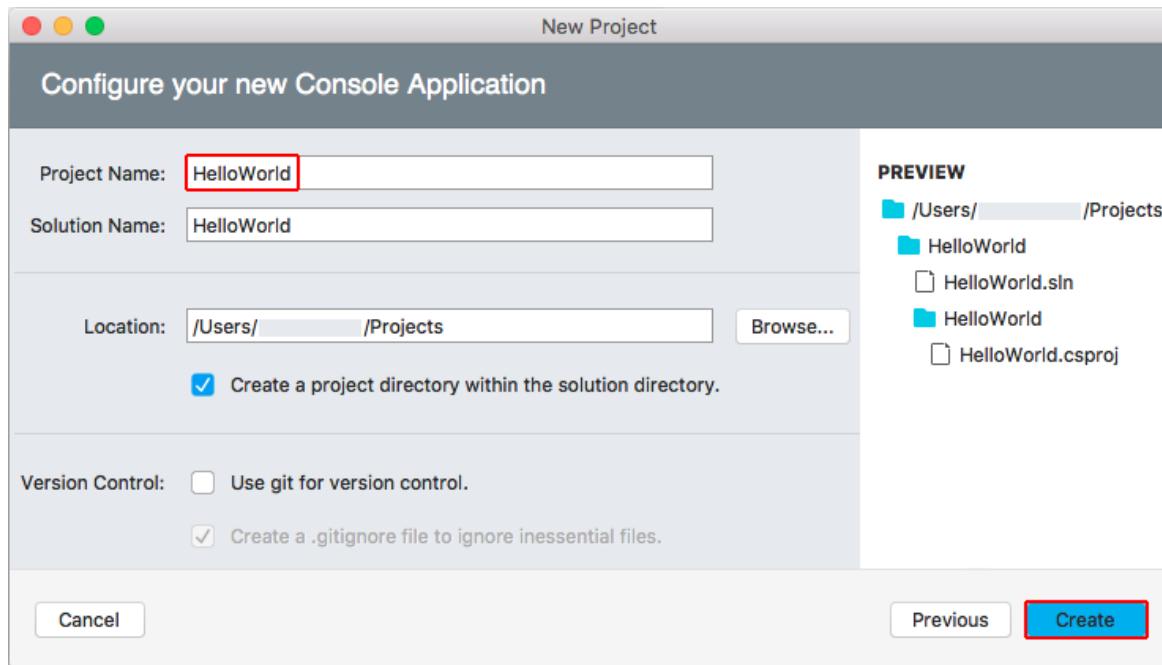
1. Start Visual Studio for Mac.
2. Select **New** in the start window.



3. In the **New Project** dialog, select **App** under the **Web and Console** node. Select the **Console Application** template, and select **Next**.



4. In the Target Framework drop-down of the Configure your new Console Application dialog, select .NET 5.0, and select Next.
5. Type "HelloWorld" for the Project Name, and select Create.



The template creates a simple "Hello World" application. It calls the `Console.WriteLine(String)` method to display "Hello World!" in the terminal window.

The template code defines a class, `Program`, with a single method, `Main`, that takes a `String` array as an argument:

```
using System;

namespace HelloWorld
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

Main is the application entry point, the method that's called automatically by the runtime when it launches the application. Any command-line arguments supplied when the application is launched are available in the args array.

Run the app

1. Press ⌘+option+command+enter to run the app without debugging.

The screenshot shows the Visual Studio interface on a Mac. The menu bar includes Apple, Visual Studio, File, Edit, View, Search, Project, Build, and Run. The toolbar has buttons for Stop, Start, Debug, and Default. A status bar message says "We'd love to hear about your .NET Core experience in Visual Studio for Mac." The code editor window is titled "Program.cs" and contains the provided C# code. The terminal window at the bottom is titled "Terminal - HelloWorld" and displays the output "Hello World!".

```
1 using System;
2
3 namespace HelloWorld
4 {
5     class Program
6     {
7         static void Main(string[] args)
8         {
9             Console.WriteLine("Hello World!");
10        }
11    }
12 }
13
```

```
Terminal - HelloWorld
Hello World!
```

2. Close the **Terminal** window.

Enhance the app

Enhance the application to prompt the user for their name and display it along with the date and time.

1. In *Program.cs*, replace the contents of the `Main` method, which is the line that calls `Console.WriteLine`, with the following code:

```
Console.WriteLine("What is your name?");
var name = Console.ReadLine();
var currentDate = DateTime.Now;
Console.WriteLine($"{Environment.NewLine}Hello, {name}, on {currentDate:d} at {currentDate:t}!");
Console.Write($"{Environment.NewLine}Press any key to exit...");
Console.ReadKey(true);
```

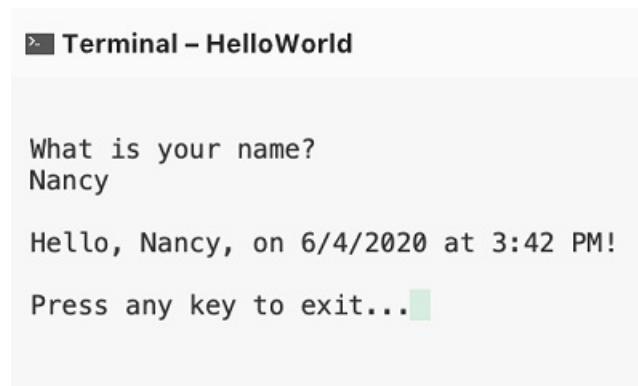
This code displays a prompt in the console window and waits until the user enters a string followed by the enter key. It stores this string in a variable named `name`. It also retrieves the value of the `DateTime.Now` property, which contains the current local time, and assigns it to a variable named `currentDate`. And it displays these values in the console window. Finally, it displays a prompt in the console window and calls the `Console.ReadKey(Boolean)` method to wait for user input.

`NewLine` is a platform-independent and language-independent way to represent a line break.

Alternatives are `\n` in C# and `vbcrlf` in Visual Basic.

The dollar sign (`$`) in front of a string lets you put expressions such as variable names in curly braces in the string. The expression value is inserted into the string in place of the expression. This syntax is referred to as [interpolated strings](#).

2. Press `⌘⌃⏎` (option+command+enter) to run the app.
3. Respond to the prompt by entering a name and pressing enter.



4. Close the terminal.

Next steps

In this tutorial, you created a .NET console application. In the next tutorial, you debug the app.

[Debug a .NET console application using Visual Studio for Mac](#)

Tutorial: Debug a .NET console application using Visual Studio for Mac

9/20/2022 • 6 minutes to read • [Edit Online](#)

This tutorial introduces the debugging tools available in Visual Studio for Mac.

Prerequisites

- This tutorial works with the console app that you create in [Create a .NET console application using Visual Studio for Mac](#).

Use Debug build configuration

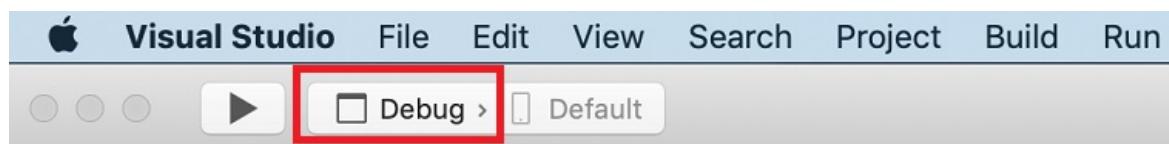
Debug and *Release* are Visual Studio's built-in build configurations. You use the Debug build configuration for debugging and the Release configuration for the final release distribution.

In the Debug configuration, a program compiles with full symbolic debug information and no optimization. Optimization complicates debugging, because the relationship between source code and generated instructions is more complex. The release configuration of a program has no symbolic debug information and is fully optimized.

By default, Visual Studio for Mac uses the Debug build configuration, so you don't need to change it before debugging.

1. Start Visual Studio for Mac.
2. Open the project that you created in [Create a .NET console application using Visual Studio for Mac](#).

The current build configuration is shown on the toolbar. The following toolbar image shows that Visual Studio is configured to compile the Debug version of the app:



Set a breakpoint

A *breakpoint* temporarily interrupts the execution of the application before the line with the breakpoint is executed.

1. Set a breakpoint on the line that displays the name, date, and time. To do that, place the cursor in the line of code and press $\text{⌘}\backslash$ ($\text{command} + \backslash$). Another way to set a breakpoint is by selecting **Run > Toggle Breakpoint** from the menu.

Visual Studio indicates the line on which the breakpoint is set by highlighting it and displaying a red dot in the left margin.

A screenshot of the Visual Studio code editor showing the file `Program.cs`. The code defines a `Program` class with a `Main` method. A red rectangle highlights the line `Console.WriteLine($"{Environment.NewLine}Hello, {name}, on {currentDate}");`, which contains a breakpoint (indicated by a circled number 12). The code also includes a `Console.ReadKey(true);` statement at the end of the method.

```
1  using System;
2
3  namespace HelloWorld
4  {
5      class Program
6      {
7          static void Main(string[] args)
8          {
9              Console.WriteLine("What is your name?");
10             var name = Console.ReadLine();
11             var currentDate = DateTime.Now;
12             Console.WriteLine($"{Environment.NewLine}Hello, {name}, on {currentDate}");
13             Console.WriteLine($"{Environment.NewLine}Press any key to exit...");
14             Console.ReadKey(true);
15         }
16     }
17 }
```

2. Press ⌘+↑ (command+enter) to start the program in debugging mode. Another way to start debugging is by choosing Run > Start Debugging from the menu.
3. Enter a string in the terminal window when the program prompts for a name, and then press enter.
4. Program execution stops when it reaches the breakpoint, before the `Console.WriteLine` method executes.

A screenshot of the Visual Studio code editor showing the file `Program.cs`. The code is identical to the previous screenshot, but the line `Console.WriteLine($"{Environment.NewLine}Hello, {name}, on {currentDate}");` does not have a red highlight or a breakpoint icon. The `Console.ReadKey(true);` statement is still present at the end of the method.

```
1  using System;
2
3  namespace HelloWorld
4  {
5      class Program
6      {
7          static void Main(string[] args)
8          {
9              Console.WriteLine("What is your name?");
10             var name = Console.ReadLine();
11             var currentDate = DateTime.Now;
12             Console.WriteLine($"{Environment.NewLine}Hello, {name}, on {currentDate}");
13             Console.WriteLine($"{Environment.NewLine}Press any key to exit...");
14             Console.ReadKey(true);
15         }
16     }
17 }
```

Use the Immediate window

The **Immediate** window lets you interact with the application you're debugging. You can interactively change the value of variables to see how it affects your program.

1. If the **Immediate** window is not visible, display it by choosing View > Debug Pads > **Immediate**.
2. Enter `name = "Gracie"` in the **Immediate** window and press enter.
3. Enter `currentDate = currentDate.AddDays(1)` in the **Immediate** window and press enter.

The **Immediate** window displays the new value of the string variable and the properties of the `DateTime` value.

```
[-] Immediate
name = "Gracie"
"Gracie"
currentDate = currentDate.AddDays(1)
{5/27/2021 11:15:29 AM}
    Date: {5/27/2021 12:00:00 AM}
    Day: 27
    DayOfWeek: System.DayOfWeek.Thursday
    DayOfYear: 147
    Hour: 11
    Kind: System.DateTimeKind.Local
    Millisecond: 799
    Minute: 15
    Month: 5
    Second: 29
    Ticks: 637577109297996840
    TimeOfDay: {11:15:29.7996840}
    Year: 2021
Static members:
Non-Public members:
```

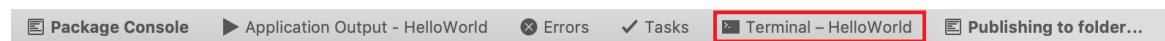
The **Locals** window displays the values of variables that are defined in the currently executing method. The values of the variables that you just changed are updated in the **Locals** window.

Breakpoints		Locals	Watch	Threads
Name		Type		
args	{string[0]}	string[]		
name	Gracie	string		
▶ currentDate	{5/27/2021 11:15:29 AM}	System.DateTime		

4. Press ⌘← (command+enter) to continue debugging.

The values displayed in the terminal correspond to the changes you made in the **Immediate** window.

If you don't see the Terminal, select **Terminal - HelloWorld** in the bottom navigation bar.



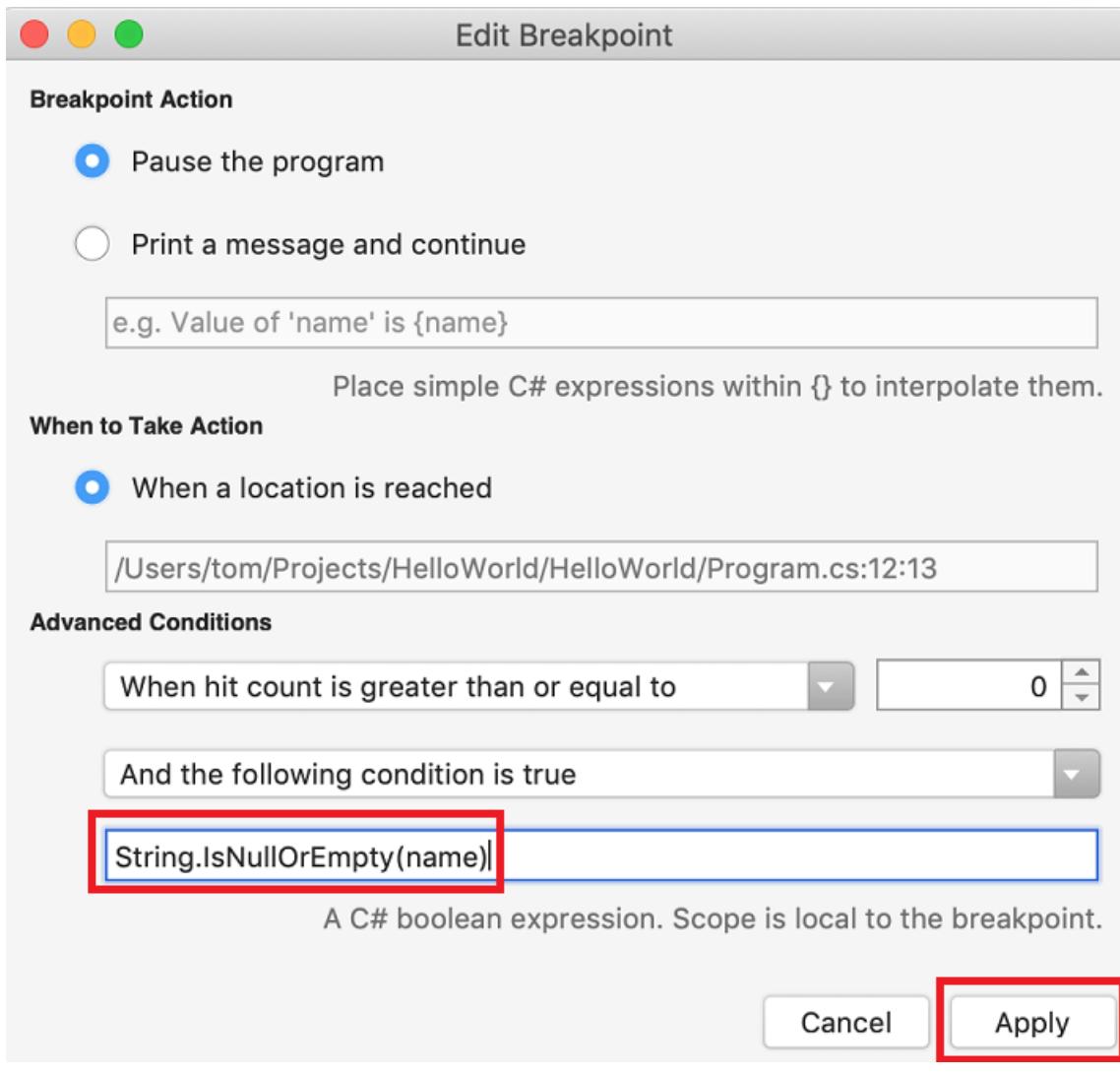
5. Press any key to exit the program.
6. Close the terminal window.

Set a conditional breakpoint

The program displays a string that the user enters. What happens if the user doesn't enter anything? You can test this with a useful debugging feature called a *conditional breakpoint*.

1. **ctrl-click** on the red dot that represents the breakpoint. In the context menu, select **Edit Breakpoint**.
2. In the **Edit Breakpoint** dialog, enter the following code in the field that follows **And the following condition is true**, and select **Apply**.

```
String.IsNullOrEmpty(name)
```



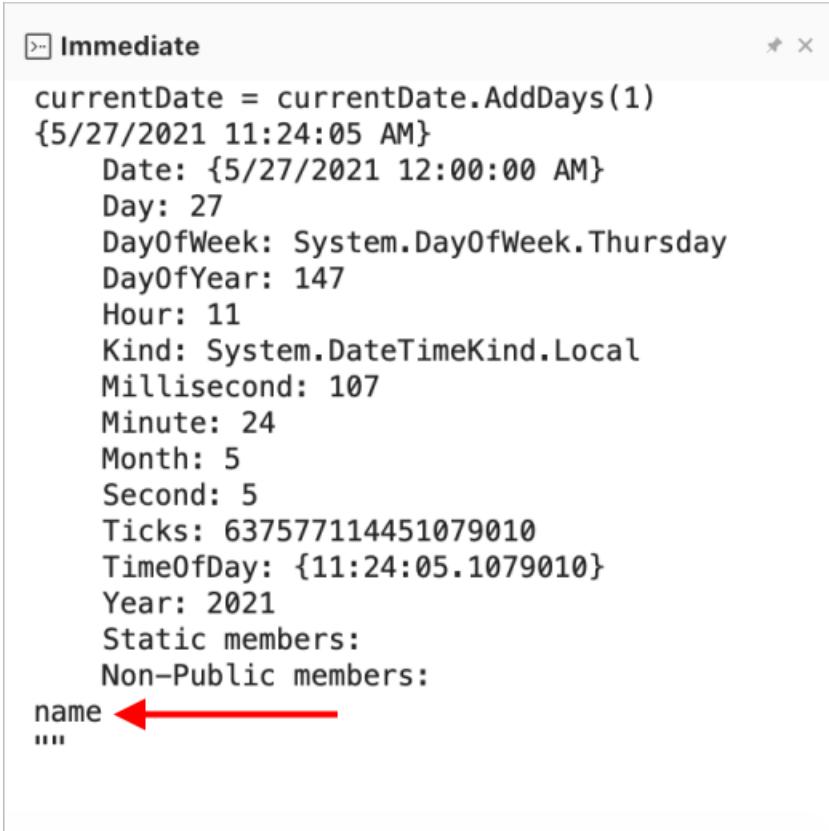
Each time the breakpoint is hit, the debugger calls the `String.IsNullOrEmpty(name)` method, and it breaks on this line only if the method call returns `true`.

Instead of a conditional expression, you can specify a *hit count*, which interrupts program execution before a statement is executed a specified number of times.

3. Press ⌘← (command+enter) to start debugging.
4. In the terminal window, press enter when prompted to enter your name.

Because the condition you specified (`name` is either `null` or `String.Empty`) has been satisfied, program execution stops when it reaches the breakpoint.

5. Select the **Locals** window, which shows the values of variables that are local to the currently executing method. In this case, `Main` is the currently executing method. Observe that the value of the `name` variable is `""`, that is, `String.Empty`.
6. You can also see that the value is an empty string by entering the `name` variable name in the **Immediate** window and pressing enter.



```
... Immediate ...
currentDate = currentDate.AddDays(1)
{5/27/2021 11:24:05 AM}
  Date: {5/27/2021 12:00:00 AM}
  Day: 27
  DayOfWeek: System.DayOfWeek.Thursday
  DayOfYear: 147
  Hour: 11
  Kind: System.DateTimeKind.Local
  Millisecond: 107
  Minute: 24
  Month: 5
  Second: 5
  Ticks: 637577114451079010
  TimeOfDay: {11:24:05.1079010}
  Year: 2021
  Static members:
  Non-Public members:
name ←
""
```

7. Press ⌘← (command+enter) to continue debugging.
8. In the terminal window, press any key to exit the program.
9. Close the terminal window.
10. Clear the breakpoint by clicking on the red dot in the left margin of the code window. Another way to clear a breakpoint is by choosing Run > Toggle Breakpoint while the line of code is selected.

Step through a program

Visual Studio also allows you to step line by line through a program and monitor its execution. Ordinarily, you'd set a breakpoint and follow program flow through a small part of your program code. Since this program is small, you can step through the entire program.

1. Set a breakpoint on the curly brace that marks the start of the `Main` method (press command+\).

2. Press ⌘← (command+enter) to start debugging.

Visual Studio stops on the line with the breakpoint.

3. Press ⇧⌘I (shift+command+I) or select Run > Step Into to advance one line.

Visual Studio highlights and displays an arrow beside the next line of execution.

```
1  using System;
2
3  namespace HelloWorld
4  {
5      class Program
6      {
7          static void Main(string[] args)
8          {
9              Console.WriteLine("What is your name?");
10             var name = Console.ReadLine();
11             var currentDate = DateTime.Now;
12             Console.WriteLine($"{Environment.NewLine}Hello, {name}, on {currentDate}");
13             Console.WriteLine($"{Environment.NewLine}Press any key to exit...");
14             Console.ReadKey(true);
15         }
16     }
17 }
```

At this point, the **Locals** window shows that the `args` array is empty, and `name` and `currentDate` have default values. In addition, Visual Studio has opened a blank terminal.

4. Press $\text{Shift} + \text{Command} + \text{I}$ (`shift+command+I`).

Visual Studio highlights the statement that includes the `name` variable assignment. The **Locals** window shows that `name` is `null`, and the terminal displays the string "What is your name?".

5. Respond to the prompt by entering a string in the console window and pressing enter.

6. Press $\text{Shift} + \text{Command} + \text{I}$ (`shift+command+I`).

Visual Studio highlights the statement that includes the `currentDate` variable assignment. The **Locals** window shows the value returned by the call to the `Console.ReadLine` method. The terminal displays the string you entered at the prompt.

7. Press $\text{Shift} + \text{Command} + \text{I}$ (`shift+command+I`).

The **Locals** window shows the value of the `currentDate` variable after the assignment from the `DateTime.Now` property. The terminal is unchanged.

8. Press $\text{Shift} + \text{Command} + \text{I}$ (`shift+command+I`).

Visual Studio calls the `Console.WriteLine(String, Object, Object)` method. The terminal displays the formatted string.

9. Press $\text{Shift} + \text{Command} + \text{U}$ (`shift+command+U`) or select **Run > Step Out**.

The terminal displays a message and waits for you to press a key.

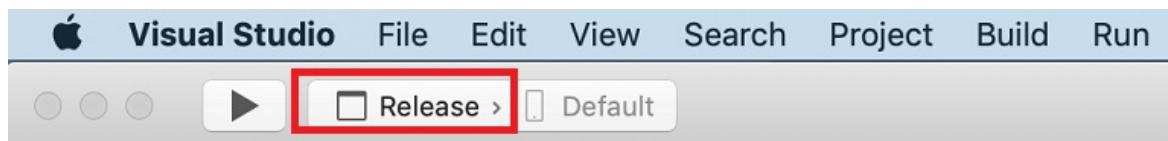
10. Press any key to exit the program.

Use Release build configuration

Once you've tested the Debug version of your application, you should also compile and test the Release version. The Release version incorporates compiler optimizations that can negatively affect the behavior of an application. For example, compiler optimizations that are designed to improve performance can create race conditions in multithreaded applications.

To build and test the Release version of the console application, do the following steps:

1. Change the build configuration on the toolbar from **Debug** to **Release**.



2. Press $\text{option} + \text{command} + \text{enter}$ (option+command+enter) to run without debugging.

Next steps

In this tutorial, you used Visual Studio debugging tools. In the next tutorial, you publish a deployable version of the app.

[Publish a .NET console application using Visual Studio for Mac](#)

Tutorial: Publish a .NET console application using Visual Studio for Mac

9/20/2022 • 2 minutes to read • [Edit Online](#)

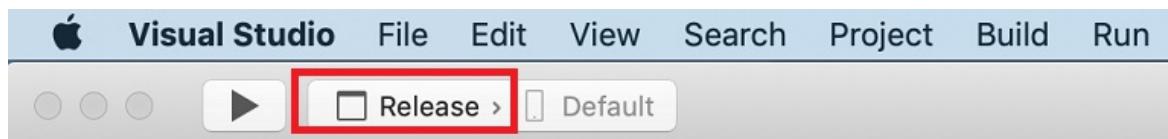
This tutorial shows how to publish a console app so that other users can run it. Publishing creates the set of files that are needed to run your application. To deploy the files, copy them to the target machine.

Prerequisites

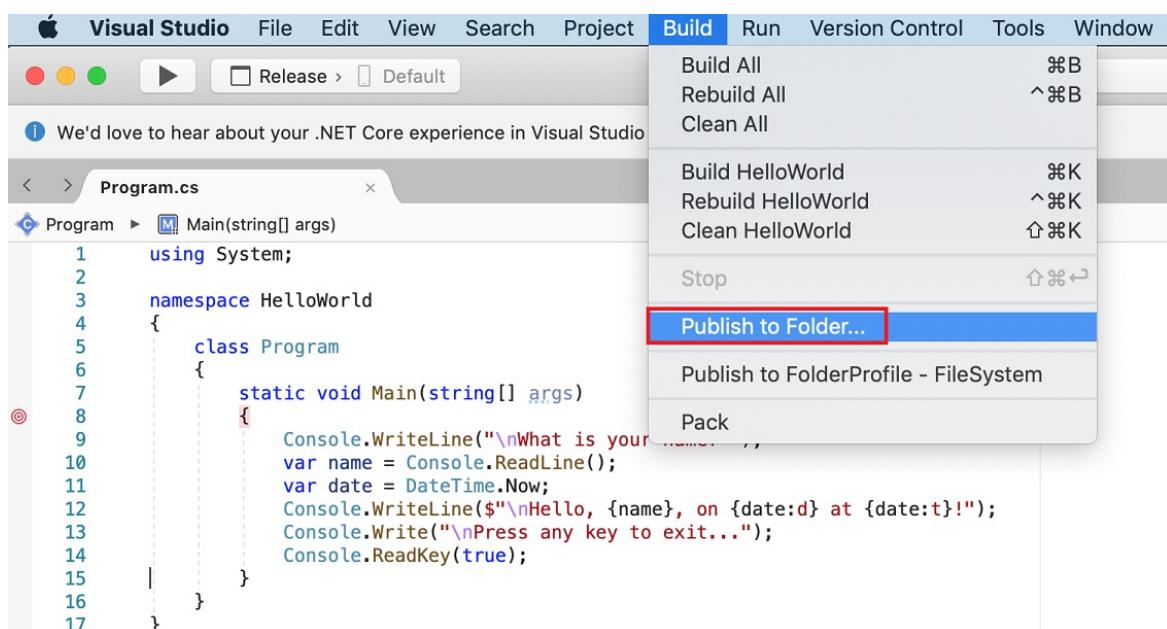
- This tutorial works with the console app that you create in [Create a .NET console application using Visual Studio for Mac](#).

Publish the app

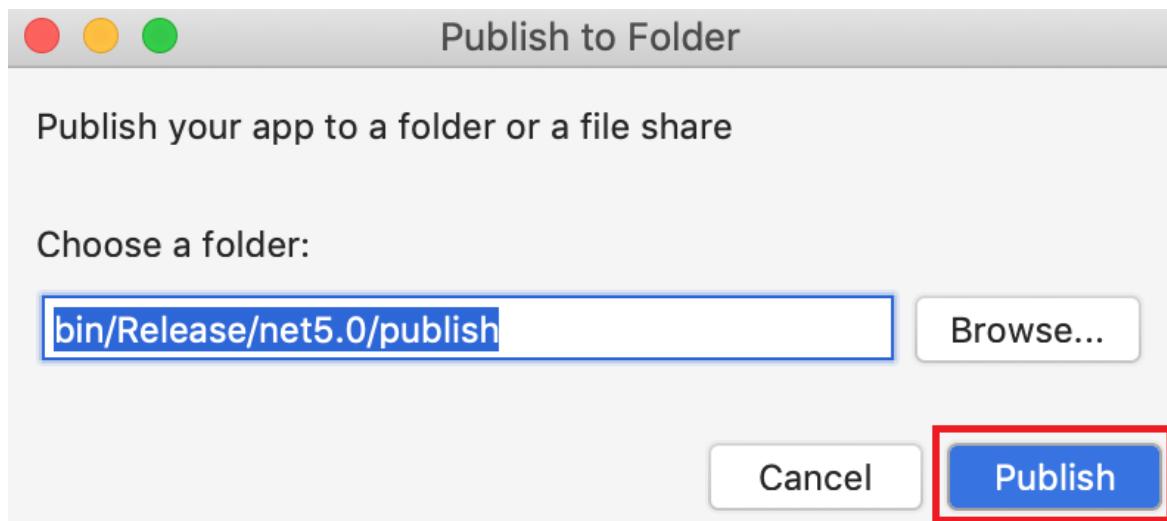
1. Start Visual Studio for Mac.
2. Open the HelloWorld project that you created in [Create a .NET console application using Visual Studio for Mac](#).
3. Make sure that Visual Studio is building the Release version of your application. If necessary, change the build configuration setting on the toolbar from **Debug** to **Release**.



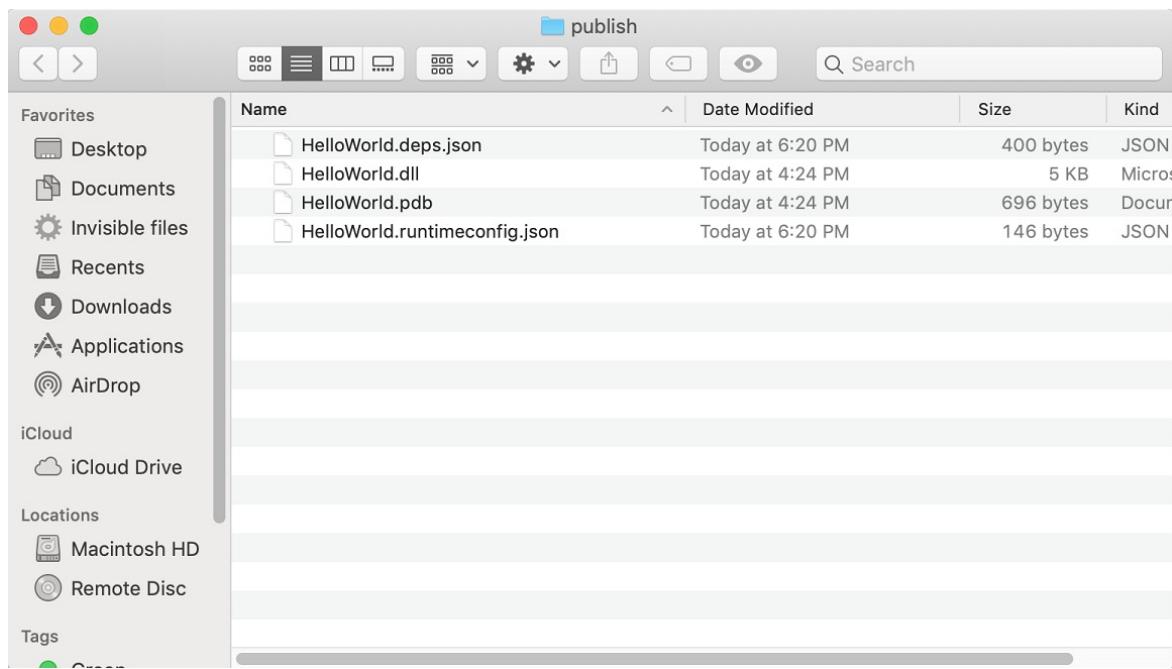
4. From the main menu, choose **Build > Publish to Folder....**



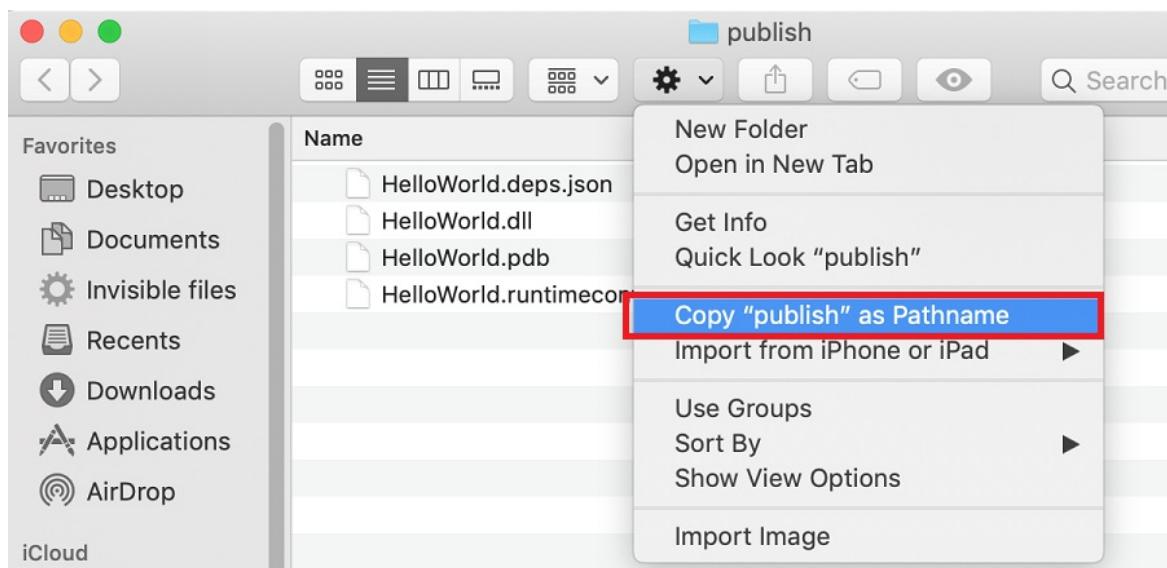
5. In the **Publish to Folder** dialog, select **Publish**.



The publish folder opens, showing the files that were created.



6. Select the gear icon, and select Copy "publish" as Pathname from the context menu.



Inspect the files

The publishing process creates a framework-dependent deployment, which is a type of deployment where the

published application runs on a machine that has the .NET runtime installed. Users can run the published app by running the `dotnet HelloWorld.dll` command from a command prompt.

As the preceding image shows, the published output includes the following files:

- *HelloWorlddeps.json*

This is the application's runtime dependencies file. It defines the .NET components and the libraries (including the dynamic link library that contains your application) needed to run the app. For more information, see [Runtime configuration files](#).

- *HelloWorld.dll*

This is the [framework-dependent deployment](#) version of the application. To execute this dynamic link library, enter `dotnet HelloWorld.dll` at a command prompt. This method of running the app works on any platform that has the .NET runtime installed.

- *HelloWorld.pdb* (optional for deployment)

This is the debug symbols file. You aren't required to deploy this file along with your application, although you should save it in the event that you need to debug the published version of your application.

- *HelloWorld.runtimeconfig.json*

This is the application's runtime configuration file. It identifies the version of .NET that your application was built to run on. You can also add configuration options to it. For more information, see [.NET runtime configuration settings](#).

Run the published app

1. Open a terminal and navigate to the *publish* folder. To do that, enter `cd` and then paste the path that you copied earlier. For example:

```
cd ~/Projects/HelloWorld/HelloWorld/bin/Release/net5.0/publish/
```

2. Run the app by using the `dotnet` command:

- a. Enter `dotnet HelloWorld.dll` and press enter.
- b. Enter a name in response to the prompt, and press any key to exit.

Additional resources

- [.NET application deployment](#)

Next steps

In this tutorial, you published a console app. In the next tutorial, you create a class library.

[Create a .NET library using Visual Studio for Mac](#)

Tutorial: Create a .NET class library using Visual Studio for Mac

9/20/2022 • 4 minutes to read • [Edit Online](#)

In this tutorial, you create a class library that contains a single string-handling method.

A *class library* defines types and methods that are called by an application. If the library targets .NET Standard 2.0, it can be called by any .NET implementation (including .NET Framework) that supports .NET Standard 2.0. If the library targets .NET 5, it can be called by any application that targets .NET 5. This tutorial shows how to target .NET 5.

NOTE

Your feedback is highly valued. There are two ways you can provide feedback to the development team on Visual Studio for Mac:

- In Visual Studio for Mac, select **Help > Report a Problem** from the menu or **Report a Problem** from the Welcome screen, which opens a window for filing a bug report. You can track your feedback in the [Developer Community](#) portal.
- To make a suggestion, select **Help > Provide a Suggestion** from the menu or **Provide a Suggestion** from the Welcome screen, which takes you to the [Visual Studio for Mac Developer Community webpage](#).

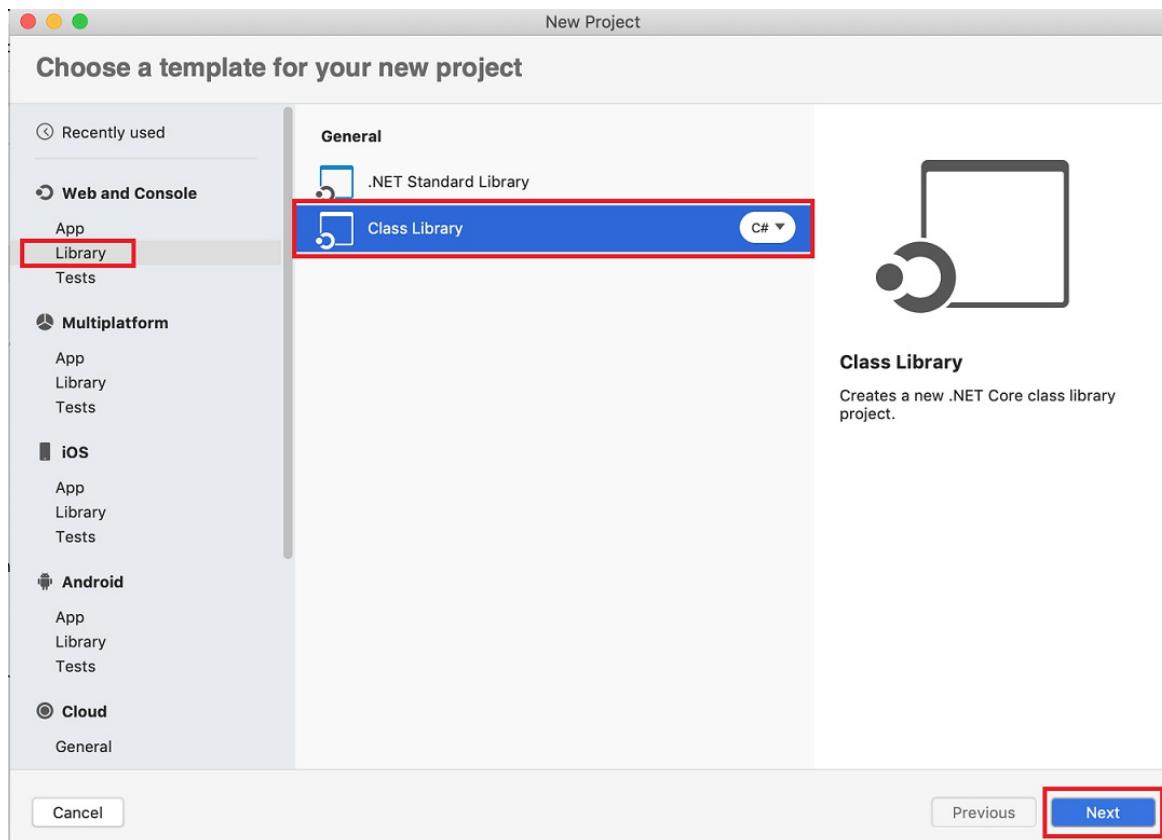
Prerequisites

- [Install Visual Studio for Mac version 8.8 or later](#). Select the option to install .NET Core. Installing Xamarin is optional for .NET development. For more information, see the following resources:
 - [Tutorial: Install Visual Studio for Mac](#).
 - [Supported macOS versions](#).
 - [.NET versions supported by Visual Studio for Mac](#).

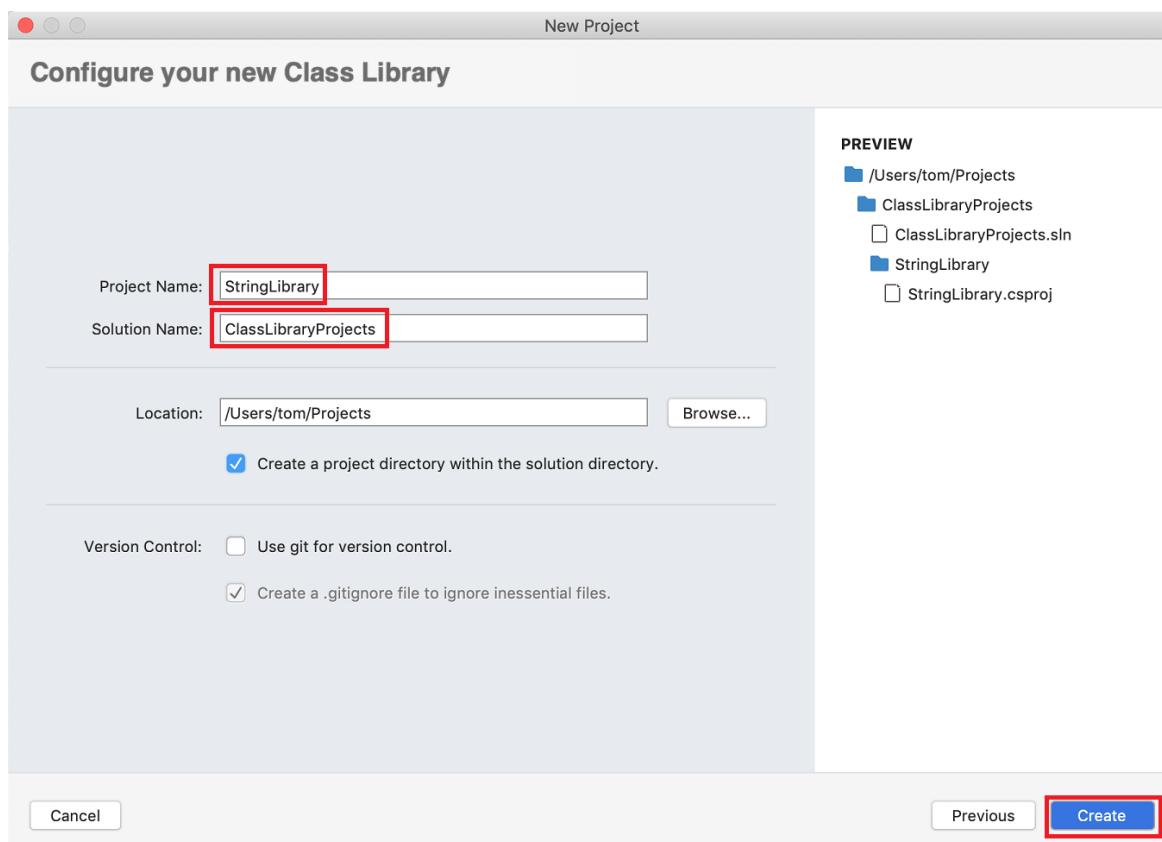
Create a solution with a class library project

A Visual Studio solution serves as a container for one or more projects. Create a solution and a class library project in the solution. You'll add additional, related projects to the same solution later.

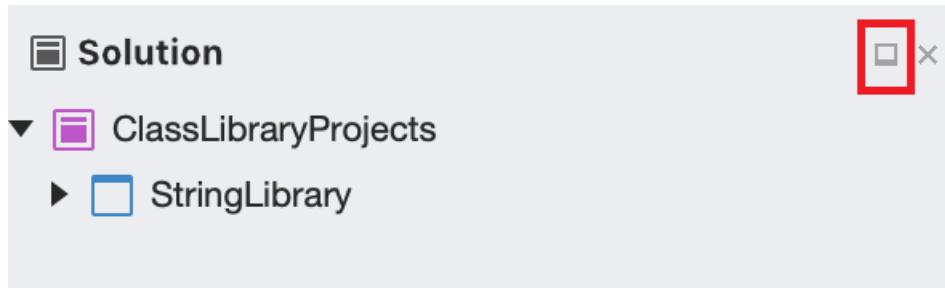
1. Start Visual Studio for Mac.
2. In the start window, select **New Project**.
3. In the **Choose a template for your new project** dialog select **Web and Console > Library > Class Library**, and then select **Next**.



4. In the **Configure your new Class Library** dialog, choose **.NET 5.0**, and select **Next**.
5. Name the project "StringLibrary" and the solution "ClassLibraryProjects". Leave **Create a project directory within the solution directory** selected. Select **Create**.



6. From the main menu, select **View > Solution**, and select the dock icon to keep the pad open.



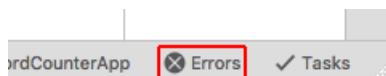
7. In the **Solution** pad, expand the `StringLibrary` node to reveal the class file provided by the template, `Class1.cs`. **ctrl**-click the file, select **Rename** from the context menu, and rename the file to `StringLibrary.cs`. Open the file and replace the contents with the following code:

```
using System;

namespace UtilityLibraries
{
    public static class StringLibrary
    {
        public static bool StartsWithUpper(this string str)
        {
            if (string.IsNullOrWhiteSpace(str))
                return false;

            char ch = str[0];
            return char.IsUpper(ch);
        }
    }
}
```

8. Press **⌘S** (command+S) to save the file.
9. Select **Errors** in the margin at the bottom of the IDE window to open the **Errors** panel. Select the **Build Output** button.



10. Select **Build > Build All** from the menu.

The solution builds. The build output panel shows that the build is successful.

```
Build succeeded.
  0 Warning(s)
  0 Error(s)

Time Elapsed 00:00:01.50

===== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped =====

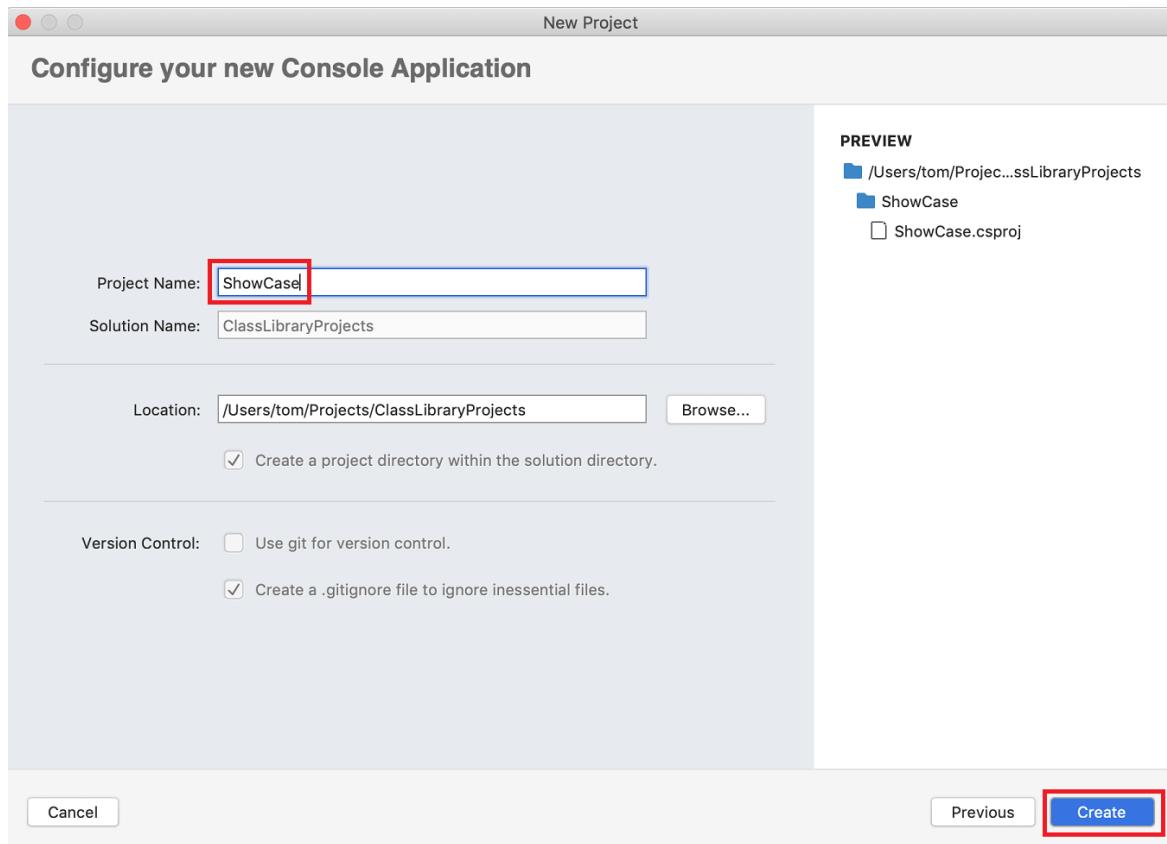
Build successful.
```

Add a console app to the solution

Add a console application that uses the class library. The app will prompt the user to enter a string and report whether the string begins with an uppercase character.

1. In the **Solution** pad, **ctrl**-click the `ClassLibraryProjects` solution. Add a new **Console Application** project by selecting the template from the **Web and Console > App** templates, and select **Next**.
2. Select **.NET 5.0** as the **Target Framework** and select **Next**.

3. Name the project **ShowCase**. Select **Create** to create the project in the solution.



4. Open the *Program.cs* file. Replace the code with the following code:

```

using System;
using UtilityLibraries;

class Program
{
    static void Main(string[] args)
    {
        int row = 0;

        do
        {
            if (row == 0 || row >= 25)
                ResetConsole();

            string? input = Console.ReadLine();
            if (string.IsNullOrEmpty(input)) break;
            Console.WriteLine($"Input: {input} {"Begins with uppercase? ",30}: " +
                $"{{(input.StartsWithUpper() ? "Yes" : "No")}{Environment.NewLine}}");
            row += 3;
        } while (true);
        return;
    }

    // Declare a ResetConsole local method
    void ResetConsole()
    {
        if (row > 0)
        {
            Console.WriteLine("Press any key to continue...");
            Console.ReadKey();
        }
        Console.Clear();
        Console.WriteLine($"{Environment.NewLine}Press <Enter> only to exit; otherwise, enter a
string and press <Enter>:{Environment.NewLine}");
        row = 3;
    }
}

```

The program prompts the user to enter a string. It indicates whether the string starts with an uppercase character. If the user presses the enter key without entering a string, the application ends, and the console window closes.

The code uses the `row` variable to maintain a count of the number of rows of data written to the console window. Whenever it's greater than or equal to 25, the code clears the console window and displays a message to the user.

Add a project reference

Initially, the new console app project doesn't have access to the class library. To allow it to call methods in the class library, create a project reference to the class library project.

1. In the **Solutions** pad, **ctrl-click** the **Dependencies** node of the new **ShowCase** project. In the context menu, select **Add Reference**.
2. In the **References** dialog, select **StringLibrary** and select **OK**.

Run the app

1. **ctrl-click** the **ShowCase** project and select **Run project** from the context menu.
2. Try out the program by entering strings and pressing enter, then press enter to exit.

Terminal – ShowCase

```
Press <Enter> only to exit; otherwise, enter a string and press <Enter>:
```

```
Begins with uppercase
```

```
Input: Begins with uppercase
```

```
Begins with uppercase? : Yes
```

```
begins with lowercase
```

```
Input: begins with lowercase
```

```
Begins with uppercase? : No
```

Additional resources

- [Develop libraries with the .NET CLI](#)
- [Visual Studio 2019 for Mac Release Notes](#)
- [.NET Standard versions and the platforms they support.](#)

Next steps

In this tutorial, you created a solution and a library project, and added a console app project that uses the library. In the next tutorial, you add a unit test project to the solution.

[Test a .NET class library using Visual Studio for Mac](#)

Test a .NET class library using Visual Studio

9/20/2022 • 7 minutes to read • [Edit Online](#)

This tutorial shows how to automate unit testing by adding a test project to a solution.

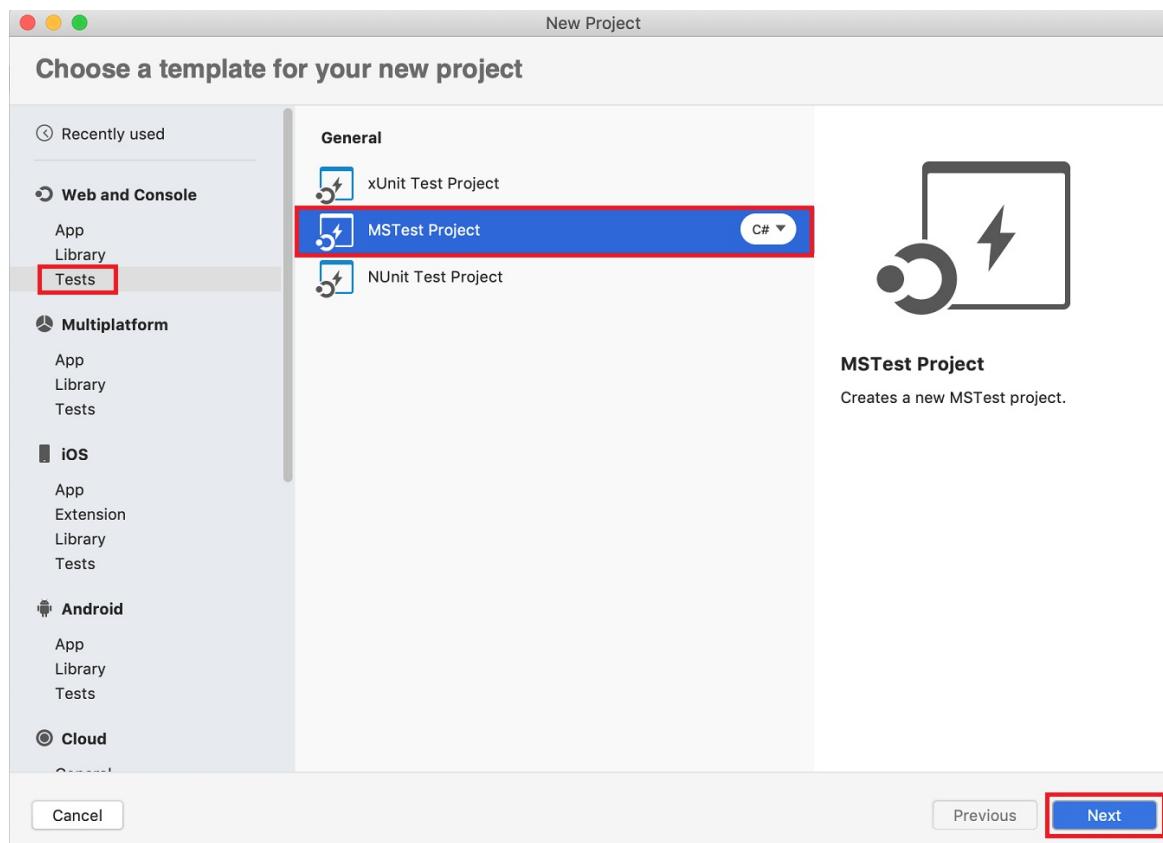
Prerequisites

- This tutorial works with the solution that you create in [Create a .NET class library using Visual Studio for Mac](#).

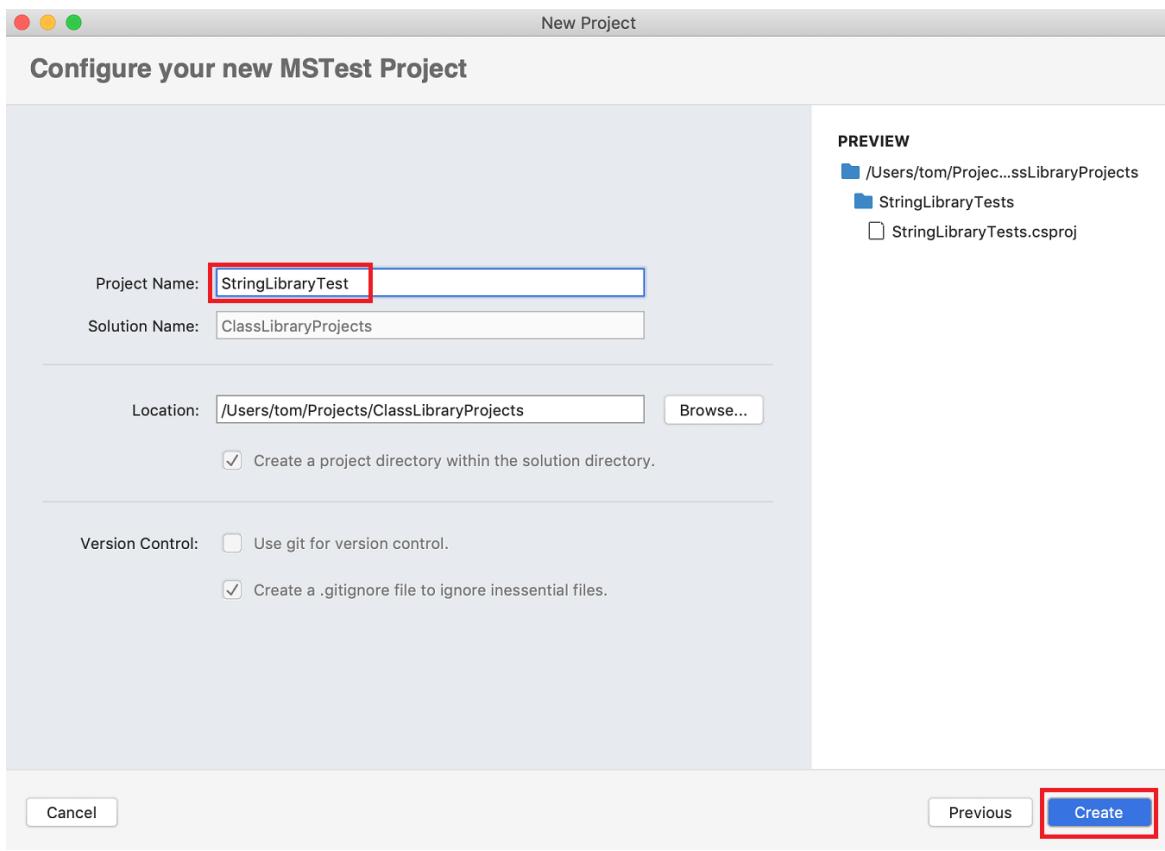
Create a unit test project

Unit tests provide automated software testing during your development and publishing. [MSTest](#) is one of three test frameworks you can choose from. The others are [xUnit](#) and [nUnit](#).

1. Start Visual Studio for Mac.
2. Open the `ClassLibraryProjects` solution you created in [Create a .NET class library using Visual Studio for Mac](#).
3. In the **Solution** pad, **ctrl**-click the `ClassLibraryProjects` solution and select **Add > New Project**.
4. In the **New Project** dialog, select **Tests** from the **Web and Console** node. Select the **MSTest Project** followed by **Next**.



5. Select .NET 5.0 as the **Target Framework** and select **Next**.
6. Name the new project "StringLibraryTest" and select **Create**.



Visual Studio creates a class file with the following code:

```
using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace StringLibraryTest
{
    [TestClass]
    public class UnitTest1
    {
        [TestMethod]
        public void TestMethod1()
        {
        }
    }
}
```

The source code created by the unit test template does the following:

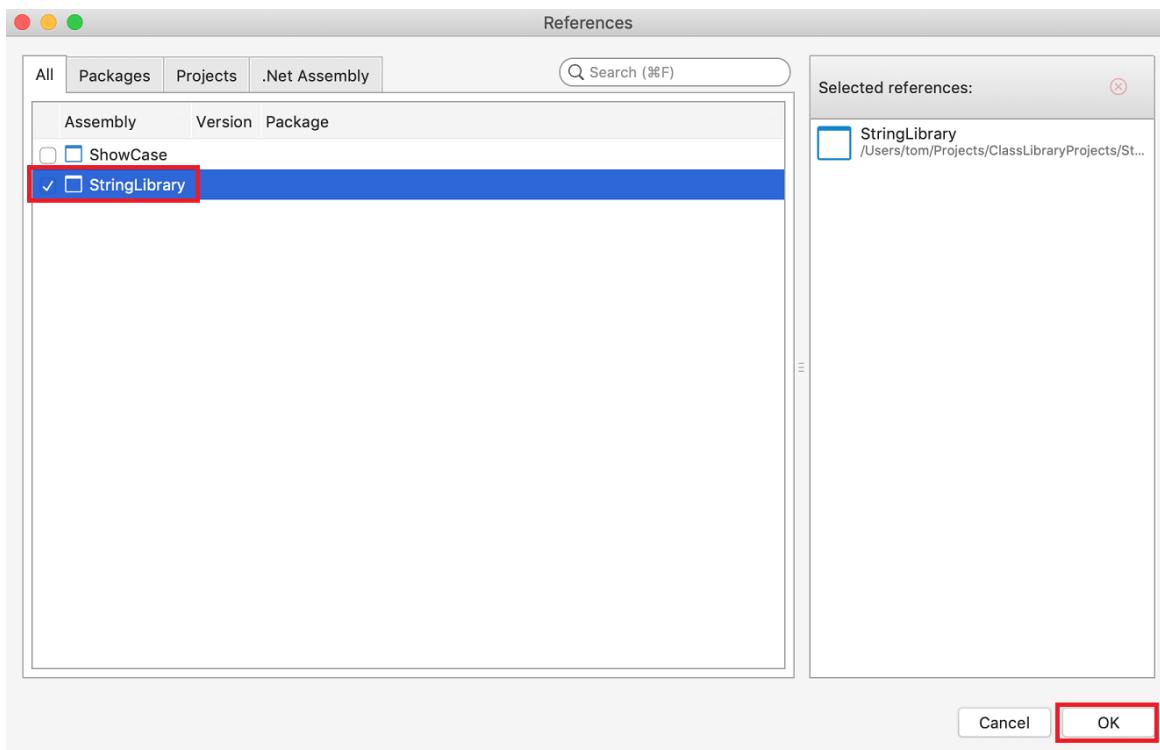
- It imports the `Microsoft.VisualStudio.TestTools.UnitTesting` namespace, which contains the types used for unit testing.
- It applies the `TestClassAttribute` attribute to the `UnitTest1` class.
- It applies the `TestMethodAttribute` attribute to `TestMethod1`.

Each method tagged with `[TestMethod]` in a test class tagged with `[TestClass]` is executed automatically when the unit test is run.

Add a project reference

For the test project to work with the `StringLibrary` class, add a reference to the `StringLibrary` project.

1. In the **Solution** pad, **ctrl+click** **Dependencies** under `StringLibraryTest`. Select **Add Reference** from the context menu.
2. In the **References** dialog, select the `StringLibrary` project. Select **OK**.



Add and run unit test methods

When Visual Studio runs a unit test, it executes each method that is marked with the `TestMethodAttribute` attribute in a class that is marked with the `TestClassAttribute` attribute. A test method ends when the first failure is found or when all tests contained in the method have succeeded.

The most common tests call members of the `Assert` class. Many assert methods include at least two parameters, one of which is the expected test result and the other of which is the actual test result. Some of the `Assert` class's most frequently called methods are shown in the following table:

ASSERT METHODS	FUNCTION
<code>Assert.AreEqual</code>	Vерифицирует, что два значения или объекта равны. Тестовый метод fails, если значения или объекты не равны.
<code>Assert.AreSame</code>	Верифицирует, что два объекта указывают на один и тот же объект. Тестовый метод fails, если объекты указывают на разные объекты.
<code>Assert.IsFalse</code>	Верифицирует, что условие равно <code>false</code> . Тестовый метод fails, если условие равно <code>true</code> .
<code>Assert.IsNotNull</code>	Верифицирует, что объект не равен <code>null</code> . Тестовый метод fails, если объект равен <code>null</code> .

You can also use the `Assert.ThrowsException` method in a test method to indicate the type of exception it's expected to throw. The test fails if the specified exception isn't thrown.

In testing the `StringLibrary.StartsWithUpper` method, you want to provide a number of strings that begin with an uppercase character. You expect the method to return `true` in these cases, so you can call the `Assert.IsTrue` method. Similarly, you want to provide a number of strings that begin with something other than an uppercase character. You expect the method to return `false` in these cases, so you can call the `Assert.IsFalse` method.

Since your library method handles strings, you also want to make sure that it successfully handles an `empty string` (`String.Empty`), a valid string that has no characters and whose `Length` is 0, and a `null` string that hasn't

been initialized. You can call `StartsWithUpper` directly as a static method and pass a single `String` argument. Or you can call `StartsWithUpper` as an extension method on a `string` variable assigned to `null`.

You'll define three methods, each of which calls an `Assert` method for each element in a string array. You'll call a method overload that lets you specify an error message to be displayed in case of test failure. The message identifies the string that caused the failure.

To create the test methods:

1. Open the `UnitTest1.cs` file and replace the code with the following code:

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using UtilityLibraries;

namespace StringLibraryTest
{
    [TestClass]
    public class UnitTest1
    {
        [TestMethod]
        public void TestStartsWithUpper()
        {
            // Tests that we expect to return true.
            string[] words = { "Alphabet", "Zebra", "ABC", "Αθήνα", "Москва" };
            foreach (var word in words)
            {
                bool result = word.StartsWithUpper();
                Assert.IsTrue(result,
                    string.Format("Expected for '{0}': true; Actual: {1}",
                        word, result));
            }
        }

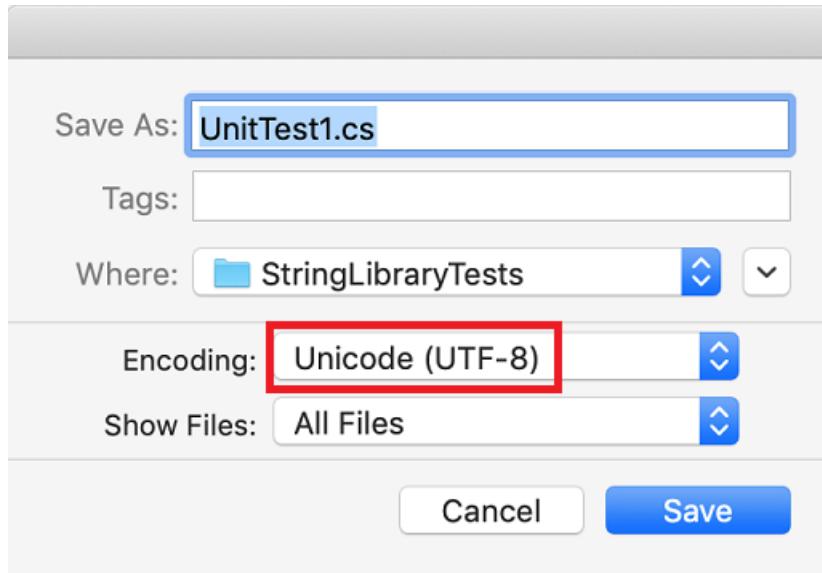
        [TestMethod]
        public void TestDoesNotStartWithUpper()
        {
            // Tests that we expect to return false.
            string[] words = { "alphabet", "zebra", "abc", "αυτοκινητοβιομηχανία", "государство",
                "1234", ".", ";", " " };
            foreach (var word in words)
            {
                bool result = word.StartsWithUpper();
                Assert.IsFalse(result,
                    string.Format("Expected for '{0}': false; Actual: {1}",
                        word, result));
            }
        }

        [TestMethod]
        public void DirectCallWithNullOrEmpty()
        {
            // Tests that we expect to return false.
            string?[] words = { string.Empty, null };
            foreach (var word in words)
            {
                bool result = StringLibrary.StartsWithUpper(word ?? string.Empty);
                Assert.IsFalse(result,
                    string.Format("Expected for '{0}': false; Actual: {1}",
                        word == null ? "<null>" : word, result));
            }
        }
    }
}
```

The test of uppercase characters in the `TestStartsWithUpper` method includes the Greek capital letter

alpha (U+0391) and the Cyrillic capital letter EM (U+041C). The test of lowercase characters in the `TestDoesNotStartWithUpper` method includes the Greek small letter alpha (U+03B1) and the Cyrillic small letter Ghe (U+0433).

2. On the menu bar, select **File > Save As**. In the dialog, make sure that **Encoding** is set to **Unicode (UTF-8)**.

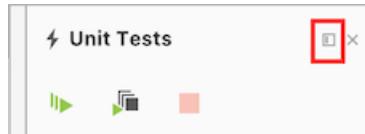


3. When you're asked if you want to replace the existing file, select **Replace**.

If you fail to save your source code as a UTF8-encoded file, Visual Studio may save it as an ASCII file. When that happens, the runtime doesn't accurately decode the UTF8 characters outside of the ASCII range, and the test results won't be correct.

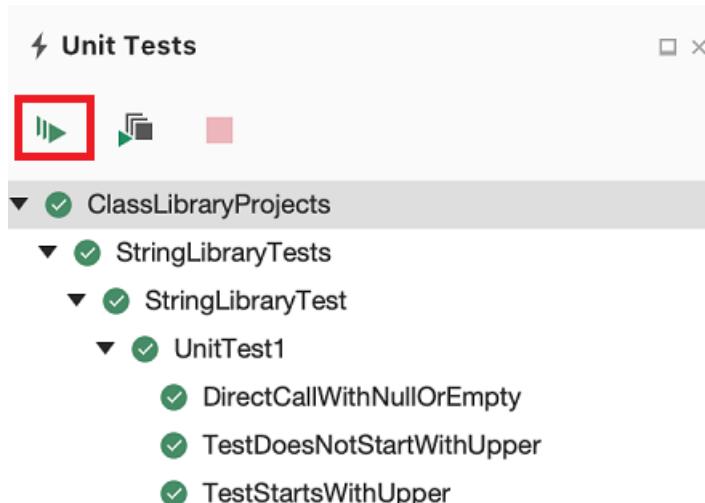
4. Open the **Unit Tests** panel on the right side of the screen. Select **View > Tests** from the menu.

5. Click the **Dock** icon to keep the panel open.



6. Click the **Run All** button.

All tests pass.



Handle test failures

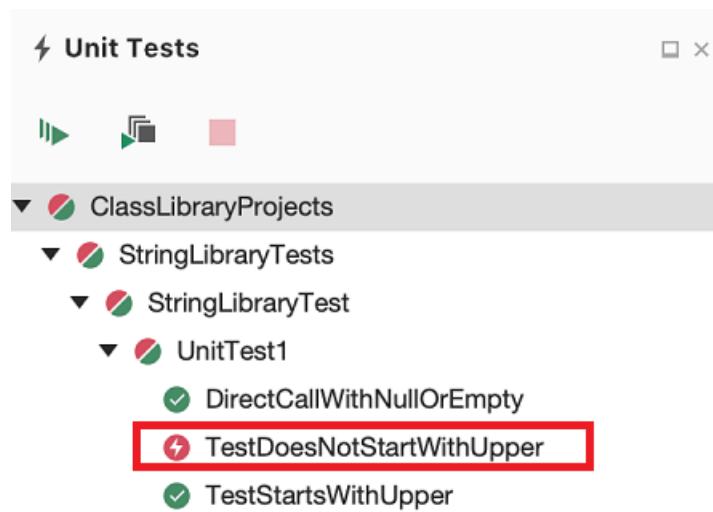
If you're doing test-driven development (TDD), you write tests first and they fail the first time you run them. Then you add code to the app that makes the test succeed. For this tutorial, you created the test after writing the app code that it validates, so you haven't seen the test fail. To validate that a test fails when you expect it to fail, add an invalid value to the test input.

1. Modify the `words` array in the `TestDoesNotStartWithUpper` method to include the string "Error". You don't need to save the file because Visual Studio automatically saves open files when a solution is built to run tests.

```
string[] words = { "alphabet", "Error", "zebra", "abc", "αυτοκινητοβιομηχανία", "государство",  
    "1234", ".", ";" };
```

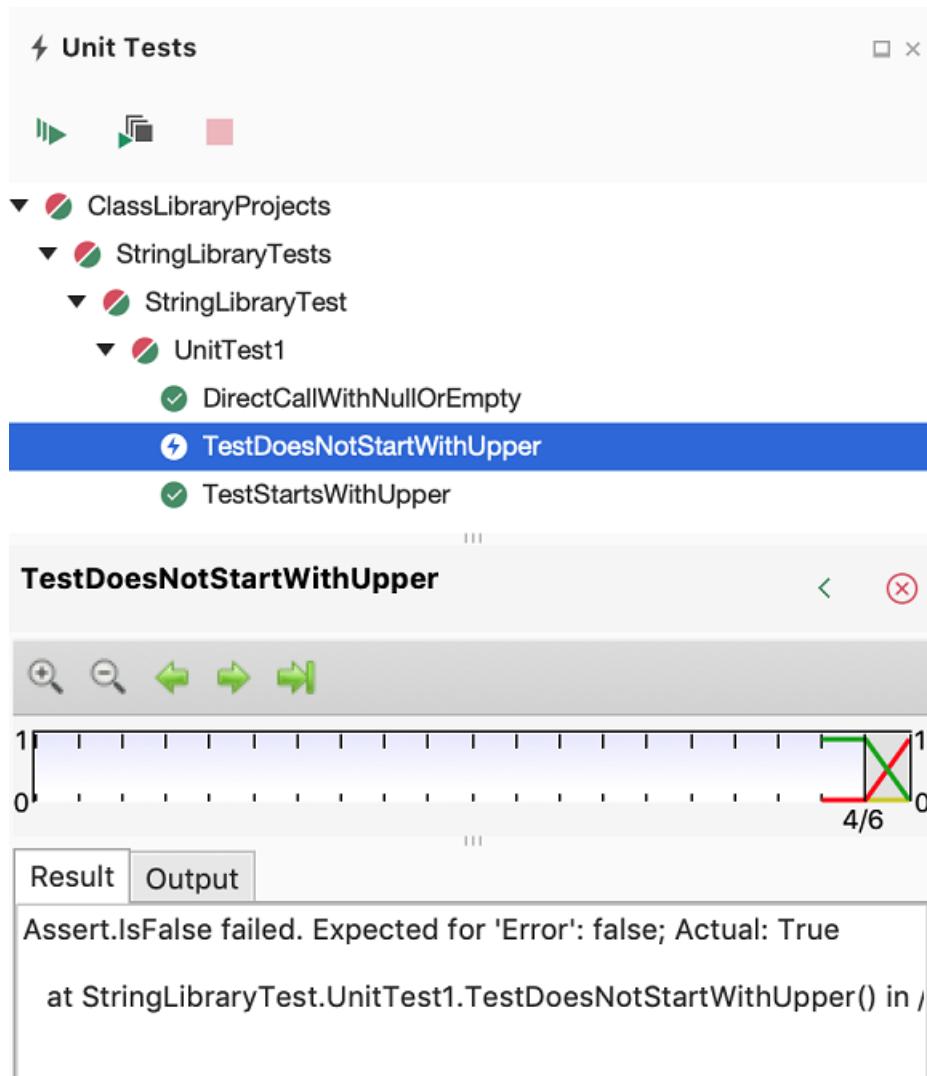
2. Run the tests again.

This time, the **Test Explorer** window indicates that two tests succeeded and one failed.



3. **ctrl-click** the failed test, `TestDoesNotStartWithUpper`, and select **Show Results Pad** from the context menu.

The **Results** pad displays the message produced by the assert: "Assert.IsFalse failed. Expected for 'Error': false; actual: True". Because of the failure, no strings in the array after "Error" were tested.



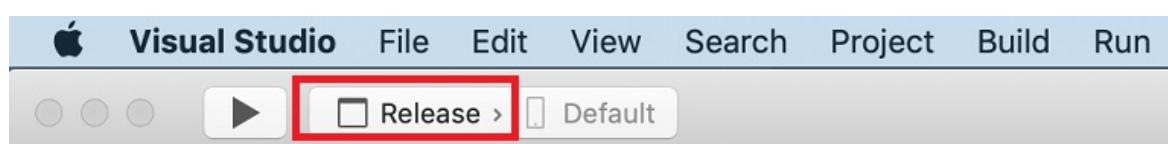
4. Remove the string "Error" that you added in step 1. Rerun the test and the tests pass.

Test the Release version of the library

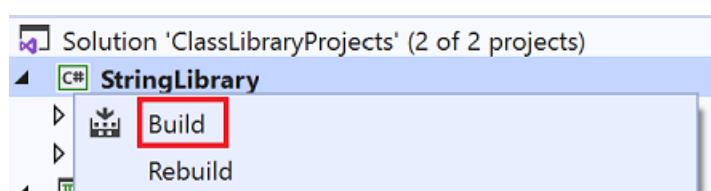
Now that the tests have all passed when running the Debug build of the library, run the tests an additional time against the Release build of the library. A number of factors, including compiler optimizations, can sometimes produce different behavior between Debug and Release builds.

To test the Release build:

1. In the Visual Studio toolbar, change the build configuration from Debug to Release.



2. In the Solution pad, **ctrl-click** the **StringLibrary** project and select **Build** from the context menu to recompile the library.



3. Run the unit tests again.

The tests pass.

Debug tests

If you're using Visual Studio for Mac as your IDE, you can use the same process shown in [Tutorial: Debug a .NET console application using Visual Studio for Mac](#) to debug code using your unit test project. Instead of starting the *ShowCase* app project, **ctrl-click** the **StringLibraryTests** project, and select **Start Debugging Project** from the context menu.

Visual Studio starts the test project with the debugger attached. Execution will stop at any breakpoint you've added to the test project or the underlying library code.

Additional resources

- [Unit testing in .NET](#)

Next steps

In this tutorial, you unit tested a class library. You can make the library available to others by publishing it to [NuGet](#) as a package. To learn how, follow a NuGet tutorial:

[Create and publish a package \(dotnet CLI\)](#)

If you publish a library as a NuGet package, others can install and use it. To learn how, follow a NuGet tutorial:

[Install and use a package in Visual Studio for Mac](#)

A library doesn't have to be distributed as a package. It can be bundled with a console app that uses it. To learn how to publish a console app, see the earlier tutorial in this series:

[Publish a .NET console application using Visual Studio for Mac](#)

Learn .NET and the .NET SDK tools by exploring these tutorials

9/20/2022 • 2 minutes to read • [Edit Online](#)

The following tutorials show how to develop console apps and libraries for .NET Core, .NET 5, and later versions. For other types of applications, see [Tutorials for getting started with .NET](#).

Use Visual Studio

- [Create a console app](#)
- [Debug an app](#)
- [Publish an app](#)
- [Create a class library](#)
- [Unit test a class library](#)
- [Install and use a package](#)
- [Create and publish a package](#)
- [Create an F# console app](#)

Use Visual Studio Code

Choose these tutorials if you want to use Visual Studio Code or some other code editor. All use the CLI for .NET Core development tasks, so all except the debugging tutorial can be used with any code editor.

- [Create a console app](#)
- [Debug an app](#)
- [Publish an app](#)
- [Create a class library](#)
- [Unit test a class library](#)
- [Install and use a package](#)
- [Create and publish a package](#)
- [Create an F# console app](#)

Use Visual Studio for Mac

- [Create a console app](#)
- [Debug an app](#)
- [Publish an app](#)
- [Create a class library](#)
- [Unit test a class library](#)
- [Install and use a package](#)
- [Create an F# console app](#)

Advanced topics

- [How to create libraries](#)
- [Unit test an app with xUnit](#)

- Unit test using C#/VB/F# with NUnit/xUnit/MSTest
- Live unit test with Visual Studio
- Create templates for the CLI
- Create and use tools for the CLI
- Create an app with plugins

What's new in .NET 6

9/20/2022 • 12 minutes to read • [Edit Online](#)

.NET 6 delivers the final parts of the .NET unification plan that started with [.NET 5](#). .NET 6 unifies the SDK, base libraries, and runtime across mobile, desktop, IoT, and cloud apps. In addition to this unification, the .NET 6 ecosystem offers:

- **Simplified development:** Getting started is easy. New language features in [C# 10](#) reduce the amount of code you need to write. And investments in the web stack and minimal APIs make it easy to quickly write smaller, faster microservices.
- **Better performance:** .NET 6 is the fastest full stack web framework, which lowers compute costs if you're running in the cloud.
- **Ultimate productivity:** .NET 6 and [Visual Studio 2022](#) provide hot reload, new git tooling, intelligent code editing, robust diagnostics and testing tools, and better team collaboration.

.NET 6 will be [supported for three years](#) as a long-term support (LTS) release.

Preview features are disabled by default. They are also not supported for use in production and may be removed in a future version. The new [RequiresPreviewFeaturesAttribute](#) is used to annotate preview APIs, and a corresponding analyzer alerts you if you're using these preview APIs.

.NET 6 is supported by Visual Studio 2022 and Visual Studio 2022 for Mac (and later versions).

This article does not cover all of the new features of .NET 6. To see all of the new features, and for further information about the features listed in this article, see the [Announcing .NET 6](#) blog post.

Performance

.NET 6 includes numerous performance improvements. This section lists some of the improvements—in [FileStream](#), [profile-guided optimization](#), and [AOT compilation](#). For detailed information, see the [Performance improvements in .NET 6](#) blog post.

FileStream

The [System.IO.FileStream](#) type has been rewritten for .NET 6 to provide better performance and reliability on Windows. Now, [FileStream](#) never blocks when created for asynchronous I/O on Windows. For more information, see the [File IO improvements in .NET 6](#) blog post.

Profile-guided optimization

Profile-guided optimization (PGO) is where the JIT compiler generates optimized code in terms of the types and code paths that are most frequently used. .NET 6 introduces *dynamic* PGO. Dynamic PGO works hand-in-hand with tiered compilation to further optimize code based on additional instrumentation that's put in place during tier 0. Dynamic PGO is disabled by default, but you can enable it with the `DOTNET_TieredPGO` environment variable. For more information, see [JIT performance improvements](#).

Crossgen2

.NET 6 introduces Crossgen2, the successor to Crossgen, which has been removed. Crossgen and Crossgen2 are tools that provide ahead-of-time (AOT) compilation to improve the startup time of an app. Crossgen2 is written in C# instead of C++, and can perform analysis and optimization that weren't possible with the previous version. For more information, see [Conversation about Crossgen2](#).

Arm64 support

The .NET 6 release includes support for macOS Arm64 (or "Apple Silicon") and Windows Arm64 operating systems, for both native Arm64 execution and x64 emulation. In addition, the x64 and Arm64 .NET installers now install side by side. For more information, see [.NET Support for macOS 11 and Windows 11 for Arm64 and x64](#).

Hot reload

Hot reload is a feature that lets you modify your app's source code and instantly apply those changes to your running app. The feature's purpose is to increase your productivity by avoiding app restarts between edits. Hot reload is available in Visual Studio 2022 and the `dotnet watch` command-line tool. Hot reload works with most types of .NET apps, and for C#, Visual Basic, and C++ source code. For more information, see the [Hot reload blog post](#).

.NET MAUI

.NET Multi-platform App UI (.NET MAUI) is still in *preview*, with a release candidate coming in the first quarter of 2022 and general availability (GA) in the second quarter of 2022. .NET MAUI makes it possible to build native client apps for desktop and mobile operating systems with a single codebase. For more information, see the [Update on .NET Multi-platform App UI](#) blog post.

C# 10 and templates

C# 10 includes innovations such as `global using` directives, file-scoped namespace declarations, and record structs. For more information, see [What's new in C# 10](#).

In concert with that work, the .NET SDK project templates for C# have been modernized to use some of the new language features:

- `async Main` method
- Top-level statements
- Target-typed new expressions
- `Implicit global using` directives
- File-scoped namespaces
- Nullable reference types

By adding these new language features to the project templates, new code starts with the features enabled. However, existing code isn't affected when you upgrade to .NET 6. For more information about these template changes, see the [.NET SDK: C# project templates modernized](#) blog post.

F# and Visual Basic

F# 6 adds several improvements to the F# language and F# Interactive. For more information, see [What's new in F# 6](#).

Visual Basic has improvements in the Visual Studio experience and Windows Forms project startup.

SDK Workloads

To keep the size of the .NET SDK smaller, some components have been placed in new, optional *SDK workloads*. These components include .NET MAUI and Blazor WebAssembly AOT. If you use Visual Studio, it will take care of installing any SDK workloads that you need. If you use the [.NET CLI](#), you can manage workloads using the new `dotnet workload` commands:

COMMAND	DESCRIPTION
<code>dotnet workload search</code>	Searches for available workloads.
<code>dotnet workload install</code>	Installs a specified workload.
<code>dotnet workload uninstall</code>	Removes a specified workload.
<code>dotnet workload update</code>	Updates installed workloads.
<code>dotnet workload repair</code>	Reinstalls all installed workloads to repair a broken installation.
<code>dotnet workload list</code>	Lists installed workloads.

For more information, see [Optional SDK workloads](#).

System.Text.Json APIs

Many improvements have been made in [System.Text.Json](#) in .NET 6, such that it is now an "industrial strength" serialization solution.

Source generator

.NET 6 adds a new [source generator](#) for [System.Text.Json](#). Source generation works with [JsonSerializer](#) and can be configured in multiple ways. It can improve performance, reduce memory usage, and facilitate assembly trimming. For more information, see [How to choose reflection or source generation in System.Text.Json](#) and [How to use source generation in System.Text.Json](#).

Writeable DOM

A new, writeable document object model (DOM) has been added, which supplements the pre-existing read-only DOM. The new API provides a lightweight serialization alternative for cases when use of plain old CLR object (POCO) types isn't possible. It also allows you to efficiently navigate to a subsection of a large JSON tree and read an array or deserialize a POCO from that subsection. The following new types have been added to support the writeable DOM:

- [JsonNode](#)
- [JsonArray](#)
- [JsonObject](#)
- [JsonValue](#)

For more information, see [JSON DOM choices](#).

IAsyncEnumerable serialization

[System.Text.Json](#) now supports serialization and deserialization with [IAsyncEnumerable<T>](#) instances. Asynchronous serialization methods enumerate any [IAsyncEnumerable<T>](#) instances in an object graph and then serialize them as JSON arrays. For deserialization, the new method [JsonSerializer.DeserializeAsyncEnumerable< TValue >\(Stream, JsonSerializerOptions, CancellationToken\)](#) was added. For more information, see [IAsyncEnumerable serialization](#).

Other new APIs

New serialization interfaces for validation and defaulting values:

- [IJsonOnDeserialized](#)
- [IJsonOnDeserializing](#)

- [IJsonOnSerialized](#)
- [IJsonOnSerializing](#)

For more information, see [Callbacks](#).

New property ordering attribute:

- [JsonPropertyOrderAttribute](#)

For more information, see [Configure the order of serialized properties](#).

New method to write "raw" JSON:

- [Utf8JsonWriter.WriteRawValue](#)

For more information, see [Write Raw JSON](#).

Synchronous serialization and deserialization to a stream:

- [JsonSerializer.Deserialize\(Stream, Type, JsonSerializerOptions\)](#)
- [JsonSerializer.Deserialize\(Stream, Type, JsonSerializerContext\)](#)
- [JsonSerializer.Deserialize< TValue >\(Stream, JsonSerializerOptions\)](#)
- [JsonSerializer.Deserialize< TValue >\(Stream, JsonTypeInfo< TValue >\)](#)
- [JsonSerializer.Serialize\(Stream, Object, Type, JsonSerializerOptions\)](#)
- [JsonSerializer.Serialize\(Stream, Object, Type, JsonSerializerContext\)](#)
- [JsonSerializer.Serialize< TValue >\(Stream, TValue, JsonSerializerOptions\)](#)
- [JsonSerializer.Serialize< TValue >\(Stream, TValue, JsonTypeInfo< TValue >\)](#)

New option to ignore an object when a reference cycle is detected during serialization:

- [ReferenceHandler.IgnoreCycles](#)

For more information, see [Ignore circular references](#).

For more information about serializing and deserializing with `System.Text.Json`, see [JSON serialization and deserialization in .NET](#).

HTTP/3

.NET 6 includes preview support for HTTP/3, a new version of HTTP. HTTP/3 solves some existing functional and performance challenges by using a new underlying connection protocol called QUIC. QUIC establishes connections more quickly, and connections are independent of the IP address, allowing mobile clients to roam between Wi-fi and cellular networks. For more information, see [Use HTTP/3 with HttpClient](#).

ASP.NET Core

ASP.NET Core includes improvements in minimal APIs, ahead-of-time (AOT) compilation for Blazor WebAssembly apps, and single-page apps. In addition, Blazor components can now be rendered from JavaScript and integrated with existing JavaScript based apps. For more information, see [What's new in ASP.NET Core 6](#).

OpenTelemetry

.NET 6 brings improved support for [OpenTelemetry](#), which is a collection of tools, APIs, and SDKs that help you analyze your software's performance and behavior. APIs in the `System.Diagnostics.Metrics` namespace implement the [OpenTelemetry Metrics API specification](#). For example, there are four instrument classes to support different metrics scenarios. The instrument classes are:

- [Counter< T >](#)

- [Histogram<T>](#)
- [ObservableCounter<T>](#)
- [ObservableGauge<T>](#)

Security

.NET 6 adds preview support for two key security mitigations: Control-flow Enforcement Technology (CET) and "write exclusive execute" (W^X).

CET is an Intel technology available in some newer Intel and AMD processors. It adds capabilities to the hardware that protect against some control-flow hijacking attacks. .NET 6 provides support for CET for Windows x64 apps, and you must explicitly enable it. For more information, see [.NET 6 compatibility with Intel CET shadow stacks](#).

W^X is available all operating systems with .NET 6 but only enabled by default on Apple Silicon. W^X blocks the simplest attack path by disallowing memory pages to be writeable and executable at the same time.

IL trimming

Trimming of self-contained deployments is improved. In .NET 5, only unused assemblies were trimmed. .NET 6 adds trimming of unused types and members too. In addition, trim warnings, which alert you to places where trimming may remove code that's used at run time, are now *enabled* by default. For more information, see [Trim self-contained deployments and executables](#).

Code analysis

The .NET 6 SDK includes a handful of new code analyzers that concern API compatibility, platform compatibility, trimming safety, use of span in string concatenation and splitting, faster string APIs, and faster collection APIs. For a full list of new (and removed) analyzers, see [Analyzer releases - .NET 6](#).

Custom platform guards

The [Platform compatibility analyzer](#) recognizes the `Is<Platform>` methods in the [OperatingSystem](#) class, for example, `OperatingSystem.IsWindows()`, as platform guards. To allow for custom platform guards, .NET 6 introduces two new attributes that you can use to annotate fields, properties, or methods with a supported or unsupported platform name:

- [SupportedOSPlatformGuardAttribute](#)
- [UnsupportedOSPlatformGuardAttribute](#)

Windows Forms

[Application.SetDefaultFont\(Font\)](#) is a new method in .NET 6 that sets the default font across your application.

The templates for C# Windows Forms apps have been updated to support `global using` directives, file-scoped namespaces, and nullable reference types. In addition, they include application bootstrap code, which reduces boilerplate code and allows the Windows Forms designer to render the design surface in the preferred font. The bootstrap code is a call to `ApplicationConfiguration.Initialize()`, which is a source-generated method that emits calls to other configuration methods, such as `Application.EnableVisualStyles()`. Additionally, if you set a non-default font via the `ApplicationDefaultFont` MSBuild property, `ApplicationConfiguration.Initialize()` emits a call to `SetDefaultFont(Font)`.

For more information, see the [What's new in Windows Forms](#) blog post.

Source build

The *source tarball*, which contains all the source for the .NET SDK, is now a product of the .NET SDK build. Other organizations, such as Red Hat, can build their own version of the SDK using this source tarball.

Target framework monikers

Additional OS-specific target framework monikers (TFMs) have been added for .NET 6, for example, `net6.0-android`, `net6.0-ios`, and `net6.0-macos`. For more information, see [.NET 5+ OS-specific TFMs](#).

Generic math

In *preview* is the ability to use operators on generic types in .NET 6. .NET 6 introduces numerous interfaces that make use of C# 10's new preview feature, `static abstract` interface members. These interfaces correspond to different operators, for example, `IAdditionOperators` represents the `+` operator. The interfaces are available in the `System.Runtime.Experimental` NuGet package. For more information, see the [Generic math](#) blog post.

NuGet package validation

If you're a NuGet library developer, new [package-validation tooling](#) enables you to validate that your packages are consistent and well-formed. You can determine if:

- There are any breaking changes across package versions.
- The package has the same set of public APIs for all runtime-specific implementations.
- There are any gaps for target-framework or runtime applicability.

For more information, see the [Package Validation](#) blog post.

Reflection APIs

.NET 6 introduces the following new APIs that inspect code and provide nullability information:

- `System.Reflection.NullabilityInfo`
- `System.Reflection.NullabilityInfoContext`
- `System.Reflection.NullabilityState`

These APIs are useful for reflection-based tools and serializers.

Microsoft.Extensions APIs

Several extensions namespaces have improvements in .NET 6, as the following table shows.

NAMESPACE	IMPROVEMENTS
<code>Microsoft.Extensions.DependencyInjection</code>	<code>CreateAsyncScope</code> lets you safely use a <code>using</code> statement for a service provider that registers an <code>IAsyncDisposable</code> service.
<code>Microsoft.Extensions.Hosting</code>	New <code>ConfigureHostOptions</code> methods simplify application setup.

NAMESPACE	IMPROVEMENTS
Microsoft.Extensions.Logging	<p>Microsoft.Extensions.Logging has a new source generator for performant logging APIs. The source generator is triggered if you add the new LoggerMessageAttribute to a <code>partial</code> logging method. At compile time, the generator generates the implementation of the <code>partial</code> method, which is typically faster at run time than existing logging solutions. For more information, see Compile-time logging source generation.</p>

New LINQ APIs

Numerous LINQ methods have been added in .NET 6. Most of the new methods listed in the following table have equivalent methods in the [System.Linq.Queryable](#) type.

METHOD	DESCRIPTION
<code>Enumerable.TryGetNonEnumeratedCount<TSource>(IEnumerable<TSource>, Int32)</code>	Attempts to determine the number of elements in a sequence without forcing an enumeration.
<code>Enumerable.Chunk<TSource>(IEnumerable<TSource>, Int32)</code>	Splits the elements of a sequence into chunks of a specified size.
<code>Enumerable.MaxBy</code> and <code>Enumerable.MinBy</code>	Finds maximal or minimal elements using a key selector.
<code>Enumerable.DistinctBy</code> , <code>Enumerable.ExceptBy</code> , <code>Enumerable.IntersectBy</code> , and <code>Enumerable.UnionBy</code>	These new variations of methods that perform set-based operations let you specify equality using a key selector function.
<code>Enumerable.ElementAt<TSource>(IEnumerable<TSource>, Index)</code> and <code>Enumerable.ElementAtOrDefault<TSource>(IEnumerable<TSource>, Index)</code>	Accepts indexes counted from the beginning or end of the sequence—for example, <code>Enumerable.Range(1, 10).ElementAt(^2)</code> returns <code>9</code> .
<code>Enumerable.FirstOrDefault<TSource>(IEnumerable<TSource>, TSource)</code> and <code>Enumerable.FirstOrDefault<TSource>(IEnumerable<TSource>, Func<TSource, Boolean>, TSource)</code> <code>Enumerable.LastOrDefault<TSource>(IEnumerable<TSource>, TSource)</code> and <code>Enumerable.LastOrDefault<TSource>(IEnumerable<TSource>, Func<TSource, Boolean>, TSource)</code> <code>Enumerable.SingleOrDefault<TSource>(IEnumerable<TSource>, TSource)</code> and <code>Enumerable.SingleOrDefault<TSource>(IEnumerable<TSource>, Func<TSource, Boolean>, TSource)</code>	New overloads let you specify a default value to use if the sequence is empty.
<code>Enumerable.Max<TSource>(IEnumerable<TSource>, IComparer<TSource>)</code> and <code>Enumerable.Min<TSource>(IEnumerable<TSource>, IComparer<TSource>)</code>	New overloads let you specify a comparer.
<code>Enumerable.Take<TSource>(IEnumerable<TSource>, Range)</code>	Accepts a <code>Range</code> argument to simplify taking a slice of a sequence—for example, you can use <code>source.Take(2..7)</code> instead of <code>source.Take(7).Skip(2)</code> .

METHOD	DESCRIPTION
<code>Enumerable.Zip<TFirst,TSecond,TThird>(IEnumerable<TFirst>, IEnumerable<TSecond>, IEnumerable<TThird>)</code>	Produces a sequence of tuples with elements from <i>three</i> specified sequences.

Date, time, and time zone improvements

The following two structs were added in .NET 6: [System.DateOnly](#) and [System.TimeOnly](#). These represent the date part and the time part of a [DateTime](#), respectively. [DateOnly](#) is useful for birthdays and anniversaries, and [TimeOnly](#) is useful for daily alarms and weekly business hours.

You can now use either Internet Assigned Numbers Authority (IANA) or Windows time zone IDs on any operating system that has time zone data installed. The [TimeZoneInfo.FindSystemTimeZoneById\(String\)](#) method has been updated to automatically convert its input from a Windows time zone to an IANA time zone (or vice versa) if the requested time zone is not found on the system. In addition, the new methods [TryConvertIanaIdToWindowsId\(String, String\)](#) and [TryConvertWindowsIdToIanaId](#) have been added for scenarios when you still need to manually convert from one time zone format to another.

There are a few other time zone improvements as well. For more information, see [Date, Time, and Time Zone Enhancements in .NET 6](#).

PriorityQueue class

The new [PriorityQueue<TElement,TPriority>](#) class represents a collection of items that have both a value and a priority. Items are dequeued in increasing priority order—that is, the item with the lowest priority value is dequeued first. This class implements a [min heap](#) data structure.

See also

- [What's new in C# 10](#)
- [What's new in F# 6](#)
- [What's new in EF Core 6](#)
- [What's new in ASP.NET Core 6](#)
- [Release notes for .NET 6](#)
- [Release notes for Visual Studio 2022](#)
- [Blog: Announcing .NET 6](#)
- [Blog: Try the new System.Text.Json source generator](#)

Breaking changes in .NET 6

9/20/2022 • 5 minutes to read • [Edit Online](#)

If you're migrating an app to .NET 6, the breaking changes listed here might affect you. Changes are grouped by technology area, such as ASP.NET Core or Windows Forms.

This article categorizes each breaking change as *binary incompatible* or *source incompatible*.

- **Binary incompatible** - Existing binaries may encounter a breaking change in behavior, such as failure to load or execute, or different run-time behavior.
- **Source incompatible** - Source code may encounter a breaking change in behavior when targeting the new runtime or using the new SDK or component. Behavior changes can include compile errors or different run-time behavior.

NOTE

This article is a work-in-progress. It's not a complete list of breaking changes in .NET 6. To query breaking changes that are still pending publication, see [Issues of .NET](#).

ASP.NET Core

TITLE	BINARY COMPATIBLE	SOURCE COMPATIBLE	INTRODUCED
ActionResult<T> sets StatusCode to 200	✓	✗	
AddDataAnnotationsValidation method made obsolete	✓	✗	
Assemblies removed from Microsoft.AspNetCore.App shared framework	✗	✓	
Blazor: Parameter name changed in RequestImageFileAsync method	✓	✗	Preview 1
Blazor: WebEventDescriptorEventArgsType property replaced	✗	✗	
Blazor: Byte array interop	✓	✗	Preview 6
Changed MessagePack library in @microsoft/signalr-protocol-msgpack	✗	✓	

Title	Binary Compatible	Source Compatible	Introduced
ClientCertificate property doesn't trigger renegotiation for HttpSys	✓	✗	
EndpointName metadata not set automatically	✓	✗	RC 2
Identity: Default Bootstrap version of UI changed	✗	✗	
Kestrel: Log message attributes changed	✓	✗	
Microsoft.AspNetCore.Http. Features split	✗	✓	
Middleware: HTTPS Redirection Middleware throws exception on ambiguous HTTPS ports	✓	✗	
Middleware: New Use overload	✓	✗	Preview 4
Minimal API renames in RC 1	✗	✗	RC 1
Minimal API renames in RC 2	✗	✗	RC 2
MVC doesn't buffer IAsyncEnumerable types when using System.Text.Json	✓	✗	Preview 4
Nullable reference type annotations changed	✓	✗	
Obsolete and removed APIs	✓	✗	Preview 1
PreserveCompilationContext not configured by default	✗	✓	
Razor: Compiler no longer produces a Views assembly	✓	✗	Preview 3
Razor: Logging ID changes	✗	✓	RC1
Razor: RazorEngine APIs marked obsolete	✓	✗	Preview 1
SignalR: Java Client updated to RxJava3	✗	✓	Preview 4

Title	Binary Compatible	Source Compatible	Introduced
TryParse and BindAsync methods are validated	✗	✗	RC 2

Containers

Title	Binary Compatible	Source Compatible	Introduced
Default console logger formatting in container images	✓	✗	Servicing 6.0.6

For information on other breaking changes for containers in .NET 6, see [.NET 6 Container Release Notes](#).

Core .NET libraries

Title	Binary Compatible	Source Compatible	Introduced
API obsolesions with non-default diagnostic IDs	✓	✗	Preview 1
Changes to nullable reference type annotations	✓	✗	Preview 1-2
Conditional string evaluation in Debug methods	✓	✗	RC 1
Environment.ProcessorCount behavior on Windows	✓	✗	Preview 2
File.Replace on Unix throws exceptions to match Windows	✓	✗	Preview 7
FileStream locks files with shared lock on Unix	✗	✓	Preview 1
FileStream no longer synchronizes file offset with OS	✗	✗	Preview 4
FileStream.Position updates after ReadAsync or WriteAsync completes	✗	✗	Preview 4
New diagnostic IDs for obsoleted APIs	✓	✗	Preview 5
New nullable annotation in AssociatedMetadataTypeTypeDescriptionProvider	✓	✗	RC 2

Title	Binary Compatible	Source Compatible	Introduced
New System.Linq.Queryable method overloads	✓	✗	Preview 3-4
Older framework versions dropped from package	✗	✓	Preview 5
Parameter names changed	✓	✗	Preview 1
Parameter names in Stream-derived types	✓	✗	Preview 1
Partial and zero-byte reads in DeflateStream, GZipStream, and CryptoStream	✓	✗	Preview 6
Set timestamp on read-only file on Windows	✗	✓	Servicing 6.0.2
Standard numeric format parsing precision	✓	✗	Preview 2
Static abstract members in interfaces	✗	✓	Preview 7
StringBuilder.Append overloads and evaluation order	✗	✓	RC 1
Strong-name APIs throw PlatformNotSupportedException	✗	✓	Preview 4
System.Drawing.Common only supported on Windows	✗	✗	Preview 7
System.Security.SecurityContext is marked obsolete	✓	✗	RC 1
Task.FromResult may return singleton	✗	✓	Preview 1
Unhandled exceptions from a BackgroundService	✓	✗	Preview 4

Cryptography

Title	Binary Compatible	Source Compatible	Introduced
CreateEncryptor methods throw exception for incorrect feedback size	✗	✓	Preview 7

Deployment

TITLE	BINARY COMPATIBLE	SOURCE COMPATIBLE	INTRODUCED
x86 host path on 64-bit Windows	✓	✓	Servicing release

Entity Framework Core

[Breaking changes in EF Core 6](#)

Extensions

TITLE	BINARY COMPATIBLE	SOURCE COMPATIBLE	INTRODUCED
AddProvider checks for non-null provider	✓	✗	RC 1
FileConfigurationProvider.Load throws InvalidDataException	✓	✗	RC 1
Resolving disposed ServiceProvider throws exception	✓	✗	RC 1

Globalization

TITLE	BINARY COMPATIBLE	SOURCE COMPATIBLE	INTRODUCED
Culture creation and case mapping in globalization-invariant mode			Preview 7

Interop

TITLE	BINARY COMPATIBLE	SOURCE COMPATIBLE	INTRODUCED
Static abstract members in interfaces	✗	✓	Preview 7

JIT compiler

TITLE	BINARY COMPATIBLE	SOURCE COMPATIBLE	INTRODUCED
Coerce call arguments according to ECMA-335	✓	✓	Preview 1

Networking

Title	Binary Compatible	Source Compatible	Introduced
Port removed from SPN for Kerberos and Negotiate	✗	✓	RC 1
WebRequest, WebClient, and ServicePoint are obsolete	✓	✗	Preview 1

SDK

Title	Binary Compatible	Source Compatible	Introduced
-p option for dotnet run is deprecated	✓	✗	Preview 6
C# code in templates not supported by earlier versions	✓	✓	Preview 7
EditorConfig files implicitly included	✓	✗	
Generate apphost for macOS	✓	✗	Preview 6
Generate error for duplicate files in publish output	✗	✓	Preview 1
GetTargetFrameworkProperties and GetNearestTargetFramework removed from ProjectReference protocol	✗	✓	Preview 1
Install location for x64 emulated on Arm64	✓	✗	RC 2
MSBuild no longer supports calling GetType()			RC 1
OutputType not automatically set to WinExe	✓	✗	RC 1
Publish ReadyToRun with --no-restore requires changes	✓	✗	6.0.100
runtimeconfig.dev.json file not generated	✗	✓	6.0.100
Runtimeldentifier warning if self-contained is unspecified	✓	✗	RC 1
Version requirements for .NET 6 SDK	✓	✓	6.0.300

Title	Binary Compatible	Source Compatible	Introduced
.version file includes build version	✓	✓	6.0.401
Write reference assemblies to IntermediateOutputPath	✗	✓	6.0.200

Serialization

Title	Binary Compatible	Source Compatible	Introduced
Default serialization format for TimeSpan	✗	✓	Servicing 6.0.2
IAsyncEnumerable serialization	✓	✗	Preview 4
JSON source-generation API refactoring	✗	✓	RC 2
JsonNumberHandlingAttribute on collection properties	✗	✓	RC 1
New JsonSerializer source generator overloads	✗	✓	Preview 6

Windows Forms

Title	Binary Compatible	Source Compatible	Introduced
C# templates use application bootstrap	✓	✗	RC 1
Selected TableLayoutSettings properties throw InvalidEnumArgumentException	✗	✓	Preview 1
DataGridView-related APIs now throw InvalidOperationException	✗	✓	Preview 4
ListViewGroupCollection methods throw new InvalidOperationException	✗	✓	RC 2
NotifyIcon.Text maximum text length increased	✗	✓	Preview 1
ScaleControl called only when needed	✓	✗	Servicing 6.0.101

Title	Binary Compatible	Source Compatible	Introduced
Some APIs throw ArgumentNullException	✗	✓	Preview 1-4
TreeNodeCollection.Item throws exception if node is assigned elsewhere	✗	✓	Preview 1

XML and XSLT

Title	Binary Compatible	Source Compatible	Introduced
XmlDocument.XmlResolver nullability change	✗	✓	RC 1
XNodeReader.GetAttribute behavior for invalid index	✓	✗	Preview 2

What's new in .NET 5

9/20/2022 • 4 minutes to read • [Edit Online](#)

.NET 5 is the next major release of .NET Core following 3.1. We named this new release .NET 5 instead of .NET Core 4 for two reasons:

- We skipped version numbers 4.x to avoid confusion with .NET Framework 4.x.
 - We dropped "Core" from the name to emphasize that this is the main implementation of .NET going forward.
- .NET 5 supports more types of apps and more platforms than .NET Core or .NET Framework.

ASP.NET Core 5.0 is based on .NET 5 but retains the name "Core" to avoid confusing it with ASP.NET MVC 5. Likewise, Entity Framework Core 5.0 retains the name "Core" to avoid confusing it with Entity Framework 5 and 6.

.NET 5 includes the following improvements and new features compared to .NET Core 3.1:

- [C# updates](#)
- [F# updates](#)
- [Visual Basic updates](#)
- [System.Text.Json new features](#)
- [Single file apps](#)
- [App trimming](#)
- Windows Arm64 and Arm64 intrinsics
- Tooling support for dump debugging
- The runtime libraries are 80% annotated for [nullable reference types](#)
- Performance improvements:
 - [Garbage Collection \(GC\)](#)
 - [System.Text.Json](#)
 - [System.Text.RegularExpressions](#)
 - [Async ValueTask pooling](#)
 - [Container size optimizations](#)
 - [Many more areas](#)

.NET 5 doesn't replace .NET Framework

.NET 5 and later versions are the main implementation of .NET going forward, but .NET Framework 4.x is still supported. There are no plans to port the following technologies from .NET Framework to .NET 5, but there are alternatives in .NET:

TECHNOLOGY	RECOMMENDED ALTERNATIVE
Web Forms	ASP.NET Core Blazor or Razor Pages
Windows Workflow (WF)	Elsa-Workflows

Windows Communication Foundation

The original implementation of [Windows Communication Foundation \(WCF\)](#) was only supported on Windows. However, there's a client port available from the .NET Foundation. It's entirely [open source](#), cross platform, and

supported by Microsoft. The core NuGet packages are listed below:

- [System.ServiceModel.Duplex](#)
- [System.ServiceModel.Federation](#)
- [System.ServiceModel.Http](#)
- [System.ServiceModel.NetTcp](#)
- [System.ServiceModel.Primitives](#)
- [System.ServiceModel.Security](#)

The server components that complement the aforementioned client libraries are available through [CoreWCF](#). As of April 2022, CoreWCF is officially supported by Microsoft. However, for an alternative to WCF, consider [gRPC](#).

.NET 5 doesn't replace .NET Standard

New application development can specify the `net5.0` Target Framework Moniker (TFM) for all project types, including class libraries. Sharing code between .NET 5 workloads is simplified: all you need is the `net5.0` TFM.

For .NET 5 apps and libraries, the `net5.0` TFM combines and replaces the `netcoreapp` and `netstandard` TFMs. However, if you plan to share code between .NET Framework, .NET Core, and .NET 5 workloads, you can do so by specifying `netstandard2.0` as your TFM. For more information, see [.NET Standard](#).

C# updates

Developers writing .NET 5 apps will have access to the latest C# version and features. .NET 5 is paired with C# 9, which brings many new features to the language. Here are a few highlights:

- **Records:** Reference types with value-based equality semantics and non-destructive mutation supported by a new `with` expression.
- **Relational pattern matching:** Extends pattern matching capabilities to relational operators for comparative evaluations and expressions, including logical patterns - new keywords `and`, `or`, and `not`.
- **Top-level statements:** As a means for accelerating the adoption and learning of C#, the `Main` method can be omitted, and an application as simple as the following example is valid:

```
System.Console.WriteLine("Hello world!");
```

- **Function pointers:** Language constructs that expose the following intermediate language (IL) opcodes: `ldftn` and `calli`.

For more information on the available C# 9 features, see [What's new in C# 9](#).

Source generators

In addition to some of the highlighted new C# features, source generators are making their way into developer projects. Source generators allow code that runs during compilation to inspect your program and produce additional files that are compiled together with the rest of your code.

For more information on source generators, see [Introducing C# source generators](#) and [C# source generator samples](#).

F# updates

F# is the .NET functional programming language, and with .NET 5, developers have access to F# 5. One of the new features is interpolated strings, similar to interpolated strings in C#, and even JavaScript.

```
let name = "David"
let age = 36
let message = $"{name} is {age} years old."
```

In addition to basic string interpolation, there's typed interpolation. With typed interpolation, a given type must match the format specifier.

```
let name = "David"
let age = 36
let message = $"{name} is {age} years old."
```

This format is similar to the `sprintf` function that formats a string based on type-safe inputs.

For more information, see [What's new in F# 5](#).

Visual Basic updates

There are no new language features for Visual Basic in .NET 5. However, with .NET 5, Visual Basic support is extended to:

DESCRIPTION	DOTNET NEW PARAMETER
Console Application	console
Class library	classlib
WPF Application	wpf
WPF Class library	wpflib
WPF Custom Control Library	wpfcustomcontrollib
WPF User Control Library	wpfusercontrollib
Windows Forms (WinForms) Application	winforms
Windows Forms (WinForms) Class library	winformslib
Unit Test Project	mstest
NUnit 3 Test Project	nunit
NUnit 3 Test Item	nunit-test
xUnit Test Project	xunit

For more information on project templates from the .NET CLI, see [dotnet new](#).

System.Text.Json new features

There are new features in and for [System.Text.Json](#):

- [Preserve references and handle circular references](#)
- [HttpClient and HttpContent extension methods](#)
- [Allow or write numbers in quotes](#)
- [Support immutable types and C# 9 Records](#)
- [Support non-public property accessors](#)
- [Support fields](#)
- [Conditionally ignore properties](#)
- [Support non-string-key dictionaries](#)
- [Allow custom converters to handle null](#)
- [Copy JsonSerializerOptions](#)
- [Create JsonSerializerOptions with web defaults](#)

See also

- [The Journey to one .NET](#)
- [Performance improvements in .NET 5](#)
- [Download the .NET SDK](#)

Breaking changes in .NET 5

9/20/2022 • 5 minutes to read • [Edit Online](#)

If you're migrating an app to .NET 5, the breaking changes listed here might affect you. Changes are grouped by technology area, such as ASP.NET Core or cryptography.

This article categorizes each breaking change as *binary incompatible* or *source incompatible*.

- **Binary incompatible** - Existing binaries may encounter a breaking change in behavior, such as failure to load or execute, or different run-time behavior.
- **Source incompatible** - Source code may encounter a breaking change in behavior when targeting the new runtime or using the new SDK or component. Behavior changes can include compile errors or different run-time behavior.

ASP.NET Core

TITLE	BINARY COMPATIBLE	SOURCE COMPATIBLE
ASP.NET Core apps deserialize quoted numbers	✓	✗
AzureAD.UI and AzureADB2C.UI APIs obsolete	✓	✗
BinaryFormatter serialization methods are obsolete	✓	✗
Resource in endpoint routing is HttpContext	✓	✗
Microsoft-prefixed Azure integration packages removed	✗	✓
Blazor: Route precedence logic changed in Blazor apps	✓	✗
Blazor: Updated browser support	✓	✓
Blazor: Insignificant whitespace trimmed by compiler	✓	✗
Blazor: JObjectReference and JSInProcessObjectReference types are internal	✓	✗
Blazor: Target framework of NuGet packages changed	✗	✓
Blazor: ProtectedBrowserStorage feature moved to shared framework	✓	✗

Title	Binary Compatible	Source Compatible
Blazor: RenderTreeFrame readonly public fields are now properties	✗	✓
Blazor: Updated validation logic for static web assets	✗	✓
Cryptography APIs not supported on browser	✗	✓
Extensions: Package reference changes	✗	✓
Kestrel and IIS BadHttpRequestException types are obsolete	✓	✗
HttpClient instances created by IHHttpClientFactory log integer status codes	✓	✗
HttpSys: Client certificate renegotiation disabled by default	✓	✗
IIS: UrlRewrite middleware query strings are preserved	✓	✗
Kestrel: Configuration changes detected by default	✓	✗
Kestrel: Default supported TLS protocol versions changed	✓	✗
Kestrel: HTTP/2 disabled over TLS on incompatible Windows versions	✓	✓
Kestrel: Libuv transport marked as obsolete	✓	✗
Obsolete properties on ConsoleLoggerOptions	✓	✗
ResourceManagerWithCultureStringLocalizer class and WithCulture interface member removed	✓	✗
Pubternal APIs removed	✓	✗
Obsolete constructor removed in request localization middleware	✓	✗
Middleware: Database error page marked as obsolete	✓	✗
Exception handler middleware throws original exception	✓	✓

TITLE	BINARY COMPATIBLE	SOURCE COMPATIBLE
ObjectModelValidator calls a new overload of Validate	✓	✗
Cookie name encoding removed	✓	✗
IdentityModel NuGet package versions updated	✗	✓
SignalR: MessagePack Hub Protocol options type changed	✓	✗
SignalR: MessagePack Hub Protocol moved	✓	✗
UseSignalR and UseConnections methods removed	✓	✗
CSV content type changed to standards-compliant	✓	✗

Code analysis

TITLE	BINARY COMPATIBLE	SOURCE COMPATIBLE
CA1416 warning	✓	✗
CA1417 warning	✓	✗
CA1831 warning	✓	✗
CA2013 warning	✓	✗
CA2014 warning	✓	✗
CA2015 warning	✓	✗
CA2200 warning	✓	✗
CA2247 warning	✓	✗

Core .NET libraries

TITLE	BINARY COMPATIBLE	SOURCE COMPATIBLE
Assembly-related API changes for single-file publishing	✗	✓
BinaryFormatter serialization methods are obsolete	✓	✗

Title	Binary Compatible	Source Compatible
Code access security APIs are obsolete	✓	✗
CreateCounterSetInstance throws InvalidOperationException	✓	✗
Default ActivityIdFormat is W3C	✗	✓
Environment.OSVersion returns the correct version	✗	✓
FrameworkDescription's value is .NET not .NET Core	✓	✗
GAC APIs are obsolete	✓	✗
Hardware intrinsic IsSupported checks	✗	✓
IntPtr and UIntPtr implement IFormattable	✓	✗
LastIndexOf handles empty search strings	✗	✓
URI paths with non-ASCII characters on Unix	✗	✓
API obsolesions with non-default diagnostic IDs	✓	✗
Obsolete properties on ConsoleLoggerOptions	✓	✗
Complexity of LINQ OrderBy.First	✗	✓
OSPlatform attributes renamed or removed	✓	✗
Microsoft.DotNet.PlatformAbstractions package removed	✗	✓
PrincipalPermissionAttribute is obsolete	✓	✗
Parameter name changes from preview versions	✓	✗
Parameter name changes in reference assemblies	✓	✗
Remoting APIs are obsolete	✗	✓
Order of Activity.Tags list is reversed	✓	✗

TITLE	BINARY COMPATIBLE	SOURCE COMPATIBLE
SSE and SSE2 comparison methods handle NaN	✓	✗
Thread.Abort is obsolete	✓	✗
Uri recognition of UNC paths on Unix	✗	✓
UTF-7 code paths are obsolete	✓	✗
Behavior change for Vector2.Lerp and Vector4.Lerp	✓	✗
Vector<T> throws NotSupportedException	✗	✓

Cryptography

TITLE	BINARY COMPATIBLE	SOURCE COMPATIBLE
Cryptography APIs not supported on browser	✗	✓
Cryptography.Oid is init-only	✓	✗
Default TLS cipher suites on Linux	✗	✓
Create() overloads on cryptographic abstractions are obsolete	✓	✗
Default FeedbackSize value changed	✓	✗

Entity Framework Core

[Breaking changes in EF Core 5.0](#)

Globalization

TITLE	BINARY COMPATIBLE	SOURCE COMPATIBLE
Use ICU libraries on Windows	✗	✓
StringInfo and TextElementEnumerator are UAX29-compliant	✗	✓
Unicode category changed for Latin-1 characters	✓	✗
TextInfo.ListSeparator values changed	✓	✗

Interop

TITLE	BINARY COMPATIBLE	SOURCE COMPATIBLE
Support for WinRT is removed	✗	✓
Casting RCW to InterfacesInspectable throws exception	✗	✓
No A/W suffix probing on non-Windows platforms	✗	✓

Networking

TITLE	BINARY COMPATIBLE	SOURCE COMPATIBLE
Cookie path handling conforms to RFC 6265	✓	✗
LocalEndPoint is updated after calling SendToAsync	✓	✗
MulticastOption.Group doesn't accept null	✓	✗
Streams allow successive Begin operations	✗	✓
WinHttpHandler removed from .NET runtime	✗	✓

SDK

TITLE	BINARY COMPATIBLE	SOURCE COMPATIBLE
Directory.Packages.props files imported by default	✗	✓
Error generated when executable project references mismatched executable		✓
FrameworkReference replaced with WindowsSdkPackageVersion for Windows SDK	✓	✗
NETCOREAPP3_1 preprocessor symbol not defined	✓	✗
OutputType set to WinExe	✗	✓
PublishDepsFilePath behavior change	✗	✓
TargetFramework change from netcoreapp to net	✗	✓

TITLE	BINARY COMPATIBLE	SOURCE COMPATIBLE
WinForms and WPF apps use Microsoft.NETSdk	✗	✓

Security

TITLE	BINARY COMPATIBLE	SOURCE COMPATIBLE
Code access security APIs are obsolete	✓	✗
PrincipalPermissionAttribute is obsolete	✓	✗
UTF-7 code paths are obsolete	✓	✗

Serialization

TITLE	BINARY COMPATIBLE	SOURCE COMPATIBLE
BinaryFormatter.Deserialize rewraps exceptions	✓	✗
JsonSerializer.Deserialize requires single-character string	✓	✗
ASP.NET Core apps deserialize quoted numbers	✓	✗
JsonSerializer.Serialize throws ArgumentNullException	✓	✗
Non-public, parameterless constructors not used for deserialization	✓	✗
Options are honored when serializing key-value pairs	✓	✗

Windows Forms

TITLE	BINARY COMPATIBLE	SOURCE COMPATIBLE
Native code can't access Windows Forms objects	✓	✗
OutputType set to WinExe	✗	✓
DataGridView doesn't reset custom fonts	✓	✗
Methods throw ArgumentException	✓	✗

TITLE	BINARY COMPATIBLE	SOURCE COMPATIBLE
Methods throw ArgumentNullException	✓	✗
Properties throw ArgumentOutOfRangeException	✓	✗
TextFormatFlags.ModifyString is obsolete	✓	✗
DataGridView APIs throw InvalidOperationException	✓	✗
WinForms apps use Microsoft.NET.Sdk	✗	✓
Removed status bar controls	✓	✗

WPF

TITLE	BINARY COMPATIBLE	SOURCE COMPATIBLE
OutputType set to WinExe	✗	✓
WPF apps use Microsoft.NET.Sdk	✗	✓

What's new in .NET Core 3.1

9/20/2022 • 3 minutes to read • [Edit Online](#)

This article describes what is new in .NET Core 3.1. This release contains minor improvements to .NET Core 3.0, focusing on small, but important, fixes. The most important feature about .NET Core 3.1 is that it's a [long-term support \(LTS\)](#) release.

If you're using Visual Studio 2019, you must update to [Visual Studio 2019 version 16.4 or later](#) to work with .NET Core 3.1 projects. For information on what's new in Visual Studio version 16.4, see [What's New in Visual Studio 2019 version 16.4](#).

Visual Studio for Mac also supports and includes .NET Core 3.1 in Visual Studio for Mac 8.4.

For more information about the release, see the [.NET Core 3.1 announcement](#).

- [Download and get started with .NET Core 3.1](#) on Windows, macOS, or Linux.

Long-term support

.NET Core 3.1 is an LTS release with support from Microsoft for three years after its release. It's highly recommended that you move your apps to the latest LTS release. See the [.NET and .NET Core support policy](#) page for a list of supported releases.

RELEASE	END OF LIFE DATE
.NET Core 3.1	End of life on December 13, 2022.
.NET Core 3.0	End of life on March 3, 2020.
.NET Core 2.2	End of life on December 23, 2019.
.NET Core 2.1	End of life on August 21, 2021.

For more information, see the [.NET and .NET Core support policy](#).

macOS appHost and notarization

macOS only

Starting with the notarized .NET Core SDK 3.1 for macOS, the appHost setting is disabled by default. For more information, see [macOS Catalina Notarization and the impact on .NET Core downloads and projects](#).

When the appHost setting is enabled, .NET Core generates a native Mach-O executable when you build or publish. Your app runs in the context of the appHost when it is run from source code with the `dotnet run` command, or by starting the Mach-O executable directly.

Without the appHost, the only way a user can start a [framework-dependent](#) app is with the `dotnet <filename.dll>` command. An appHost is always created when you publish your app [self-contained](#).

You can either configure the appHost at the project level, or toggle the appHost for a specific `dotnet` command with the `-p:UseAppHost` parameter:

- Project file

```
<PropertyGroup>
  <UseAppHost>true</UseAppHost>
</PropertyGroup>
```

- Command-line parameter

```
dotnet run -p:UseAppHost=true
```

For more information about the `UseAppHost` setting, see [MSBuild properties for Microsoft.NET.Sdk](#).

Windows Forms

Windows only

WARNING

There are breaking changes in Windows Forms.

Legacy controls were included in Windows Forms that have been unavailable in the Visual Studio Designer Toolbox for some time. These were replaced with new controls back in .NET Framework 2.0. These have been removed from the Desktop SDK for .NET Core 3.1.

REMOVED CONTROL	RECOMMENDED REPLACEMENT	ASSOCIATED APIs REMOVED
DataGrid	DataGridView	DataGridCell DataGridRow DataGridTableCollection DataGridColumnCollection DataGridTableStyle DataGridColumnStyle DataGridLineStyle DataGridParentRowsLabel DataGridParentRowsLabelStyle DataGridBoolColumn DataGridTextBox GridColumnStylesCollection GridTableStylesCollection HitTestType
ToolBar	ToolStrip	ToolBarAppearance
ToolBarButton	ToolStripButton	ToolBarButtonClickEventArgs ToolBarButtonClickEventHandler ToolBarButtonStyle ToolBarTextAlign
ContextMenu	ContextMenuStrip	
Menu	ToolStripDropDown ToolStripDropDownMenu	MenuItemCollection
MainMenu	MenuStrip	
MenuItem	ToolStripMenuItem	

We recommend you update your applications to .NET Core 3.1 and move to the replacement controls. Replacing the controls is a straightforward process, essentially "find and replace" on the type.

C++/CLI

Windows only

Support has been added for creating C++/CLI (also known as "managed C++") projects. Binaries produced from these projects are compatible with .NET Core 3.0 and later versions.

To add support for C++/CLI in Visual Studio 2019 version 16.4, install the [Desktop development with C++ workload](#). This workload adds two templates to Visual Studio:

- CLR Class Library (.NET Core)
- CLR Empty Project (.NET Core)

Next steps

- [Review the breaking changes between .NET Core 3.0 and 3.1.](#)
- [Review the breaking changes in .NET Core 3.1 for Windows Forms apps.](#)

Breaking changes in .NET Core 3.1

9/20/2022 • 8 minutes to read • [Edit Online](#)

If you're migrating to version 3.1 of .NET Core or ASP.NET Core, the breaking changes listed in this article may affect your app.

ASP.NET Core

- [HTTP: Browser SameSite changes impact authentication](#)

HTTP: Browser SameSite changes impact authentication

Some browsers, such as Chrome and Firefox, made breaking changes to their implementations of `SameSite` for cookies. The changes impact remote authentication scenarios, such as OpenID Connect and WS-Federation, which must opt out by sending `SameSite=None`. However, `SameSite=None` breaks on iOS 12 and some older versions of other browsers. The app needs to sniff these versions and omit `sameSite`.

For discussion on this issue, see [dotnet/aspnetcore#14996](#).

Version introduced

3.1 Preview 1

Old behavior

`SameSite` is a 2016 draft standard extension to HTTP cookies. It's intended to mitigate Cross-Site Request Forgery (CSRF). This was originally designed as a feature the servers would opt into by adding the new parameters. ASP.NET Core 2.0 added initial support for `SameSite`.

New behavior

Google proposed a new draft standard that isn't backwards compatible. The standard changes the default mode to `Lax` and adds a new entry `None` to opt out. `Lax` suffices for most app cookies; however, it breaks cross-site scenarios like OpenID Connect and WS-Federation login. Most OAuth logins aren't affected because of differences in how the request flows. The new `None` parameter causes compatibility problems with clients that implemented the prior draft standard (for example, iOS 12). Chrome 80 will include the changes. See [SameSite Updates](#) for the Chrome product launch timeline.

ASP.NET Core 3.1 has been updated to implement the new `SameSite` behavior. The update redefines the behavior of `SameSiteMode.None` to emit `SameSite=None` and adds a new value `SameSiteMode.Unspecified` to omit the `SameSite` attribute. All cookie APIs now default to `Unspecified`, though some components that use cookies set values more specific to their scenarios such as the OpenID Connect correlation and nonce cookies.

For other recent changes in this area, see [HTTP: Some cookie SameSite defaults changed to None](#). In ASP.NET Core 3.0, most defaults were changed from `SameSiteMode.Lax` to `SameSiteMode.None` (but still using the prior standard).

Reason for change

Browser and specification changes as outlined in the preceding text.

Recommended action

Apps that interact with remote sites, such as through third-party login, need to:

- Test those scenarios on multiple browsers.
- Apply the cookie policy browser sniffing mitigation discussed in [Support older browsers](#).

For testing and browser sniffing instructions, see the following section.

Determine if you're affected

Test your web app using a client version that can opt into the new behavior. Chrome, Firefox, and Microsoft Edge Chromium all have new opt-in feature flags that can be used for testing. Verify that your app is compatible with older client versions after you've applied the patches, especially Safari. For more information, see [Support older browsers](#).

Chrome

Chrome 78 and later yield misleading test results. Those versions have a temporary mitigation in place and allow cookies less than two minutes old. With the appropriate test flags enabled, Chrome 76 and 77 yield more accurate results. To test the new behavior, toggle `chrome://flags/#same-site-by-default-cookies` to enabled. Chrome 75 and earlier are reported to fail with the new `None` setting. For more information, see [Support older browsers](#).

Google doesn't make older Chrome versions available. You can, however, download older versions of Chromium, which will suffice for testing. Follow the instructions at [Download Chromium](#).

- [Chromium 76 Win64](#)
- [Chromium 74 Win64](#)

Safari

Safari 12 strictly implemented the prior draft and fails if it sees the new `None` value in cookies. This must be avoided via the browser sniffing code shown in [Support older browsers](#). Ensure you test Safari 12 and 13 as well as WebKit-based, OS-style logins using Microsoft Authentication Library (MSAL), Active Directory Authentication Library (ADAL), or whichever library you're using. The problem is dependent on the underlying OS version. OSX Mojave 10.14 and iOS 12 are known to have compatibility problems with the new behavior. Upgrading to OSX Catalina 10.15 or iOS 13 fixes the problem. Safari doesn't currently have an opt-in flag for testing the new specification behavior.

Firefox

Firefox support for the new standard can be tested on version 68 and later by opting in on the `about:config` page with the feature flag `network.cookie.sameSite.laxByDefault`. No compatibility issues have been reported on older versions of Firefox.

Microsoft Edge

While Microsoft Edge supports the old `SameSite` standard, as of version 44 it didn't have any compatibility problems with the new standard.

Microsoft Edge Chromium

The feature flag is `edge://flags/#same-site-by-default-cookies`. No compatibility issues were observed when testing with Microsoft Edge Chromium 78.

Electron

Versions of Electron include older versions of Chromium. For example, the version of Electron used by Microsoft Teams is Chromium 66, which exhibits the older behavior. Perform your own compatibility testing with the version of Electron your product uses. For more information, see [Support older browsers](#).

Support older browsers

The 2016 `SameSite` standard mandated that unknown values be treated as `SameSite=Strict` values. Consequently, any older browsers that support the original standard may break when they see a `SameSite` property with a value of `None`. Web apps must implement browser sniffing if they intend to support these old browsers. ASP.NET Core doesn't implement browser sniffing for you because `User-Agent` request header values are highly unstable and change on a weekly basis. Instead, an extension point in the cookie policy allows you to add `User-Agent`-specific logic.

In `Startup.cs`, add the following code:

```

private void CheckSameSite(HttpContext httpContext, CookieOptions options)
{
    if (options.SameSite == SameSiteMode.None)
    {
        var userAgent = httpContext.Request.Headers["User-Agent"].ToString();
        // TODO: Use your User Agent library of choice here.
        if /* UserAgent doesn't support new behavior */
        {
            options.SameSite = SameSiteMode.Unspecified;
        }
    }
}

public void ConfigureServices(IServiceCollection services)
{
    services.Configure<CookiePolicyOptions>(options =>
    {
        options.MinimumSameSitePolicy = SameSiteMode.Unspecified;
        options.OnAppendCookie = cookieContext =>
            CheckSameSite(cookieContext.Context, cookieContext.CookieOptions);
        options.OnDeleteCookie = cookieContext =>
            CheckSameSite(cookieContext.Context, cookieContext.CookieOptions);
    });
}

public void Configure(IApplicationBuilder app)
{
    // Before UseAuthentication or anything else that writes cookies.
    app.UseCookiePolicy();

    app.UseAuthentication();
    // code omitted for brevity
}

```

Opt-out switches

The `Microsoft.AspNetCore.SuppressSameSiteNone` compatibility switch enables you to temporarily opt out of the new ASP.NET Core cookie behavior. Add the following JSON to a `runtimeconfig.template.json` file in your project:

```
{
  "configProperties": {
    "Microsoft.AspNetCore.SuppressSameSiteNone": "true"
  }
}
```

Other Versions

Related `SameSite` patches are forthcoming for:

- ASP.NET Core 2.1, 2.2, and 3.0
- `Microsoft.Owin` 4.1
- `System.Web` (for .NET Framework 4.7.2 and later)

Category

ASP.NET

Affected APIs

- `Microsoft.AspNetCore.Builder.CookiePolicyOptions.MinimumSameSitePolicy`
- `Microsoft.AspNetCore.Http.CookieBuilder.SameSite`
- `Microsoft.AspNetCore.Http.CookieOptions.SameSite`
- `Microsoft.AspNetCore.Http.SameSiteMode`
- `Microsoft.Net.Http.Headers.SameSiteMode`

- Microsoft.Net.Http.Headers.SetCookieHeaderValue.SameSite
-

Deployment

x86 host path on 64-bit Windows

MSBuild

- Design-time builds only return top-level package references

Design-time builds only return top-level package references

Starting in .NET Core SDK 3.1.400, only top-level package references are returned by the

`RunResolvePackageDependencies` target.

Version introduced

.NET Core SDK 3.1.400

Change description

In previous versions of the .NET Core SDK, the `RunResolvePackageDependencies` target created the following MSBuild items that contained information from the NuGet assets file:

- `PackageDefinitions`
- `PackageDependencies`
- `TargetDefinitions`
- `FileDefinitions`
- `FileDependencies`

This data is used by Visual Studio to populate the Dependencies node in Solution Explorer. However, it can be a large amount of data, and the data isn't needed unless the Dependencies node is expanded.

Starting in the .NET Core SDK version 3.1.400, most of these items aren't generated by default. Only items of type `Package` are returned. If Visual Studio needs the items to populate the Dependencies node, it reads the information directly from the assets file.

Reason for change

This change was introduced to improve solution-load performance inside of Visual Studio. Previously, all package references would be loaded, which involved loading many references that most users would never view.

Recommended action

If you have MSBuild logic that depends on these items being created, set the `EmitLegacyAssetsFileItems` property to `true` in your project file. This setting enables the previous behavior where all the items are created.

Category

MSBuild

Affected APIs

N/A

Windows Forms

- Removed controls
- CellFormatting event not raised if tooltip is shown

Removed controls

Starting in .NET Core 3.1, some Windows Forms controls are no longer available.

Change description

Starting with .NET Core 3.1, various Windows Forms controls are no longer available. Replacement controls that have better design and support were introduced in .NET Framework 2.0. The deprecated controls were previously removed from designer toolboxes but were still available to be used.

The following types are no longer available:

- [ContextMenu](#)
- [DataGridView](#)
- [DataGridView.HitTestType](#)
- [DataGridViewBoolColumn](#)
- [DataGridViewCell](#)
- [DataGridViewCellStyle](#)
- [DataGridViewLineStyle](#)
- [DataGridViewParentRowsLabelStyle](#)
- [DataGridViewPreferredColumnWidthTypeConverter](#)
- [DataGridViewTableStyle](#)
- [DataGridViewTextBox](#)
- [DataGridViewTextBoxColumn](#)
- [GridColumnStylesCollection](#)
- [GridTablesFactory](#)
- [GridTableStylesCollection](#)
- [IDataGridEditingService](#)
- [IMenuEditorService](#)
- [MainMenu](#)
- [Menu](#)
- [MenuItem.MenuItemCollection](#)
- [MenuItem](#)
- [ToolBar](#)
- [ToolBarAppearance](#)
- [ToolBarButton](#)
- [ToolBar.ToolBarButtonCollection](#)
- [ToolBarButtonClickEventArgs](#)
- [ToolBarButtonStyle](#)
- [ToolBar.TextAlign](#)

Version introduced

3.1

Recommended action

Each removed control has a recommended replacement control. Refer to the following table:

REMOVED CONTROL (API)	RECOMMENDED REPLACEMENT	ASSOCIATED APIs THAT ARE REMOVED
ContextMenu	ContextMenuStrip	

REMOVED CONTROL (API)	RECOMMENDED REPLACEMENT	ASSOCIATED APIs THAT ARE REMOVED
DataGridView	DataGridView	DataGridCell, DataGridRow, DataGridTableCollection, DataGridColumnCollection, DataGridTableStyle, DataGridColumnStyle, DataGridLineStyle, DataGridParentRowsLabel, DataGridParentRowsLabelStyle, DataGridBoolColumn, DataGridViewTextBox, GridColumnStylesCollection, GridTableStylesCollection, HitTestType
MainMenu	MenuStrip	
Menu	ToolStripDropDown, ToolStripDropDownMenu	MenuItemCollection
MenuItem	ToolStripMenuItem	
ToolBar	ToolStrip	ToolBarAppearance
ToolBarButton	ToolStripButton	ToolBarButtonClickEventArgs,ToolBarButtonClickEventHandler,ToolBarButtonStyle,ToolBarTextAlign

Category

Windows Forms

Affected APIs

- [System.Windows.Forms.ContextMenu](#)
- [System.Windows.Forms.GridColumnStylesCollection](#)
- [System.Windows.Forms.GridTablesFactory](#)
- [System.Windows.Forms.GridTableStylesCollection](#)
- [System.Windows.Forms.IDataGridEditingService](#)
- [System.Windows.Forms.MainMenu](#)
- [System.Windows.Forms.Menu](#)
- [System.Windows.Forms.Menu.MenuItemCollection](#)
- [System.Windows.Forms.MenuItem](#)
- [System.Windows.Forms.ToolBar](#)
- [System.Windows.Forms.ToolBar.ToolBarButtonCollection](#)
- [System.Windows.Forms.ToolBarAppearance](#)
- [System.Windows.Forms.ToolBarButton](#)
- [System.Windows.Forms.ToolBarButtonClickEventArgs](#)
- [System.Windows.Forms.ToolBarButtonStyle](#)
- [System.Windows.Forms.ToolBar.TextAlign](#)
- [System.Windows.Forms.DataGrid](#)
- [System.Windows.Forms.DataGrid.HitTestType](#)
- [System.Windows.Forms.DataGridBoolColumn](#)
- [System.Windows.Forms.DataGridCell](#)

- [System.Windows.Forms.DataGridViewColumnStyle](#)
 - [System.Windows.Forms.DataGridViewLineStyle](#)
 - [System.Windows.Forms.DataGridViewParentRowsLabelStyle](#)
 - [System.Windows.Forms.DataGridViewColumnPreferredColumnWidthTypeConverter](#)
 - [System.Windows.Forms.DataGridViewTableStyle](#)
 - [System.Windows.Forms.DataGridViewTextBox](#)
 - [System.Windows.Forms.DataGridViewColumnTextBoxColumn](#)
 - [System.Windows.Forms.Design.IMenuEditorService](#)
-

CellFormatting event not raised if tooltip is shown

A [DataGridView](#) now shows a cell's text and error tooltips when hovered by a mouse and when selected via the keyboard. If a tooltip is shown, the [DataGridView.CellFormatting](#) event is not raised.

Change description

Prior to .NET Core 3.1, a [DataGridView](#) that had the [ShowCellToolTips](#) property set to `true` showed a tooltip for a cell's text and errors when the cell was hovered by a mouse. Tooltips were not shown when a cell was selected via the keyboard (for example, by using the Tab key, shortcut keys, or arrow navigation). If the user edited a cell, and then, while the [DataGridView](#) was still in edit mode, hovered over a cell that did not have the [ToolTipText](#) property set, a [CellFormatting](#) event was raised to format the cell's text for display in the cell.

To meet accessibility standards, starting in .NET Core 3.1, a [DataGridView](#) that has the [ShowCellToolTips](#) property set to `true` shows tooltips for a cell's text and errors not only when the cell is hovered, but also when it's selected via the keyboard. As a consequence of this change, the [CellFormatting](#) event is *not* raised when cells that don't have the [ToolTipText](#) property set are hovered while the [DataGridView](#) is in edit mode. The event is not raised because the content of the hovered cell is shown as a tooltip instead of being displayed in the cell.

Version introduced

3.1

Recommended action

Refactor any code that depends on the [CellFormatting](#) event while the [DataGridView](#) is in edit mode.

Category

Windows Forms

Affected APIs

None

What's new in .NET Core 3.0

9/20/2022 • 22 minutes to read • [Edit Online](#)

This article describes what is new in .NET Core 3.0. One of the biggest enhancements is support for Windows desktop applications (Windows only). By using the .NET Core 3.0 SDK component Windows Desktop, you can port your Windows Forms and Windows Presentation Foundation (WPF) applications. To be clear, the Windows Desktop component is only supported and included on Windows. For more information, see the [Windows desktop](#) section later in this article.

.NET Core 3.0 adds support for C# 8.0. It's highly recommended that you use [Visual Studio 2019 version 16.3](#) or newer, [Visual Studio for Mac 8.3](#) or newer, or [Visual Studio Code](#) with the latest C# extension.

[Download and get started with .NET Core 3.0](#) right now on Windows, macOS, or Linux.

For more information about the release, see the [.NET Core 3.0 announcement](#).

.NET Core 3.0 RC 1 was considered production ready by Microsoft and was fully supported. If you're using a preview release, you must move to the RTM version for continued support.

Language improvements C# 8.0

C# 8.0 is also part of this release, which includes the [nullable reference types](#) feature, [async streams](#), and more patterns. For more information about C# 8.0 features, see [What's new in C# 8.0](#).

Tutorials related to C# 8.0 language features:

- [Tutorial: Express your design intent more clearly with nullable and non-nullable reference types](#)
- [Tutorial: Generate and consume async streams using C# 8.0 and .NET Core 3.0](#)
- [Tutorial: Use pattern matching to build type-driven and data-driven algorithms](#)

Language enhancements were added to support the following API features detailed below:

- [Ranges and indices](#)
- [Async streams](#)

.NET Standard 2.1

.NET Core 3.0 implements [.NET Standard 2.1](#). However, the default `dotnet new classlib` template generates a project that still targets .NET Standard 2.0. To target .NET Standard 2.1, edit your project file and change the `TargetFramework` property to `netstandard2.1`:

```
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <TargetFramework>netstandard2.1</TargetFramework>
  </PropertyGroup>

</Project>
```

If you're using Visual Studio, you need [Visual Studio 2019](#), as Visual Studio 2017 doesn't support .NET Standard 2.1 or .NET Core 3.0.

Compile/Deploy

Default executables

.NET Core now builds [framework-dependent executables](#) by default. This behavior is new for applications that use a globally installed version of .NET Core. Previously, only [self-contained deployments](#) would produce an executable.

During `dotnet build` or `dotnet publish`, an executable (known as the **appHost**) is created that matches the environment and platform of the SDK you're using. You can expect the same things with these executables as you would other native executables, such as:

- You can double-click on the executable.
- You can launch the application from a command prompt directly, such as `myapp.exe` on Windows, and `./myapp` on Linux and macOS.

macOS appHost and notarization

macOS only

Starting with the notarized .NET Core SDK 3.0 for macOS, the setting to produce a default executable (known as the **appHost**) is disabled by default. For more information, see [macOS Catalina Notarization and the impact on .NET Core downloads and projects](#).

When the **appHost** setting is enabled, .NET Core generates a native Mach-O executable when you build or publish. Your app runs in the context of the **appHost** when it is run from source code with the `dotnet run` command, or by starting the Mach-O executable directly.

Without the **appHost**, the only way a user can start a [framework-dependent](#) app is with the `dotnet <filename.dll>` command. An **appHost** is always created when you publish your app [self-contained](#).

You can either configure the **appHost** at the project level, or toggle the **appHost** for a specific `dotnet` command with the `-p:UseAppHost` parameter:

- Project file

```
<PropertyGroup>
  <UseAppHost>true</UseAppHost>
</PropertyGroup>
```

- Command-line parameter

```
dotnet run -p:UseAppHost=true
```

For more information about the `UseAppHost` setting, see [MSBuild properties for Microsoft.NET.Sdk](#).

Single-file executables

The `dotnet publish` command supports packaging your app into a platform-specific single-file executable. The executable is self-extracting and contains all dependencies (including native) that are required to run your app. When the app is first run, the application is extracted to a directory based on the app name and build identifier. Startup is faster when the application is run again. The application doesn't need to extract itself a second time unless a new version was used.

To publish a single-file executable, set the `PublishSingleFile` in your project or on the command line with the `dotnet publish` command:

```
<PropertyGroup>
  <RuntimeIdentifier>win10-x64</RuntimeIdentifier>
  <PublishSingleFile>true</PublishSingleFile>
</PropertyGroup>
```

-or-

```
dotnet publish -r win10-x64 -p:PublishSingleFile=true
```

For more information about single-file publishing, see the [single-file bundler design document](#).

Assembly trimming

The .NET core 3.0 SDK comes with a tool that can reduce the size of apps by analyzing IL and trimming unused assemblies.

Self-contained apps include everything needed to run your code, without requiring .NET to be installed on the host computer. However, many times the app only requires a small subset of the framework to function, and other unused libraries could be removed.

.NET Core now includes a setting that will use the [IL Trimmer](#) tool to scan the IL of your app. This tool detects what code is required, and then trims unused libraries. This tool can significantly reduce the deployment size of some apps.

To enable this tool, add the `<PublishTrimmed>` setting in your project and publish a self-contained app:

```
<PropertyGroup>
  <PublishTrimmed>true</PublishTrimmed>
</PropertyGroup>
```

```
dotnet publish -r <rid> -c Release
```

As an example, the basic "hello world" new console project template that is included, when published, hits about 70 MB in size. By using `<PublishTrimmed>`, that size is reduced to about 30 MB.

It's important to consider that applications or frameworks (including ASP.NET Core and WPF) that use reflection or related dynamic features, will often break when trimmed. This breakage occurs because the trimmer doesn't know about this dynamic behavior and can't determine which framework types are required for reflection. The IL Trimmer tool can be configured to be aware of this scenario.

Above all else, be sure to test your app after trimming.

For more information about the IL Trimmer tool, see the [documentation](#) or visit the [mono/linker](#) repo.

Tiered compilation

Tiered compilation (TC) is on by default with .NET Core 3.0. This feature enables the runtime to more adaptively use the just-in-time (JIT) compiler to achieve better performance.

The main benefit of tiered compilation is to provide two ways of jittering methods: in a lower-quality-but-faster tier or a higher-quality-but-slower tier. The quality refers to how well the method is optimized. TC helps to improve the performance of an application as it goes through various stages of execution, from startup through steady state. When tiered compilation is disabled, every method is compiled in a single way that's biased to steady-state performance over startup performance.

When TC is enabled, the following behavior applies for method compilation when an app starts up:

- If the method has ahead-of-time-compiled code, or [ReadyToRun](#), the pregenerated code is used.
- Otherwise, the method is jitted. Typically, these methods are generics over value types.
 - *Quick JIT* produces lower-quality (or less optimized) code more quickly. In .NET Core 3.0, Quick JIT is enabled by default for methods that don't contain loops and is preferred during startup.
 - The fully optimizing JIT produces higher-quality (or more optimized) code more slowly. For methods where Quick JIT would not be used (for example, if the method is attributed with [MethodImplOptions.AggressiveOptimization](#)), the fully optimizing JIT is used.

For frequently called methods, the just-in-time compiler eventually creates fully optimized code in the background. The optimized code then replaces the pre-compiled code for that method.

Code generated by Quick JIT may run slower, allocate more memory, or use more stack space. If there are issues, you can disable Quick JIT using this MSBuild property in the project file:

```
<PropertyGroup>
  <TieredCompilationQuickJit>false</TieredCompilationQuickJit>
</PropertyGroup>
```

To disable TC completely, use this MSBuild property in your project file:

```
<PropertyGroup>
  <TieredCompilation>false</TieredCompilation>
</PropertyGroup>
```

TIP

If you change these settings in the project file, you may need to perform a clean build for the new settings to be reflected (delete the `obj` and `bin` directories and rebuild).

For more information about configuring compilation at run time, see [Runtime configuration options for compilation](#).

ReadyToRun images

You can improve the startup time of your .NET Core application by compiling your application assemblies as ReadyToRun (R2R) format. R2R is a form of ahead-of-time (AOT) compilation.

R2R binaries improve startup performance by reducing the amount of work the just-in-time (JIT) compiler needs to do as your application loads. The binaries contain similar native code compared to what the JIT would produce. However, R2R binaries are larger because they contain both intermediate language (IL) code, which is still needed for some scenarios, and the native version of the same code. R2R is only available when you publish a self-contained app that targets specific runtime environments (RID) such as Linux x64 or Windows x64.

To compile your project as ReadyToRun, do the following:

1. Add the `<PublishReadyToRun>` setting to your project:

```
<PropertyGroup>
  <PublishReadyToRun>true</PublishReadyToRun>
</PropertyGroup>
```

2. Publish a self-contained app. For example, this command creates a self-contained app for the 64-bit version of Windows:

```
dotnet publish -c Release -r win-x64 --self-contained
```

Cross platform/architecture restrictions

The ReadyToRun compiler doesn't currently support cross-targeting. You must compile on a given target. For example, if you want R2R images for Windows x64, you need to run the publish command on that environment.

Exceptions to cross-targeting:

- Windows x64 can be used to compile Windows Arm32, Arm64, and x86 images.
- Windows x86 can be used to compile Windows Arm32 images.
- Linux x64 can be used to compile Linux Arm32 and Arm64 images.

For more information, see [Ready to Run](#).

Runtime/SDK

Major-version runtime roll forward

.NET Core 3.0 introduces an opt-in feature that allows your app to roll forward to the latest major version of .NET Core. Additionally, a new setting has been added to control how roll forward is applied to your app. This can be configured in the following ways:

- Project file property: `RollForward`
- Runtime configuration file property: `rollForward`
- Environment variable: `DOTNET_ROLL_FORWARD`
- Command-line argument: `--roll-forward`

One of the following values must be specified. If the setting is omitted, **Minor** is the default.

- **LatestPatch**

Roll forward to the highest patch version. This disables minor version roll forward.

- **Minor**

Roll forward to the lowest higher minor version, if requested minor version is missing. If the requested minor version is present, then the **LatestPatch** policy is used.

- **Major**

Roll forward to lowest higher major version, and lowest minor version, if requested major version is missing. If the requested major version is present, then the **Minor** policy is used.

- **LatestMinor**

Roll forward to highest minor version, even if requested minor version is present. Intended for component hosting scenarios.

- **LatestMajor**

Roll forward to highest major and highest minor version, even if requested major is present. Intended for component hosting scenarios.

- **Disable**

Don't roll forward. Only bind to specified version. This policy isn't recommended for general use because it disables the ability to roll forward to the latest patches. This value is only recommended for testing.

Besides the **Disable** setting, all settings will use the highest available patch version.

By default, if the requested version (as specified in `.runtimeconfig.json` for the application) is a release version, only release versions are considered for roll forward. Any pre-release versions are ignored. If there is no matching release version, then pre-release versions are taken into account. This behavior can be changed by setting `DOTNET_ROLL_FORWARD_TO_PRERELEASE=1`, in which case all versions are always considered.

Build copies dependencies

The `dotnet build` command now copies NuGet dependencies for your application from the NuGet cache to the build output folder. Previously, dependencies were only copied as part of `dotnet publish`.

There are some operations, like trimming and razor page publishing, that will still require publishing.

Local tools

.NET Core 3.0 introduces local tools. Local tools are similar to [global tools](#) but are associated with a particular location on disk. Local tools aren't available globally and are distributed as NuGet packages.

Local tools rely on a manifest file name `dotnet-tools.json` in your current directory. This manifest file defines the tools to be available at that folder and below. You can distribute the manifest file with your code to ensure that anyone who works with your code can restore and use the same tools.

For both global and local tools, a compatible version of the runtime is required. Many tools currently on NuGet.org target .NET Core Runtime 2.1. To install these tools globally or locally, you would still need to install the [NET Core 2.1 Runtime](#).

New `global.json` options

The `global.json` file has new options that provide more flexibility when you're trying to define which version of the .NET Core SDK is used. The new options are:

- `allowPrerelease` : Indicates whether the SDK resolver should consider prerelease versions when selecting the SDK version to use.
- `rollForward` : Indicates the roll-forward policy to use when selecting an SDK version, either as a fallback when a specific SDK version is missing or as a directive to use a higher version.

For more information about the changes including default values, supported values, and new matching rules, see [global.json overview](#).

Smaller Garbage Collection heap sizes

The Garbage Collector's default heap size has been reduced resulting in .NET Core using less memory. This change better aligns with the generation 0 allocation budget with modern processor cache sizes.

Garbage Collection Large Page support

Large Pages (also known as Huge Pages on Linux) is a feature where the operating system is able to establish memory regions larger than the native page size (often 4K) to improve performance of the application requesting these large pages.

The Garbage Collector can now be configured with the `GCLargePages` setting as an opt-in feature to choose to allocate large pages on Windows.

Windows Desktop & COM

.NET Core SDK Windows Installer

The MSI installer for Windows has changed starting with .NET Core 3.0. The SDK installers will now upgrade SDK feature-band releases in place. Feature bands are defined in the *hundreds* groups in the *patch* section of the version number. For example, `3.0.101` and `3.0.201` are versions in two different feature bands while `3.0.101` and `3.0.199` are in the same feature band. And, when .NET Core SDK `3.0.101` is installed, .NET Core SDK `3.0.100` will be removed from the machine if it exists. When .NET Core SDK `3.0.200` is installed on the same machine, .NET Core SDK `3.0.101` won't be removed.

For more information about versioning, see [Overview of how .NET Core is versioned](#).

Windows desktop

.NET Core 3.0 supports Windows desktop applications using Windows Presentation Foundation (WPF) and

Windows Forms. These frameworks also support using modern controls and Fluent styling from the Windows UI XAML Library (WinUI) via [XAML islands](#).

The Windows Desktop component is part of the Windows .NET Core 3.0 SDK.

You can create a new WPF or Windows Forms app with the following `dotnet` commands:

```
dotnet new wpf  
dotnet new winforms
```

Visual Studio 2019 adds [New Project](#) templates for .NET Core 3.0 Windows Forms and WPF.

For more information about how to port an existing .NET Framework application, see [Port WPF projects](#) and [Port Windows Forms projects](#).

WinForms high DPI

.NET Core Windows Forms applications can set high DPI mode with

[Application.SetHighDpiMode\(HighDpiMode\)](#). The `SetHighDpiMode` method sets the corresponding high DPI mode unless the setting has been set by other means like `App.Manifest` or P/Invoke before `Application.Run`.

The possible `highDpiMode` values, as expressed by the [System.Windows.Forms.HighDpiMode](#) enum are:

- `DpiUnaware`
- `SystemAware`
- `PerMonitor`
- `PerMonitorV2`
- `DpiUnawareGdiScaled`

For more information about high DPI modes, see [High DPI Desktop Application Development on Windows](#).

Create COM components

On Windows, you can now create COM-callable managed components. This capability is critical to use .NET Core with COM add-in models and also to provide parity with .NET Framework.

Unlike .NET Framework where the *mscoree.dll* was used as the COM server, .NET Core will add a native launcher dll to the *b/in* directory when you build your COM component.

For an example of how to create a COM component and consume it, see the [COM Demo](#).

Windows Native Interop

Windows offers a rich native API in the form of flat C APIs, COM, and WinRT. While .NET Core supports [P/Invoke](#), .NET Core 3.0 adds the ability to [CoCreate COM APIs](#) and [Activate WinRT APIs](#). For a code example, see the [Excel Demo](#).

MSIX Deployment

[MSIX](#) is a new Windows application package format. It can be used to deploy .NET Core 3.0 desktop applications to Windows 10.

The [Windows Application Packaging Project](#), available in Visual Studio 2019, allows you to create MSIX packages with [self-contained](#) .NET Core applications.

The .NET Core project file must specify the supported runtimes in the `<RuntimeIdentifiers>` property:

```
<RuntimeIdentifiers>win-x86;win-x64</RuntimeIdentifiers>
```

Linux improvements

SerialPort for Linux

.NET Core 3.0 provides basic support for `System.IO.Ports.SerialPort` on Linux.

Previously, .NET Core only supported using `SerialPort` on Windows.

For more information about the limited support for the serial port on Linux, see [GitHub issue #33146](#).

Docker and cgroup memory Limits

Running .NET Core 3.0 on Linux with Docker works better with cgroup memory limits. Running a Docker container with memory limits, such as with `docker run -m`, changes how .NET Core behaves.

- Default Garbage Collector (GC) heap size: maximum of 20 mb or 75% of the memory limit on the container.
- Explicit size can be set as an absolute number or percentage of cgroup limit.
- Minimum reserved segment size per GC heap is 16 mb. This size reduces the number of heaps that are created on machines.

GPIO Support for Raspberry Pi

Two packages have been released to NuGet that you can use for GPIO programming:

- `System.Device.Gpio`
- `Iot.Device.Bindings`

The GPIO packages include APIs for *GPIO*, *SPI*, *I2C*, and *PWM* devices. The IoT bindings package includes device bindings. For more information, see the [devices GitHub repo](#).

Arm64 Linux support

.NET Core 3.0 adds support for Arm64 for Linux. The primary use case for Arm64 is currently with IoT scenarios. For more information, see [.NET Core Arm64 Status](#).

Docker images for [.NET Core on Arm64](#) are available for Alpine, Debian, and Ubuntu.

NOTE

Support for the macOS Arm64 (or "Apple Silicon") and Windows Arm64 operating systems was later added in .NET 6.

Security

TLS 1.3 & OpenSSL 1.1.1 on Linux

.NET Core now takes advantage of [TLS 1.3 support in OpenSSL 1.1.1](#), when it's available in a given environment.

With TLS 1.3:

- Connection times are improved with reduced round trips required between the client and server.
- Improved security because of the removal of various obsolete and insecure cryptographic algorithms.

When available, .NET Core 3.0 uses `OpenSSL 1.1.1`, `OpenSSL 1.1.0`, or `OpenSSL 1.0.2` on a Linux system.

When `OpenSSL 1.1.1` is available, both `System.Net.Security.SslStream` and `System.Net.Http.HttpClient` types will use TLS 1.3 (assuming both the client and server support TLS 1.3).

IMPORTANT

Windows and macOS do not yet support TLS 1.3.

The following C# 8.0 example demonstrates .NET Core 3.0 on Ubuntu 18.10 connecting to

<https://www.cloudflare.com>:

```
using System;
using System.Net.Security;
using System.Net.Sockets;
using System.Threading.Tasks;

namespace whats_new
{
    public static class TLS
    {
        public static async Task ConnectCloudFlare()
        {
            var targetHost = "www.cloudflare.com";

            using TcpClient tcpClient = new TcpClient();

            await tcpClient.ConnectAsync(targetHost, 443);

            using SslStream sslStream = new SslStream(tcpClient.GetStream());

            await sslStream.AuthenticateAsClientAsync(targetHost);
            await Console.Out.WriteLineAsync($"Connected to {targetHost} with {sslStream.SslProtocol}");
        }
    }
}
```

Cryptography ciphers

.NET Core 3.0 adds support for AES-GCM and AES-CCM ciphers, implemented with [System.Security.Cryptography.AesGcm](#) and [System.Security.Cryptography.AesCcm](#) respectively. These algorithms are both [Authenticated Encryption with Association Data \(AEAD\) algorithms](#).

The following code demonstrates using `AesGcm` cipher to encrypt and decrypt random data.

```

using System;
using System.Linq;
using System.Security.Cryptography;

namespace whats_new
{
    public static class Cipher
    {
        public static void Run()
        {
            // key should be: pre-known, derived, or transported via another channel, such as RSA encryption
            byte[] key = new byte[16];
            RandomNumberGenerator.Fill(key);

            byte[] nonce = new byte[12];
            RandomNumberGenerator.Fill(nonce);

            // normally this would be your data
            byte[] dataToEncrypt = new byte[1234];
            byte[] associatedData = new byte[333];
            RandomNumberGenerator.Fill(dataToEncrypt);
            RandomNumberGenerator.Fill(associatedData);

            // these will be filled during the encryption
            byte[] tag = new byte[16];
            byte[] ciphertext = new byte[dataToEncrypt.Length];

            using (AesGcm aesGcm = new AesGcm(key))
            {
                aesGcm.Encrypt(nonce, dataToEncrypt, ciphertext, tag, associatedData);
            }

            // tag, nonce, ciphertext, associatedData should be sent to the other part

            byte[] decryptedData = new byte[ciphertext.Length];

            using (AesGcm aesGcm = new AesGcm(key))
            {
                aesGcm.Decrypt(nonce, ciphertext, tag, decryptedData, associatedData);
            }

            // do something with the data
            // this should always print that data is the same
            Console.WriteLine($"AES-GCM: Decrypted data is {(dataToEncrypt.SequenceEqual(decryptedData) ? "the same as" : "different than")}{original data."});
        }
    }
}

```

Cryptographic Key Import/Export

.NET Core 3.0 supports the import and export of asymmetric public and private keys from standard formats. You don't need to use an X.509 certificate.

All key types, such as *RSA*, *DSA*, *ECDsa*, and *ECDiffieHellman*, support the following formats:

- **Public Key**
 - X.509 SubjectPublicKeyInfo
- **Private key**
 - PKCS#8 PrivateKeyInfo
 - PKCS#8 EncryptedPrivateKeyInfo

RSA keys also support:

- **Public Key**

- PKCS#1 RSA PublicKey

- **Private key**

- PKCS#1 RSAPrivateKey

The export methods produce DER-encoded binary data, and the import methods expect the same. If a key is stored in the text-friendly PEM format, the caller will need to base64-decode the content before calling an import method.

```
using System;
using System.Security.Cryptography;

namespace whats_new
{
    public static class RSATest
    {
        public static void Run(string keyFile)
        {
            using var rsa = RSA.Create();

            byte[] keyBytes = System.IO.File.ReadAllBytes(keyFile);
            rsa.ImportRSAPrivateKey(keyBytes, out int bytesRead);

            Console.WriteLine($"Read {bytesRead} bytes, {keyBytes.Length - bytesRead} extra byte(s) in
file.");
            RSAParameters rsaParameters = rsa.ExportParameters(true);
            Console.WriteLine(BitConverter.ToString(rsaParameters.D));
        }
    }
}
```

PKCS#8 files can be inspected with [System.Security.Cryptography.Pkcs.Pkcs8PrivateKeyInfo](#) and PFX/PKCS#12 files can be inspected with [System.Security.Cryptography.Pkcs.Pkcs12Info](#). PFX/PKCS#12 files can be manipulated with [System.Security.Cryptography.Pkcs.Pkcs12Builder](#).

.NET Core 3.0 API changes

Ranges and indices

The new [System.Index](#) type can be used for indexing. You can create one from an `int` that counts from the beginning, or with a prefix `^` operator (C#) that counts from the end:

```
Index i1 = 3; // number 3 from beginning
Index i2 = ^4; // number 4 from end
int[] a = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
Console.WriteLine($"{a[i1]}, {a[i2]}"); // "3, 6"
```

There's also the [System.Range](#) type, which consists of two `Index` values, one for the start and one for the end, and can be written with a `x..y` range expression (C#). You can then index with a `Range`, which produces a slice:

```
var slice = a[i1..i2]; // { 3, 4, 5 }
```

For more information, see the [ranges and indices tutorial](#).

Async streams

The [IAsyncEnumerable<T>](#) type is a new asynchronous version of [IEnumerable<T>](#). The language lets you

`await foreach` over `IAsyncEnumerable<T>` to consume their elements, and use `yield return` to them to produce elements.

The following example demonstrates both production and consumption of async streams. The `foreach` statement is async and itself uses `yield return` to produce an async stream for callers. This pattern (using `yield return`) is the recommended model for producing async streams.

```
async IAsyncEnumerable<int> GetBigResultsAsync()
{
    await foreach (var result in GetResultsAsync())
    {
        if (result > 20) yield return result;
    }
}
```

In addition to being able to `await foreach`, you can also create async iterators, for example, an iterator that returns an `IAsyncEnumerable/IAsyncEnumerator` that you can both `await` and `yield` in. For objects that need to be disposed, you can use `IAsyncDisposable`, which various BCL types implement, such as `Stream` and `Timer`.

For more information, see the [async streams tutorial](#).

IEEE Floating-point

Floating point APIs are being updated to comply with [IEEE 754-2008 revision](#). The goal of these changes is to expose all **required** operations and ensure that they're behaviorally compliant with the IEEE spec. For more information about floating-point improvements, see the [Floating-Point Parsing and Formatting improvements in .NET Core 3.0](#) blog post.

Parsing and formatting fixes include:

- Correctly parse and round inputs of any length.
- Correctly parse and format negative zero.
- Correctly parse `Infinity` and `Nan` by doing a case-insensitive check and allowing an optional preceding `+` where applicable.

New `System.Math` APIs include:

- `BitIncrement(Double)` and `BitDecrement(Double)`

Corresponds to the `nextUp` and `nextDown` IEEE operations. They return the smallest floating-point number that compares greater or lesser than the input (respectively). For example, `Math.BitIncrement(0.0)` would return `double.Epsilon`.

- `MaxMagnitude(Double, Double)` and `MinMagnitude(Double, Double)`

Corresponds to the `maxNumMag` and `minNumMag` IEEE operations, they return the value that is greater or lesser in magnitude of the two inputs (respectively). For example, `Math.MaxMagnitude(2.0, -3.0)` would return `-3.0`.

- `ILogB(Double)`

Corresponds to the `logB` IEEE operation that returns an integral value, it returns the integral base-2 log of the input parameter. This method is effectively the same as `floor(log2(x))`, but done with minimal rounding error.

- `ScaleB(Double, Int32)`

Corresponds to the `scaleB` IEEE operation that takes an integral value, it returns effectively `x * pow(2, n)`, but is done with minimal rounding error.

- `Log2(Double)`

Corresponds to the `log2` IEEE operation, it returns the base-2 logarithm. It minimizes rounding error.

- [FusedMultiplyAdd\(Double, Double, Double\)](#)

Corresponds to the `fma` IEEE operation, it performs a fused multiply add. That is, it does $(x * y) + z$ as a single operation, thereby minimizing the rounding error. An example is `FusedMultiplyAdd(1e308, 2.0, -1e308)`, which returns `1e308`. The regular `(1e308 * 2.0) - 1e308` returns `double.PositiveInfinity`.

- [CopySign\(Double, Double\)](#)

Corresponds to the `copySign` IEEE operation, it returns the value of `x`, but with the sign of `y`.

.NET Platform-Dependent Intrinsics

APIs have been added that allow access to certain perf-oriented CPU instructions, such as the **SIMD** or **Bit Manipulation instruction** sets. These instructions can help achieve significant performance improvements in certain scenarios, such as processing data efficiently in parallel.

Where appropriate, the .NET libraries have begun using these instructions to improve performance.

For more information, see [.NET Platform-Dependent Intrinsics](#).

Improved .NET Core Version APIs

Starting with .NET Core 3.0, the version APIs provided with .NET Core now return the information you expect.

For example:

```
System.Console.WriteLine($"Environment.Version: {System.Environment.Version}");  
  
// Old result  
//   Environment.Version: 4.0.30319.42000  
//  
// New result  
//   Environment.Version: 3.0.0
```

```
System.Console.WriteLine($"RuntimeInformation.FrameworkDescription:  
{System.Runtime.InteropServices.RuntimeInformation.FrameworkDescription}");  
  
// Old result  
//   RuntimeInformation.FrameworkDescription: .NET Core 4.6.27415.71  
//  
// New result (notice the value includes any preview release information)  
//   RuntimeInformation.FrameworkDescription: .NET Core 3.0.0-preview4-27615-11
```

WARNING

Breaking change. This is technically a breaking change because the versioning scheme has changed.

Fast built-in JSON support

.NET users have largely relied on [Newtonsoft.Json](#) and other popular JSON libraries, which continue to be good choices. `Newtonsoft.Json` uses .NET strings as its base datatype, which is UTF-16 under the hood.

The new built-in JSON support is high-performance, low allocation, and works with UTF-8 encoded JSON text. For more information about the `System.Text.Json` namespace and types, see the following articles:

- [JSON serialization in .NET - overview](#)
- [How to serialize and deserialize JSON in .NET](#).
- [How to migrate from Newtonsoft.Json to System.Text.Json](#)

HTTP/2 support

The `System.Net.Http.HttpClient` type supports the HTTP/2 protocol. If HTTP/2 is enabled, the HTTP protocol

version is negotiated via TLS/ALPN, and HTTP/2 is used if the server elects to use it.

The default protocol remains HTTP/1.1, but HTTP/2 can be enabled in two different ways. First, you can set the HTTP request message to use HTTP/2:

```
var client = new HttpClient() { BaseAddress = new Uri("https://localhost:5001") };

// HTTP/1.1 request
using (var response = await client.GetAsync("/"))
    Console.WriteLine(response.Content);

// HTTP/2 request
using (var request = new HttpRequestMessage(HttpMethod.Get, "/") { Version = new Version(2, 0) })
using (var response = await client.SendAsync(request))
    Console.WriteLine(response.Content);
```

Second, you can change [HttpClient](#) to use HTTP/2 by default:

```
var client = new HttpClient()
{
    BaseAddress = new Uri("https://localhost:5001"),
    DefaultRequestVersion = new Version(2, 0)
};

// HTTP/2 is default
using (var response = await client.GetAsync("/"))
    Console.WriteLine(response.Content);
```

Many times when you're developing an application, you want to use an unencrypted connection. If you know the target endpoint will be using HTTP/2, you can turn on unencrypted connections for HTTP/2. You can turn it on by setting the `DOTNET_SYSTEM_NET_HTTP_SOCKETSHTTPHANDLER_HTTP2UNENCRYPTEDSUPPORT` environment variable to `1` or by enabling it in the app context:

```
AppContext.SetSwitch("System.Net.Http.SocketsHttpHandler.Http2UnencryptedSupport", true);
```

Next steps

- [Review the breaking changes between .NET Core 2.2 and 3.0.](#)
- [Review the breaking changes in .NET Core 3.0 for Windows Forms apps.](#)

Breaking changes in .NET Core 3.0

9/20/2022 • 69 minutes to read • [Edit Online](#)

If you're migrating to version 3.0 of .NET Core, ASP.NET Core, or EF Core, the breaking changes listed in this article may affect your app.

ASP.NET Core

- [Obsolete Antiforgery, CORS, Diagnostics, MVC, and Routing APIs removed](#)
- ["Pubternal" APIs removed](#)
- [Authentication: Google+ deprecation](#)
- [Authentication: HttpContext.Authentication property removed](#)
- [Authentication: Newtonsoft.Json types replaced](#)
- [Authentication: OAuthHandler ExchangeCodeAsync signature changed](#)
- [Authorization: AddAuthorization overload moved to different assembly](#)
- [Authorization: IAllowAnonymous removed from AuthorizationFilterContext.Filters](#)
- [Authorization: IAuthorizationPolicyProvider implementations require new method](#)
- [Caching: CompactOnMemoryPressure property removed](#)
- [Caching: Microsoft.Extensions.Caching.SqlServer uses new SqlClient package](#)
- [Caching: ResponseCaching "pubternal" types changed to internal](#)
- [Data Protection: DataProtection.Blobs uses new Azure Storage APIs](#)
- [Hosting: AspNetCoreModule V1 removed from Windows Hosting Bundle](#)
- [Hosting: Generic host restricts Startup constructor injection](#)
- [Hosting: HTTPS redirection enabled for IIS out-of-process apps](#)
- [Hosting: IHostingEnvironment and IApplicationLifetime types replaced](#)
- [Hosting: ObjectPoolProvider removed from WebHostBuilder dependencies](#)
- [HTTP: DefaultHttpContext extensibility removed](#)
- [HTTP: HeaderNames fields changed to static readonly](#)
- [HTTP: Response body infrastructure changes](#)
- [HTTP: Some cookie SameSite default values changed](#)
- [HTTP: Synchronous IO disabled by default](#)
- [Identity: AddDefaultUI method overload removed](#)
- [Identity: UI Bootstrap version change](#)
- [Identity: SignInAsync throws exception for unauthenticated identity](#)
- [Identity: SignInManager constructor accepts new parameter](#)
- [Identity: UI uses static web assets feature](#)
- [Kestrel: Connection adapters removed](#)
- [Kestrel: Empty HTTPS assembly removed](#)
- [Kestrel: Request trailer headers moved to new collection](#)
- [Kestrel: Transport abstraction layer changes](#)
- [Localization: APIs marked obsolete](#)
- [Logging: DebugLogger class made internal](#)
- [MVC: Controller action Async suffix removed](#)
- [MVC: JsonResult moved to Microsoft.AspNetCore.Mvc.Core](#)
- [MVC: Precompilation tool deprecated](#)
- [MVC: Types changed to internal](#)
- [MVC: Web API compatibility shim removed](#)
- [Razor: RazorTemplateEngine API removed](#)
- [Razor: Runtime compilation moved to a package](#)
- [Session state: Obsolete APIs removed](#)
- [Shared framework: Assembly removal from Microsoft.AspNetCore.App](#)
- [Shared framework: Microsoft.AspNetCore.All removed](#)
- [SignalR: HandshakeProtocol.SuccessHandshakeData replaced](#)
- [SignalR: HubConnection methods removed](#)
- [SignalR: HubConnectionContext constructors changed](#)
- [SignalR: JavaScript client package name change](#)
- [SignalR: Obsolete APIs](#)
- [SPAs: SpaServices and NodeServices marked obsolete](#)
- [SPAs: SpaServices and NodeServices console logger fallback default change](#)
- [Target framework: .NET Framework not supported](#)

Obsolete Antiforgery, CORS, Diagnostics, MVC, and Routing APIs removed

Obsolete members and compatibility switches in ASP.NET Core 2.2 were removed.

Version introduced

3.0

Reason for change

Improvement of API surface over time.

Recommended action

While targeting .NET Core 2.2, follow the guidance in the obsolete build messages to adopt new APIs instead.

Category

ASP.NET Core

Affected APIs

The following types and members were marked as obsolete for ASP.NET Core 2.1 and 2.2:

Types

- [Microsoft.AspNetCore.Diagnostics.Views.WelcomePage](#)
- [Microsoft.AspNetCore.DiagnosticsViewPage.Views.AttributeValue](#)
- [Microsoft.AspNetCore.DiagnosticsViewPage.Views.BaseView](#)
- [Microsoft.AspNetCore.DiagnosticsViewPage.Views.HelperResult](#)
- [Microsoft.AspNetCore.Mvc.Formatters.Xml.ProblemDetails21Wrapper](#)
- [Microsoft.AspNetCore.Mvc.Formatters.Xml.ValidationProblemDetails21Wrapper](#)
- [Microsoft.AspNetCore.Mvc.Razor.Compilation.ViewsFeatureProvider](#)
- [Microsoft.AspNetCore.Mvc.RazorPages.Infrastructure.PageArgumentBinder](#)
- [Microsoft.AspNetCore.Routing.IRouteValuesAddressMetadata](#)
- [Microsoft.AspNetCore.Routing.RouteValuesAddressMetadata](#)

Constructors

- [Microsoft.AspNetCore.Cors.Infrastructure.CorsService\(IOptions{CorsOptions}\)](#)
- [Microsoft.AspNetCore.Routing.Tree.TreeRouteBuilder\(ILoggerFactory,UrlEncoder,ObjectPool{UriBuildingContext},IInlineConstraintResolver\)](#)
- [Microsoft.AspNetCore.Mvc.Formatters.OutputFormatterCanWriteContext](#)
- [Microsoft.AspNetCore.Mvc.ApiExplorer.DefaultApiDescriptionProvider\(IOptions{MvcOptions},IInlineConstraintResolver,IModelMetadataProvider\)](#)
- [Microsoft.AspNetCore.Mvc.ApiExplorer.DefaultApiDescriptionProvider\(IOptions{MvcOptions},IInlineConstraintResolver,IModelMetadataProvider,IActionDescriptor\)](#)
- [Microsoft.AspNetCore.Mvc.Formatters.FormatFilter\(IOptions{MvcOptions}\)](#)
- [Microsoft.AspNetCore.Mvc.ModelBinding.Binders.ArrayModelBinder`1\(IModelBinder\)](#)
- [Microsoft.AspNetCore.Mvc.ModelBinding.Binders.ByteArrayModelBinder](#)
- [Microsoft.AspNetCore.Mvc.ModelBinding.Binders.CollectionModelBinder`1\(IModelBinder\)](#)
- [Microsoft.AspNetCore.Mvc.ModelBinding.Binders.ComplexTypeModelBinder\(IDictionary`2\)](#)
- [Microsoft.AspNetCore.Mvc.ModelBinding.Binders.DictionaryModelBinder`2\(IModelBinder,IModelBinder\)](#)
- [Microsoft.AspNetCore.Mvc.ModelBinding.Binders.DoubleModelBinder\(System.Globalization.NumberStyles\)](#)
- [Microsoft.AspNetCore.Mvc.ModelBinding.Binders.FloatModelBinder\(System.Globalization.NumberStyles\)](#)
- [Microsoft.AspNetCore.Mvc.ModelBinding.Binders.FormCollectionModelBinder](#)
- [Microsoft.AspNetCore.Mvc.ModelBinding.Binders.FormFileModelBinder](#)
- [Microsoft.AspNetCore.Mvc.ModelBinding.Binders.HeaderModelBinder](#)
- [Microsoft.AspNetCore.Mvc.ModelBinding.Binders.KeyValuePairModelBinder`2\(IModelBinder,IModelBinder\)](#)
- [Microsoft.AspNetCore.Mvc.ModelBinding.Binders.SimpleTypeModelBinder\(System.Type\)](#)
- [Microsoft.AspNetCore.Mvc.ModelBinding.ModelAttributes\(IEnumerable{System.Object}\)](#)
- [Microsoft.AspNetCore.Mvc.ModelBinding.ModelAttributes\(IEnumerable{System.Object}\)](#)
- [Microsoft.AspNetCore.Mvc.ModelBinding.ModelBinderFactory\(IModelMetadataProvider,IOptions{MvcOptions}\)](#)
- [Microsoft.AspNetCore.Mvc.ModelBinding.ParameterBinder\(IModelMetadataProvider,IModelBinderFactory,IObjectModelValidator\)](#)
- [Microsoft.AspNetCore.Mvc.Routing.KnownRouteValueConstraint](#)
- [Microsoft.AspNetCore.Mvc.Formatters.XmlDataContractSerializerInputFormatter](#)
- [Microsoft.AspNetCore.Mvc.Formatters.XmlDataContractSerializerInputFormatter\(System.Boolean\)](#)
- [Microsoft.AspNetCore.Mvc.Formatters.XmlDataContractSerializerInputFormatter\(MvcOptions\)](#)
- [Microsoft.AspNetCore.Mvc.Formatters.XmlSerializerInputFormatter](#)
- [Microsoft.AspNetCore.Mvc.Formatters.XmlSerializerInputFormatter\(System.Boolean\)](#)
- [Microsoft.AspNetCore.Mvc.Formatters.XmlSerializerInputFormatter\(MvcOptions\)](#)
- [Microsoft.AspNetCore.Mvc.TagHelpers.ImageTagHelper\(IHostingEnvironment,IMemoryCache,HtmlEncoder,IUrlHelperFactory\)](#)

- `Microsoft.AspNetCore.Mvc.TagHelpers.LinkTagHelper(IHostingEnvironment, IMemoryCache, HtmlEncoder, JavaScriptEncoder, IUrlHelperFactory)`
- `Microsoft.AspNetCore.Mvc.TagHelpers.ScriptTagHelper(IHostingEnvironment, IMemoryCache, HtmlEncoder, JavaScriptEncoder, IUrlHelperFactory)`
- `Microsoft.AspNetCore.Mvc.RazorPages.Infrastructure.RazorPageAdapter(RazorPageBase)`

Properties

- `Microsoft.AspNetCore.Antiforgery.AntiforgeryOptions.CookieDomain`
- `Microsoft.AspNetCore.Antiforgery.AntiforgeryOptions.CookieName`
- `Microsoft.AspNetCore.Antiforgery.AntiforgeryOptions.CookiePath`
- `Microsoft.AspNetCore.Antiforgery.AntiforgeryOptions.RequireSsl`
- `Microsoft.AspNetCore.Mvc.ApiBehaviorOptions.AllowInferringBindingSourceForCollectionTypesAsFromQuery`
- `Microsoft.AspNetCore.Mvc.ApiBehaviorOptions.SuppressUseValidationProblemDetailsForInvalidModelStateResponses`
- `Microsoft.AspNetCore.Mvc.CookieTempDataProviderOptions.CookieName`
- `Microsoft.AspNetCore.Mvc.CookieTempDataProviderOptions.Domain`
- `Microsoft.AspNetCore.Mvc.CookieTempDataProviderOptions.Path`
- `Microsoft.AspNetCore.Mvc.DataAnnotations.MvcDataAnnotationsLocalizationOptions.AllowDataAnnotationsLocalizationForEnumDisplayAttributes`
- `Microsoft.AspNetCore.Mvc.Formatters.Xml.MvcXmlOptions.AllowRfc7807CompliantProblemDetailsFormat`
- `Microsoft.AspNetCore.Mvc.MvcOptions.AllowBindingHeaderValuesToNonStringModelTypes`
- `Microsoft.AspNetCore.Mvc.MvcOptions.AllowCombiningAuthorizeFilters`
- `Microsoft.AspNetCore.Mvc.MvcOptions.AllowShortCircuitingValidationWhenNoValidatorsArePresent`
- `Microsoft.AspNetCore.Mvc.MvcOptions.AllowValidatingTopLevelNodes`
- `Microsoft.AspNetCore.Mvc.MvcOptions.InputFormatterExceptionPolicy`
- `Microsoft.AspNetCore.Mvc.MvcOptions.SuppressBindingUndefinedValueToEnumType`
- `Microsoft.AspNetCore.Mvc.MvcViewOptions.AllowRenderingMaxLengthAttribute`
- `Microsoft.AspNetCore.Mvc.MvcViewOptions.SuppressTempDataAttributePrefix`
- `Microsoft.AspNetCore.Mvc.RazorPages.RazorPagesOptions.AllowAreas`
- `Microsoft.AspNetCore.Mvc.RazorPages.RazorPagesOptions.AllowDefaultHandlingForOptionsRequests`
- `Microsoft.AspNetCore.Mvc.RazorPages.RazorPagesOptions.AllowMappingHeadRequestsToGetHandler`

Methods

- `Microsoft.AspNetCore.Mvc.LocalRedirectResult.ExecuteResult(ActionContext)`
- `Microsoft.AspNetCore.Mvc.RedirectResult.ExecuteResult(ActionContext)`
- `Microsoft.AspNetCore.Mvc.RedirectToActionResult.ExecuteResult(ActionContext)`
- `Microsoft.AspNetCore.Mvc.RedirectToPageResult.ExecuteResult(ActionContext)`
- `Microsoft.AspNetCore.Mvc.RedirectToRouteResult.ExecuteResult(ActionContext)`
- `Microsoft.AspNetCore.Mvc.ModelBinding.ParameterBinder.BindModelAsync(ActionContext, IValueProvider, ParameterDescriptor)`
- `Microsoft.AspNetCore.Mvc.ModelBinding.ParameterBinder.BindModelAsync(ActionContext, IValueProvider, ParameterDescriptor, Object)`

"Pubternal" APIs removed

To better maintain the public API surface of ASP.NET Core, most of the types in `*.Internal` namespaces (referred to as "pubternal" APIs) have become truly internal. Members in these namespaces were never meant to be supported as public-facing APIs. The APIs could break in minor releases and often did. Code that depends on these APIs breaks when updating to ASP.NET Core 3.0.

For more information, see [dotnet/aspnetcore#4932](#) and [dotnet/aspnetcore#11312](#).

Version introduced

3.0

Old behavior

The affected APIs are marked with the `public` access modifier and exist in `*.Internal` namespaces.

New behavior

The affected APIs are marked with the `internal` access modifier and can no longer be used by code outside that assembly.

Reason for change

The guidance for these "pubternal" APIs was that they:

- Could change without notice.
- Weren't subject to .NET policies to prevent breaking changes.

Leaving the APIs `public` (even in the `*.Internal` namespaces) was confusing to customers.

Recommended action

Stop using these "pubternal" APIs. If you have questions about alternate APIs, open an issue in the [dotnet/aspnetcore](#) repository.

For example, consider the following HTTP request buffering code in an ASP.NET Core 2.2 project. The `EnableRewind` extension method exists in the `Microsoft.AspNetCore.Http.Internal` namespace.

```
HttpContext.Request.EnableRewind();
```

In an ASP.NET Core 3.0 project, replace the `EnableRewind` call with a call to the `EnableBuffering` extension method. The request buffering feature works as it did in the past. `EnableBuffering` calls the now `internal` API.

```
HttpContext.Request.EnableBuffering();
```

Category

ASP.NET Core

Affected APIs

All APIs in the `Microsoft.AspNetCore.*` and `Microsoft.Extensions.*` namespaces that have an `Internal` segment in the namespace name. For example:

- `Microsoft.AspNetCore.Authentication.Internal`
- `Microsoft.AspNetCore.Builder.Internal`
- `Microsoft.AspNetCore.DataProtection.Cng.Internal`
- `Microsoft.AspNetCore.DataProtection.Internal`
- `Microsoft.AspNetCore.Hosting.Internal`
- `Microsoft.AspNetCore.Http.Internal`
- `Microsoft.AspNetCore.Mvc.Core.Infrastructure`
- `Microsoft.AspNetCore.Mvc.Core.Internal`
- `Microsoft.AspNetCore.Mvc.Cors.Internal`
- `Microsoft.AspNetCore.Mvc.DataAnnotations.Internal`
- `Microsoft.AspNetCore.Mvc.Formatters.Internal`
- `Microsoft.AspNetCore.Mvc.Formatters.Json.Internal`
- `Microsoft.AspNetCore.Mvc.Formatters.Xml.Internal`
- `Microsoft.AspNetCore.Mvc.Internal`
- `Microsoft.AspNetCore.Mvc.ModelBinding.Internal`
- `Microsoft.AspNetCore.Mvc.Razor.Internal`
- `Microsoft.AspNetCore.Mvc.RazorPages.Internal`
- `Microsoft.AspNetCore.Mvc.TagHelpers.Internal`
- `Microsoft.AspNetCore.Mvc.ViewFeatures.Internal`
- `Microsoft.AspNetCore.Rewrite.Internal`
- `Microsoft.AspNetCore.Routing.Internal`
- `Microsoft.AspNetCore.Server.Kestrel.Core.Adapter.Internal`
- `Microsoft.AspNetCore.Server.Kestrel.Core.Internal.Http`
- `Microsoft.AspNetCore.Server.Kestrel.Core.Internal.Infrastructure`
- `Microsoft.AspNetCore.Server.Kestrel.Https.Internal`

Authentication: Google+ deprecated and replaced

Google is starting to [shut down](#) Google+ Sign-in for apps as early as January 28, 2019.

Change description

ASP.NET 4.x and ASP.NET Core have been using the Google+ Sign-in APIs to authenticate Google account users in web apps. The affected NuGet packages are `Microsoft.AspNetCore.Authentication.Google` for ASP.NET Core and `Microsoft.Owin.Security.Google` for `Microsoft.Owin` with ASP.NET Web Forms and MVC.

Google's replacement APIs use a different data source and format. The mitigations and solutions provided below account for the structural changes. Apps should verify the data itself still satisfies their requirements. For example, names, email addresses, profile links, and profile photos may provide subtly different values than before.

Version introduced

All versions. This change is external to ASP.NET Core.

Recommended action

Owin with ASP.NET Web Forms and MVC

For `Microsoft.Owin` 3.1.0 and later, a temporary mitigation is outlined [here](#). Apps should complete testing with the mitigation to check for changes in the data format. There are plans to release `Microsoft.Owin` 4.0.1 with a fix. Apps using any prior version should update to version 4.0.1.

ASP.NET Core 1.x

The mitigation in [Owin with ASP.NET Web Forms and MVC](#) can be adapted to ASP.NET Core 1.x. NuGet package

patches aren't planned because 1.x has reached [end of life](#) status.

ASP.NET Core 2.x

For `Microsoft.AspNetCore.Authentication.Google` version 2.x, replace your existing call to `AddGoogle` in `Startup.ConfigureServices` with the following code:

```
.AddGoogle(o =>
{
    o.ClientId = Configuration["Authentication:Google:ClientId"];
    o.ClientSecret = Configuration["Authentication:Google:ClientSecret"];
    o.UserInformationEndpoint = "https://www.googleapis.com/oauth2/v2/userinfo";
    o.ClaimActions.Clear();
    o.ClaimActions.MapJsonKey(ClaimTypes.NameIdentifier, "id");
    o.ClaimActions.MapJsonKey(ClaimTypes.Name, "name");
    o.ClaimActions.MapJsonKey(ClaimTypes.GivenName, "given_name");
    o.ClaimActions.MapJsonKey(ClaimTypes.Surname, "family_name");
    o.ClaimActions.MapJsonKey("urn:google:profile", "link");
    o.ClaimActions.MapJsonKey(ClaimTypes.Email, "email");
});
```

The February 2.1 and 2.2 patches incorporated the preceding reconfiguration as the new default. No patch is planned for ASP.NET Core 2.0 since it has reached [end of life](#).

ASP.NET Core 3.0

The mitigation given for ASP.NET Core 2.x can also be used for ASP.NET Core 3.0. In future 3.0 previews, the `Microsoft.AspNetCore.Authentication.Google` package may be removed. Users would be directed to `Microsoft.AspNetCore.Authentication.OpenIdConnect` instead. The following code shows how to replace `AddGoogle` with `AddOpenIdConnect` in `Startup.ConfigureServices`. This replacement can be used with ASP.NET Core 2.0 and later and can be adapted for ASP.NET Core 1.x as needed.

```
.AddOpenIdConnect("Google", o =>
{
    o.ClientId = Configuration["Authentication:Google:ClientId"];
    o.ClientSecret = Configuration["Authentication:Google:ClientSecret"];
    o.Authority = "https://accounts.google.com";
    o.ResponseType = OpenIdConnectResponseType.Code;
    o.CallbackPath = "/signin-google"; // Or register the default "/signin-oidc"
    o.Scope.Add("email");
});
JwtSecurityTokenHandler.DefaultInboundClaimTypeMap.Clear();
```

Category

ASP.NET Core

Affected APIs

[Microsoft.AspNetCore.Authentication.Google](#)

Authentication: `HttpContext.Authentication` property removed

The deprecated `Authentication` property on `HttpContext` has been removed.

Change description

As part of [dotnet/aspnetcore#6504](#), the deprecated `Authentication` property on `HttpContext` has been removed. The `Authentication` property has been deprecated since 2.0. A [migration guide](#) was published to migrate code using this deprecated property to the new replacement APIs. The remaining unused classes / APIs related to the old ASP.NET Core 1.x authentication stack were removed in commit [dotnet/aspnetcore@d7a7c65](#).

For discussion, see [dotnet/aspnetcore#6533](#).

Version introduced

3.0

Reason for change

ASP.NET Core 1.0 APIs have been replaced by extension methods in [Microsoft.AspNetCore.Authentication.AuthenticationHttpContextExtensions](#).

Recommended action

See the [migration guide](#).

Category

ASP.NET Core

Affected APIs

- [Microsoft.AspNetCore.Http.Authentication.AuthenticateInfo](#)
- [Microsoft.AspNetCore.Http.Authentication.AuthenticationManager](#)
- [Microsoft.AspNetCore.Http.Authentication.AuthenticationProperties](#)
- [Microsoft.AspNetCore.Http.Features.Authentication.AuthenticateContext](#)
- [Microsoft.AspNetCore.Http.Features.Authentication.ChallengeBehavior](#)
- [Microsoft.AspNetCore.Http.Features.Authentication.ChallengeContext](#)
- [Microsoft.AspNetCore.Http.Features.Authentication.DescribeSchemesContext](#)

- Microsoft.AspNetCore.Http.Features.Authentication.IAuthenticationHandler
- Microsoft.AspNetCore.Http.Features.Authentication.IHttpAuthenticationFeature.Handler
- Microsoft.AspNetCore.Http.Features.Authentication.SignInContext
- Microsoft.AspNetCore.Http.Features.Authentication.SignOutContext
- Microsoft.AspNetCore.Http.HttpContext.Authentication

Authentication: Newtonsoft.Json types replaced

In ASP.NET Core 3.0, `Newtonsoft.Json` types used in Authentication APIs have been replaced with `System.Text.Json` types. Except for the following cases, basic usage of the Authentication packages remains unaffected:

- Classes derived from the OAuth providers, such as those from `aspnet-contrib`.
- Advanced claim manipulation implementations.

For more information, see [dotnet/aspnetcore#7105](#). For discussion, see [dotnet/aspnetcore#7289](#).

Version introduced

3.0

Recommended action

For derived OAuth implementations, the most common change is to replace `JObject.Parse` with `JsonDocument.Parse` in the `CreateTicketAsync` override as shown [here](#). `JsonDocument` implements `IDisposable`.

The following list outlines known changes:

- `ClaimAction.Run(JObject, ClaimsIdentity, String)` becomes
`ClaimAction.Run(JsonElement userData, ClaimsIdentity identity, string issuer)`. All derived implementations of `ClaimAction` are similarly affected.
- `ClaimActionCollectionMapExtensions.MapCustomJson(ClaimActionCollection, String, Func<JObject, String>)` becomes
`MapCustomJson(this ClaimActionCollection collection, string claimType, Func<JsonElement, string> resolver)`
- `ClaimActionCollectionMapExtensions.MapCustomJson(ClaimActionCollection, String, String, Func<JObject, String>)` becomes
`MapCustomJson(this ClaimActionCollection collection, string claimType, string valueType, Func<JsonElement, string> resolver)`
- `OAuthCreatingTicketContext` has had one old constructor removed and the other replaced `JObject` with `JsonElement`. The `User` property and `RunClaimActions` method have been updated to match.
- `Success JObject` now accepts a parameter of type `JsonDocument` instead of `JObject`. The `Response` property has been updated to match. `OAuthTokenResponse` is now disposable and will be disposed by `OAuthHandler`. Derived OAuth implementations overriding `ExchangeCodeAsync` don't need to dispose the `JsonDocument` or `OAuthTokenResponse`.
- `UserInformationReceivedContext.User` changed from `JObject` to `JsonDocument`.
- `TwitterCreatingTicketContext.User` changed from `JObject` to `JsonElement`.
- The last parameter of `TwitterHandler.CreateTicketAsync(ClaimsIdentity, AuthenticationProperties, AccessToken, JObject)` changed from `JObject` to `JsonElement`. The replacement method is `TwitterHandler.CreateTicketAsync(ClaimsIdentity, AuthenticationProperties, AccessToken, JsonElement)`.

Category

ASP.NET Core

Affected APIs

- Microsoft.AspNetCore.Authentication.Facebook
- Microsoft.AspNetCore.Authentication.Google
- Microsoft.AspNetCore.Authentication.MicrosoftAccount
- Microsoft.AspNetCore.Authentication.OAuth
- Microsoft.AspNetCore.Authentication.OpenIdConnect
- Microsoft.AspNetCore.Authentication.Twitter

Authentication: OAuthHandler ExchangeCodeAsync signature changed

In ASP.NET Core 3.0, the signature of `OAuthHandler.ExchangeCodeAsync` was changed from:

```
protected virtual System.Threading.Tasks.Task<Microsoft.AspNetCore.Authentication.OAuth.OAuthTokenResponse>
ExchangeCodeAsync(string code, string redirectUri) { throw null; }
```

To:

```
protected virtual System.Threading.Tasks.Task<Microsoft.AspNetCore.Authentication.OAuth.OAuthTokenResponse>
ExchangeCodeAsync(Microsoft.AspNetCore.Authentication.OAuth.OAuthCodeExchangeContext context) { throw null;
}
```

Version introduced

3.0

Old behavior

The `code` and `redirectUri` strings were passed as separate arguments.

New behavior

`Code` and `RedirectUri` are properties on `OAuthCodeExchangeContext` that can be set via the `OAuthCodeExchangeContext` constructor. The new `OAuthCodeExchangeContext` type is the only argument passed to `OAuthHandler.ExchangeCodeAsync`.

Reason for change

This change allows additional parameters to be provided in a non-breaking manner. There's no need to create new `ExchangeCodeAsync` overloads.

Recommended action

Construct an `OAuthCodeExchangeContext` with the appropriate `code` and `redirectUri` values. An `AuthenticationProperties` instance must be provided. This single `OAuthCodeExchangeContext` instance can be passed to `OAuthHandler.ExchangeCodeAsync` instead of multiple arguments.

Category

ASP.NET Core

Affected APIs[OAuthHandler<TOptions>.ExchangeCodeAsync\(String, String\)](#)**Authorization: AddAuthorization overload moved to different assembly**

The core `AddAuthorization` methods that used to reside in `Microsoft.AspNetCore.Authorization` were renamed to `AddAuthorizationCore`. The old `AddAuthorization` methods still exist, but are in the `Microsoft.AspNetCore.Authorization.Policy` assembly instead. Apps using both methods should see no impact. Note that `Microsoft.AspNetCore.Authorization.Policy` now ships in the shared framework rather than a standalone package as discussed in [Shared framework: Assemblies removed from Microsoft.AspNetCore.App](#).

Version introduced

3.0

Old behavior

`AddAuthorization` methods existed in `Microsoft.AspNetCore.Authorization`.

New behavior

`AddAuthorization` methods exist in `Microsoft.AspNetCore.Authorization.Policy`. `AddAuthorizationCore` is the new name for the old methods.

Reason for change

`AddAuthorization` is a better method name for adding all common services needed for authorization.

Recommended action

Either add a reference to `Microsoft.AspNetCore.Authorization.Policy` or use `AddAuthorizationCore` instead.

Category

ASP.NET Core

Affected APIs[Microsoft.Extensions.DependencyInjection.AuthorizationServiceCollectionExtensions.AddAuthorization\(IServiceCollection collection, Action<AuthorizationOptions>\)](#)**Authorization: IAllowAnonymous removed from AuthorizationFilterContext.Filters**

As of ASP.NET Core 3.0, MVC doesn't add `AllowAnonymousFilters` for `[AllowAnonymous]` attributes that were discovered on controllers and action methods. This change is addressed locally for derivatives of `AuthorizeAttribute`, but it's a breaking change for `IAsyncAuthorizationFilter` and `IAuthorizationFilter` implementations. Such implementations wrapped in a `[TypeFilter]` attribute are a [popular](#) and supported way to achieve strongly-typed, attribute-based authorization when both configuration and dependency injection are required.

Version introduced

3.0

Old behavior

`IAllowAnonymous` appeared in the `AuthorizationFilterContext.Filters` collection. Testing for the interface's presence was a valid approach to override or disable the filter on individual controller methods.

New behavior

`IAllowAnonymous` no longer appears in the `AuthorizationFilterContext.Filters` collection.

`IAsyncAuthorizationFilter` implementations that are dependent on the old behavior typically cause intermittent HTTP 401 Unauthorized or HTTP 403 Forbidden responses.

Reason for change

A new endpoint routing strategy was introduced in ASP.NET Core 3.0.

Recommended action

Search the endpoint metadata for `IAllowAnonymous`. For example:

```
var endpoint = context.HttpContext.GetEndpoint();
if (endpoint?.Metadata?.GetMetadata<IAuthorizationPolicyProvider>() != null)
{
}
```

An example of this technique is seen in [this HasAllowAnonymous method](#).

Category

ASP.NET Core

Affected APIs

None

Authorization: IAuthorizationPolicyProvider implementations require new method

In ASP.NET Core 3.0, a new `GetFallbackPolicyAsync` method was added to `IAuthorizationPolicyProvider`. This fallback policy is used by the authorization middleware when no policy is specified.

For more information, see [dotnet/aspnetcore#9759](#).

Version introduced

3.0

Old behavior

Implementations of `IAuthorizationPolicyProvider` didn't require a `GetFallbackPolicyAsync` method.

New behavior

Implementations of `IAuthorizationPolicyProvider` require a `GetFallbackPolicyAsync` method.

Reason for change

A new method was needed for the new `AuthorizationMiddleware` to use when no policy is specified.

Recommended action

Add the `GetFallbackPolicyAsync` method to your implementations of `IAuthorizationPolicyProvider`.

Category

ASP.NET Core

Affected APIs

[Microsoft.AspNetCore.Authorization.IAuthorizationPolicyProvider](#)

Caching: CompactOnMemoryPressure property removed

The ASP.NET Core 3.0 release removed the [obsolete MemoryCacheOptions APIs](#).

Change description

This change is a follow-up to [aspnet/Caching#221](#). For discussion, see [dotnet/extensions#1062](#).

Version introduced

3.0

Old behavior

`MemoryCacheOptions.CompactOnMemoryPressure` property was available.

New behavior

The `MemoryCacheOptions.CompactOnMemoryPressure` property has been removed.

Reason for change

Automatically compacting the cache caused problems. To avoid unexpected behavior, the cache should only be compacted when needed.

Recommended action

To compact the cache, downcast to `MemoryCache` and call `Compact` when needed.

Category

ASP.NET Core

Affected APIs

[MemoryCacheOptions.CompactOnMemoryPressure](#)

Caching: Microsoft.Extensions.Caching.SqlServer uses new SqlClient package

The `Microsoft.Extensions.Caching.SqlServer` package will use the new `Microsoft.Data.SqlClient` package instead of `System.Data.SqlClient` package. This change could cause slight behavioral breaking changes. For more information, see [Introducing the new Microsoft.Data.SqlClient](#).

Version introduced

3.0

Old behavior

The `Microsoft.Extensions.Caching.SqlServer` package used the `System.Data.SqlClient` package.

New behavior

`Microsoft.Extensions.Caching.SqlServer` is now using the `Microsoft.Data.SqlClient` package.

Reason for change

`Microsoft.Data.SqlClient` is a new package that is built off of `System.Data.SqlClient`. It's where all new feature work will be done from now on.

Recommended action

Customers shouldn't need to worry about this breaking change unless they were using types returned by the `Microsoft.Extensions.Caching.SqlServer` package and casting them to `System.Data.SqlClient` types. For example, if someone was casting a `DbConnection` to the [old SqlConnection type](#), they would need to change the cast to the new `Microsoft.Data.SqlClient.SqlConnection` type.

Category

ASP.NET Core

Affected APIs

None

Caching: ResponseCaching "pubternal" types changed to internal

In ASP.NET Core 3.0, "pubternal" types in `ResponseCaching` have been changed to `internal`.

In addition, default implementations of `IResponseCachingPolicyProvider` and `IResponseCachingKeyProvider` are no longer added to services as part of the `AddResponseCaching` method.

Change description

In ASP.NET Core, "pubternal" types are declared as `public` but reside in a namespace suffixed with `.Internal`. While these types are public, they have no support policy and are subject to breaking changes. Unfortunately, accidental use of these types has been common, resulting in breaking changes to these projects and limiting the ability to maintain the framework.

Version introduced

3.0

Old behavior

These types were publicly visible, but unsupported.

New behavior

These types are now `internal`.

Reason for change

The `internal` scope better reflects the unsupported policy.

Recommended action

Copy types that are used by your app or library.

Category

ASP.NET Core

Affected APIs

- `Microsoft.AspNetCore.ResponseCaching.Internal.CachedResponse`
- `Microsoft.AspNetCore.ResponseCaching.Internal.CachedVaryByRules`
- `Microsoft.AspNetCore.ResponseCaching.Internal.IResponseCache`
- `Microsoft.AspNetCore.ResponseCaching.Internal.IResponseCacheEntry`
- `Microsoft.AspNetCore.ResponseCaching.Internal.IResponseCachingKeyProvider`
- `Microsoft.AspNetCore.ResponseCaching.Internal.IResponseCachingPolicyProvider`
- `Microsoft.AspNetCore.ResponseCaching.Internal.MemoryResponseCache`
- `Microsoft.AspNetCore.ResponseCaching.Internal.ResponseCachingContext`
- `Microsoft.AspNetCore.ResponseCaching.Internal.ResponseCachingKeyProvider`
- `Microsoft.AspNetCore.ResponseCaching.Internal.ResponseCachingPolicyProvider`
- `Microsoft.AspNetCore.ResponseCaching.ResponseCachingMiddleware.ResponseCachingMiddleware(RequestDelegate, IOptions<ResponseCachingOptions>, ILoggerFactory, IResponseCachingPolicyProvider, IResponseCache, IResponseCachingKeyProvider)`

Data Protection: DataProtection.Blobs uses new Azure Storage APIs

`Azure.Extensions.AspNetCore.DataProtection.Blobs` depends on the [Azure Storage libraries](#). These libraries renamed their assemblies, packages, and namespaces. Starting in ASP.NET Core 3.0,

`Azure.Extensions.AspNetCore.DataProtection.Blobs` uses the new `Azure.Storage.`-prefixed APIs and packages.

For questions about the Azure Storage APIs, use <https://github.com/Azure/azure-storage-net>. For discussion on this issue, see [dotnet/aspnetcore#19570](#).

Version introduced

3.0

Old behavior

The package referenced the `WindowsAzure.Storage` NuGet package. The package references the `Microsoft.Azure.Storage.Blob` NuGet package.

New behavior

The package references the `Azure.Storage.Blob` NuGet package.

Reason for change

This change allows `Azure.Extensions.AspNetCore.DataProtection.Blobs` to migrate to the recommended Azure Storage packages.

Recommended action

If you still need to use the older Azure Storage APIs with ASP.NET Core 3.0, add a direct dependency to the package `WindowsAzure.Storage` or `Microsoft.Azure.Storage`. This package can be installed alongside the new `Azure.Storage` APIs.

In many cases, the upgrade only involves changing the `using` statements to use the new namespaces:

```
- using Microsoft.WindowsAzure.Storage;
- using Microsoft.WindowsAzure.Storage.Blob;
- using Microsoft.Azure.Storage;
- using Microsoft.Azure.Storage.Blob;
+ using Azure.Storage;
+ using Azure.Storage.Blobs;
```

Category

ASP.NET Core

Affected APIs

None

Hosting: AspNetCoreModule V1 removed from Windows Hosting Bundle

Starting with ASP.NET Core 3.0, the Windows Hosting Bundle won't contain AspNetCoreModule (ANCM) V1.

ANCM V2 is backwards compatible with ANCM OutOfProcess and is recommended for use with ASP.NET Core 3.0 apps.

For discussion, see [dotnet/aspnetcore#7095](#).

Version introduced

3.0

Old behavior

ANCM V1 is included in the Windows Hosting Bundle.

New behavior

ANCM V1 isn't included in the Windows Hosting Bundle.

Reason for change

ANCM V2 is backwards compatible with ANCM OutOfProcess and is recommended for use with ASP.NET Core 3.0 apps.

Recommended action

Use ANCM V2 with ASP.NET Core 3.0 apps.

If ANCM V1 is required, it can be installed using the ASP.NET Core 2.1 or 2.2 Windows Hosting Bundle.

This change will break ASP.NET Core 3.0 apps that:

- Explicitly opted into using ANCM V1 with `<AspNetCoreModuleName>AspNetCoreModule</AspNetCoreModuleName>`.
- Have a custom `web.config` file with
`<add name="aspNetCore" path="*" verb="*" modules="AspNetCoreModule" resourceType="Unspecified" />`.

Category

ASP.NET Core

Affected APIs

None

Hosting: Generic host restricts Startup constructor injection

The only types the generic host supports for `Startup` class constructor injection are `IHostEnvironment`, `IWebHostEnvironment`, and `IConfiguration`. Apps using `WebHost` are unaffected.

Change description

Prior to ASP.NET Core 3.0, constructor injection could be used for arbitrary types in the `Startup` class's constructor. In ASP.NET Core 3.0, the web stack was replatformed onto the generic host library. You can see the change in the `Program.cs` file of the templates:

ASP.NET Core 2.x:

<https://github.com/dotnet/aspnetcore/blob/5cb615fcbe8559e49042e93394008077e30454c0/src/Templating/src/Microsoft.DotNet.Web.ProjectTemplates/content/EmptyWeb-CSharp/Program.cs#L20-L22>

ASP.NET Core 3.0:

<https://github.com/dotnet/aspnetcore/blob/b1ca2c1155da3920f0df5108b9fedbe82efaa11c/src/ProjectTemplates/src/Web.ProjectTemplates/content/EmptyWeb-CSharp/Program.cs#L19-L24>

`Host` uses one dependency injection (DI) container to build the app. `WebHost` uses two containers: one for the host and one for the app. As a result, the `Startup` constructor no longer supports custom service injection. Only `IHostEnvironment`, `IWebHostEnvironment`, and `IConfiguration` can be injected. This change prevents DI issues such as the duplicate creation of a singleton service.

Version introduced

3.0

Reason for change

This change is a consequence of replatforming the web stack onto the generic host library.

Recommended action

Inject services into the `Startup.Configure` method signature. For example:

```
public void Configure(IApplicationBuilder app, IOptions<MyOptions> options)
```

Category

ASP.NET Core

Affected APIs

None

Hosting: HTTPS redirection enabled for IIS out-of-process apps

Version 13.0.19218.0 of the [ASP.NET Core Module \(ANCM\)](#) for hosting via IIS out-of-process enables an existing HTTPS redirection feature for ASP.NET Core 3.0 and 2.2 apps.

For discussion, see [dotnet/AspNetCore#15243](#).

Version introduced

3.0

Old behavior

The ASP.NET Core 2.1 project template first introduced support for HTTPS middleware methods like `UseHttpsRedirection` and `UseHsts`. Enabling HTTPS redirection required the addition of configuration, since apps in development don't use the default port of 443. [HTTP Strict Transport Security \(HSTS\)](#) is active only if the request is already using HTTPS. Localhost is skipped by default.

New behavior

In ASP.NET Core 3.0, the IIS HTTPS scenario was [enhanced](#). With the enhancement, an app could discover the server's HTTPS ports and make `UseHttpsRedirection` work by default. The in-process component accomplished port discovery with the `I Server Addresses` feature, which only affects ASP.NET Core 3.0 apps because the in-process library is versioned with the framework. The out-of-process component changed to automatically add the `ASPNETCORE_HTTPS_PORT` environment variable. This change affected both ASP.NET Core 2.2 and 3.0 apps because the out-of-process component is shared globally. ASP.NET Core 2.1 apps aren't affected because they use a prior version of ANCM by default.

The preceding behavior was modified in ASP.NET Core 3.0.1 and 3.1.0 Preview 3 to reverse the behavior changes in ASP.NET Core 2.x. These changes only affect IIS out-of-process apps.

As detailed above, installing ASP.NET Core 3.0.0 had the side effect of also activating the `UseHttpsRedirection` middleware in ASP.NET Core 2.x apps. A change was made to ANCM in ASP.NET Core 3.0.1 and 3.1.0 Preview 3 such that installing them no longer has this effect on ASP.NET Core 2.x apps. The `ASPNETCORE_HTTPS_PORT` environment variable that ANCM populated in ASP.NET Core 3.0.0 was changed to `ASPNETCORE_ANCM_HTTPS_PORT` in ASP.NET Core 3.0.1 and 3.1.0 Preview 3. `UseHttpsRedirection` was also updated in these releases to understand both the new and old variables. ASP.NET Core 2.x won't be updated. As a result, it reverts to the previous behavior of being disabled by default.

Reason for change

Improved ASP.NET Core 3.0 functionality.

Recommended action

No action is required if you want all clients to use HTTPS. To allow some clients to use HTTP, take one of the following steps:

- Remove the calls to `UseHttpsRedirection` and `UseHsts` from your project's `Startup.Configure` method, and redeploy the app.
- In your `web.config` file, set the `ASPNETCORE_HTTPS_PORT` environment variable to an empty string. This change can occur directly on the server without redeploying the app. For example:

```
<aspNetCore processPath="dotnet" arguments=".\\WebApplication3.dll" stdoutLogEnabled="false"
stdoutLogFile="\\?\%home%\LogFiles\stdout" >
<environmentVariables>
<environmentVariable name="ASPNETCORE_HTTPS_PORT" value="" />
</environmentVariables>
</aspNetCore>
```

`UseHttpsRedirection` can still be:

- Activated manually in ASP.NET Core 2.x by setting the `ASPNETCORE_HTTPS_PORT` environment variable to the appropriate port number (443 in most production scenarios).
- Deactivated in ASP.NET Core 3.x by defining `ASPNETCORE_ANCM_HTTPS_PORT` with an empty string value. This value is set in the same fashion as the preceding `ASPNETCORE_HTTPS_PORT` example.

Machines running ASP.NET Core 3.0.0 apps should install the ASP.NET Core 3.0.1 runtime before installing the ASP.NET Core 3.1.0 Preview 3 ANCM. Doing so ensures that `UseHttpsRedirection` continues to operate as expected for the ASP.NET Core 3.0 apps.

In Azure App Service, ANCM deploys on a separate schedule from the runtime because of its global nature. ANCM was deployed to Azure with these changes after ASP.NET Core 3.0.1 and 3.1.0 were deployed.

Category

ASP.NET Core

Affected APIs

[HttpsPolicyBuilderExtensions.UseHttpsRedirection\(IApplicationBuilder\)](#)

Hosting: `IHostingEnvironment` and `IApplicationLifetime` types marked obsolete and replaced

New types have been introduced to replace existing `IHostingEnvironment` and `IApplicationLifetime` types.

Version introduced

3.0

Old behavior

There were two different `IHostingEnvironment` and `IApplicationLifetime` types from `Microsoft.Extensions.Hosting` and `Microsoft.AspNetCore.Hosting`.

New behavior

The old types have been marked as obsolete and replaced with new types.

Reason for change

When `Microsoft.Extensions.Hosting` was introduced in ASP.NET Core 2.1, some types like `IHostingEnvironment` and `IApplicationLifetime` were copied from `Microsoft.AspNetCore.Hosting`. Some ASP.NET Core 3.0 changes cause apps to include both the `Microsoft.Extensions.Hosting` and `Microsoft.AspNetCore.Hosting` namespaces. Any use of those duplicate types causes an "ambiguous reference" compiler error when both namespaces are referenced.

Recommended action

Replaced any usages of the old types with the newly introduced types as below:

Obsolete types (warning):

- `Microsoft.Extensions.Hosting.IHostingEnvironment`
- `Microsoft.AspNetCore.Hosting.IHostingEnvironment`
- `Microsoft.Extensions.Hosting.IApplicationLifetime`
- `Microsoft.AspNetCore.Hosting.IApplicationLifetime`
- `Microsoft.Extensions.Hosting.EnvironmentName`
- `Microsoft.AspNetCore.Hosting.EnvironmentName`

New types:

- `Microsoft.Extensions.Hosting.IHostEnvironment`
- `Microsoft.AspNetCore.Hosting.IWebHostEnvironment : IHostEnvironment`
- `Microsoft.Extensions.Hosting.IHostApplicationLifetime`
- `Microsoft.Extensions.Hosting.Environments`

The new `IHostEnvironment`, `IsDevelopment` and `IsProduction` extension methods are in the `Microsoft.Extensions.Hosting` namespace. That namespace may need to be added to your project.

Category

ASP.NET Core

Affected APIs

- `Microsoft.AspNetCore.Hosting.EnvironmentName`
- `Microsoft.AspNetCore.Hosting.IApplicationLifetime`
- `Microsoft.AspNetCore.Hosting.IHostingEnvironment`

- Microsoft.Extensions.Hosting.EnvironmentName
 - Microsoft.Extensions.Hosting.IApplicationLifetime
 - Microsoft.Extensions.Hosting.IHostingEnvironment
-

Hosting: ObjectPoolProvider removed from WebHostBuilder dependencies

As part of making ASP.NET Core more pay for play, the `ObjectPoolProvider` was removed from the main set of dependencies. Specific components relying on `ObjectPoolProvider` now add it themselves.

For discussion, see [dotnet/aspnetcore#5944](#).

Version introduced

3.0

Old behavior

`WebHostBuilder` provides `ObjectPoolProvider` by default in the DI container.

New behavior

`WebHostBuilder` no longer provides `ObjectPoolProvider` by default in the DI container.

Reason for change

This change was made to make ASP.NET Core more pay for play.

Recommended action

If your component requires `ObjectPoolProvider`, it needs to be added to your dependencies via the `IServiceCollection`.

Category

ASP.NET Core

Affected APIs

None

HTTP: DefaultHttpContext extensibility removed

As part of ASP.NET Core 3.0 performance improvements, the extensibility of `DefaultHttpContext` was removed. The class is now `sealed`. For more information, see [dotnet/aspnetcore#6504](#).

If your unit tests use `Mock<DefaultHttpContext>`, use `Mock<HttpContext>` or `new DefaultHttpContext()` instead.

For discussion, see [dotnet/aspnetcore#6534](#).

Version introduced

3.0

Old behavior

Classes can derive from `DefaultHttpContext`.

New behavior

Classes can't derive from `DefaultHttpContext`.

Reason for change

The extensibility was provided initially to allow pooling of the `HttpContext`, but it introduced unnecessary complexity and impeded other optimizations.

Recommended action

If you're using `Mock<DefaultHttpContext>` in your unit tests, begin using `Mock<HttpContext>` instead.

Category

ASP.NET Core

Affected APIs

[Microsoft.AspNetCore.Http.DefaultHttpContext](#)

HTTP: HeaderNames constants changed to static readonly

Starting in ASP.NET Core 3.0 Preview 5, the fields in [Microsoft.Net.Http.Headers.HeaderNames](#) changed from `const` to `static readonly`.

For discussion, see [dotnet/aspnetcore#9514](#).

Version introduced

3.0

Old behavior

These fields used to be `const`.

New behavior

These fields are now `static readonly`.

Reason for change

The change:

- Prevents the values from being embedded across assembly boundaries, allowing for value corrections as needed.
- Enables faster reference equality checks.

Recommended action

Recompile against 3.0. Source code using these fields in the following ways can no longer do so:

- As an attribute argument
- As a `case` in a `switch` statement
- When defining another `const`

To work around the breaking change, switch to using self-defined header name constants or string literals.

Category

ASP.NET Core

Affected APIs

[Microsoft.Net.Http.Headers.HeaderNames](#)

HTTP: Response body infrastructure changes

The infrastructure backing an HTTP response body has changed. If you're using `HttpResponse` directly, you shouldn't need to make any code changes. Read further if you're wrapping or replacing `HttpResponse.Body` or accessing `HttpContext.Features`.

Version introduced

3.0

Old behavior

There were three APIs associated with the HTTP response body:

- `IHttpResponseFeature.Body`
- `IHttpSendFileFeature.SendFileAsync`
- `IHttpBufferingFeature.DisableResponseBuffering`

New behavior

If you replace `HttpResponse.Body`, it replaces the entire `IHttpResponseBodyFeature` with a wrapper around your given stream using `StreamResponseBodyFeature` to provide default implementations for all of the expected APIs. Setting back the original stream reverts this change.

Reason for change

The motivation is to combine the response body APIs into a single new feature interface.

Recommended action

Use `IHttpResponseBodyFeature` where you previously were using `IHttpResponseFeature.Body`, `IHttpSendFileFeature`, or `IHttpBufferingFeature`.

Category

ASP.NET Core

Affected APIs

- [Microsoft.AspNetCore.Http.Features.IHttpBufferingFeature](#)
- [Microsoft.AspNetCore.Http.Features.IHttpResponseFeature.Body](#)
- [Microsoft.AspNetCore.Http.Features.IHttpSendFileFeature](#)

HTTP: Some cookie SameSite defaults changed to None

`SameSite` is an option for cookies that can help mitigate some Cross-Site Request Forgery (CSRF) attacks. When this option was initially introduced, inconsistent defaults were used across various ASP.NET Core APIs. The inconsistency has led to confusing results. As of ASP.NET Core 3.0, these defaults are better aligned. You must opt in to this feature on a per-component basis.

Version introduced

3.0

Old behavior

Similar ASP.NET Core APIs used different default `SameSiteMode` values. An example of the inconsistency is seen in `HttpResponse.Cookies.Append(String, String)` and `HttpResponse.Cookies.Append(String, String, CookieOptions)`, which defaulted to `SameSiteMode.None` and `SameSiteMode.Lax`, respectively.

New behavior

All the affected APIs default to `SameSiteMode.None`.

Reason for change

The default value was changed to make `SameSite` an opt-in feature.

Recommended action

Each component that emits cookies needs to decide if `SameSite` is appropriate for its scenarios. Review your

usage of the affected APIs and reconfigure `SameSite` as needed.

Category

ASP.NET Core

Affected APIs

- `IResponseCookies.Append(String, String, CookieOptions)`
- `CookiePolicyOptions.MinimumSameSitePolicy`

HTTP: Synchronous IO disabled in all servers

Starting with ASP.NET Core 3.0, synchronous server operations are disabled by default.

Change description

`AllowSynchronousIO` is an option in each server that enables or disables synchronous IO APIs like `HttpRequest.Body.Read`, `HttpResponse.Body.Write`, and `Stream.Flush`. These APIs have long been a source of thread starvation and app hangs. Starting in ASP.NET Core 3.0 Preview 3, these synchronous operations are disabled by default.

Affected servers:

- Kestrel
- HttpSys
- IIS in-process
- TestServer

Expect errors similar to:

- Synchronous operations are disallowed. Call `ReadAsync` or set `AllowSynchronousIO` to true instead.
- Synchronous operations are disallowed. Call `WriteAsync` or set `AllowSynchronousIO` to true instead.
- Synchronous operations are disallowed. Call `FlushAsync` or set `AllowSynchronousIO` to true instead.

Each server has an `AllowSynchronousIO` option that controls this behavior and the default for all of them is now `false`.

The behavior can also be overridden on a per-request basis as a temporary mitigation. For example:

```
var syncIOFeature = HttpContext.Features.Get< IHttpBodyControlFeature>();  
if (syncIOFeature != null)  
{  
    syncIOFeature.AllowSynchronousIO = true;  
}
```

If you have trouble with a `TextWriter` or another stream calling a synchronous API in `Dispose`, call the new `DisposeAsync` API instead.

For discussion, see [dotnet/aspnetcore#7644](#).

Version introduced

3.0

Old behavior

`HttpRequest.Body.Read`, `HttpResponse.Body.Write`, and `Stream.Flush` were allowed by default.

New behavior

These synchronous APIs are disallowed by default:

Expect errors similar to:

- Synchronous operations are disallowed. Call `ReadAsync` or set `AllowSynchronousIO` to true instead.
- Synchronous operations are disallowed. Call `WriteAsync` or set `AllowSynchronousIO` to true instead.
- Synchronous operations are disallowed. Call `FlushAsync` or set `AllowSynchronousIO` to true instead.

Reason for change

These synchronous APIs have long been a source of thread starvation and app hangs. Starting in ASP.NET Core 3.0 Preview 3, the synchronous operations are disabled by default.

Recommended action

Use the asynchronous versions of the methods. The behavior can also be overridden on a per-request basis as a temporary mitigation.

```
var syncIOFeature = HttpContext.Features.Get< IHttpBodyControlFeature>();  
if (syncIOFeature != null)  
{  
    syncIOFeature.AllowSynchronousIO = true;  
}
```

Category

ASP.NET Core

Affected APIs

- `Stream.Flush`
- `Stream.Read`
- `Stream.Write`

Identity: AddDefaultUI method overload removed

Starting with ASP.NET Core 3.0, the `IdentityBuilderUIExtensions.AddDefaultUI(IdentityBuilder, UIFramework)` method overload no longer exists.

Version introduced

3.0

Reason for change

This change was a result of adoption of the static web assets feature.

Recommended action

Call `IdentityBuilderUIExtensions.AddDefaultUI(IdentityBuilder)` instead of the overload that takes two arguments. If you're using Bootstrap 3, also add the following line to a `<PropertyGroup>` element in your project file:

```
<IdentityUIFrameworkVersion>Bootstrap3</IdentityUIFrameworkVersion>
```

Category

ASP.NET Core

Affected APIs

`IdentityBuilderUIExtensions.AddDefaultUI(IdentityBuilder, UIFramework)`

Identity: Default Bootstrap version of UI changed

Starting in ASP.NET Core 3.0, Identity UI defaults to using version 4 of Bootstrap.

Version introduced

3.0

Old behavior

The `services.AddDefaultIdentity<IdentityUser>().AddDefaultUI();` method call was the same as
`services.AddDefaultIdentity<IdentityUser>().AddDefaultUI(UIFramework.Bootstrap3);`

New behavior

The `services.AddDefaultIdentity<IdentityUser>().AddDefaultUI();` method call is the same as
`services.AddDefaultIdentity<IdentityUser>().AddDefaultUI(UIFramework.Bootstrap4);`

Reason for change

Bootstrap 4 was released during ASP.NET Core 3.0 timeframe.

Recommended action

You're impacted by this change if you use the default Identity UI and have added it in `Startup.ConfigureServices` as shown in the following example:

```
services.AddDefaultIdentity<IdentityUser>().AddDefaultUI();
```

Take one of the following actions:

- Migrate your app to use Bootstrap 4 using their [migration guide](#).
- Update `Startup.ConfigureServices` to enforce usage of Bootstrap 3. For example:

```
services.AddDefaultIdentity<IdentityUser>().AddDefaultUI(UIFramework.Bootstrap3);
```

Category

ASP.NET Core

Affected APIs

None

Identity: SignInAsync throws exception for unauthenticated identity

By default, `SignInAsync` throws an exception for principals / identities in which `IsAuthenticated` is `false`.

Version introduced

3.0

Old behavior

`SignInAsync` accepts any principals / identities, including identities in which `IsAuthenticated` is `false`.

New behavior

By default, `SignInAsync` throws an exception for principals / identities in which `IsAuthenticated` is `false`.

There's a new flag to suppress this behavior, but the default behavior has changed.

Reason for change

The old behavior was problematic because, by default, these principals were rejected by `[Authorize]` / `RequireAuthenticatedUser()`.

Recommended action

In ASP.NET Core 3.0 Preview 6, there's a `RequireAuthenticatedSignIn` flag on `AuthenticationOptions` that is `true` by default. Set this flag to `false` to restore the old behavior.

Category

ASP.NET Core

Affected APIs

None

Identity: SignInManager constructor accepts new parameter

Starting with ASP.NET Core 3.0, a new `IUserConfirmation<TUser>` parameter was added to the `SignInManager` constructor. For more information, see [dotnet/aspnetcore#8356](#).

Version introduced

3.0

Reason for change

The motivation for the change was to add support for new email / confirmation flows in Identity.

Recommended action

If manually constructing a `SignInManager`, provide an implementation of `IUserConfirmation` or grab one from dependency injection to provide.

Category

ASP.NET Core

Affected APIs

`SignInManager<TUser>`

Identity: UI uses static web assets feature

ASP.NET Core 3.0 introduced a static web assets feature, and Identity UI has adopted it.

Change description

As a result of Identity UI adopting the static web assets feature:

- Framework selection is accomplished by using the `IdentityUIFrameworkVersion` property in your project file.
- Bootstrap 4 is the default UI framework for Identity UI. Bootstrap 3 has reached end of life, and you should consider migrating to a supported version.

Version introduced

3.0

Old behavior

The default UI framework for Identity UI was **Bootstrap 3**. The UI framework could be configured using a parameter to the `AddDefaultUI` method call in `Startup.ConfigureServices`.

New behavior

The default UI framework for Identity UI is **Bootstrap 4**. The UI framework must be configured in your project file, instead of in the `AddDefaultUI` method call.

Reason for change

Adoption of the static web assets feature required that the UI framework configuration move to MSBuild. The decision on which framework to embed is a build-time decision, not a runtime decision.

Recommended action

Review your site UI to ensure the new Bootstrap 4 components are compatible. If necessary, use the `IdentityUIFrameworkVersion` MSBuild property to revert to Bootstrap 3. Add the property to a `<PropertyGroup>` element in your project file:

```
<IdentityUIFrameworkVersion>Bootstrap3</IdentityUIFrameworkVersion>
```

Category

ASP.NET Core

Affected APIs

`IdentityBuilderUIExtensions.AddDefaultUI(IdentityBuilder, UIFramework)`

Kestrel: Connection adapters removed

As part of the move to move "pubternal" APIs to `public`, the concept of an `IConnectionAdapter` was removed from Kestrel. Connection adapters are being replaced with connection middleware (similar to HTTP middleware in the ASP.NET Core pipeline, but for lower-level connections). HTTPS and connection logging have moved from connection adapters to connection middleware. Those extension methods should continue to work seamlessly,

but the implementation details have changed.

For more information, see [dotnet/aspnetcore#11412](#). For discussion, see [dotnet/aspnetcore#11475](#).

Version introduced

3.0

Old behavior

Kestrel extensibility components were created using `IConnectionAdapter`.

New behavior

Kestrel extensibility components are created as [middleware](#).

Reason for change

This change is intended to provide a more flexible extensibility architecture.

Recommended action

Convert any implementations of `IConnectionAdapter` to use the new middleware pattern as shown [here](#).

Category

ASP.NET Core

Affected APIs

`Microsoft.AspNetCore.Server.Kestrel.Core.Adapter.Internal.IConnectionAdapter`

Kestrel: Empty HTTPS assembly removed

The assembly [Microsoft.AspNetCore.Server.Kestrel.Https](#) has been removed.

Version introduced

3.0

Reason for change

In ASP.NET Core 2.1, the contents of `Microsoft.AspNetCore.Server.Kestrel.Https` were moved to `Microsoft.AspNetCore.Server.Kestrel.Core`. This change was done in a non-breaking way using `[TypeForwardedTo]` attributes.

Recommended action

- Libraries referencing `Microsoft.AspNetCore.Server.Kestrel.Https` 2.0 should update all ASP.NET Core dependencies to 2.1 or later. Otherwise, they may break when loaded into an ASP.NET Core 3.0 app.
- Apps and libraries targeting ASP.NET Core 2.1 and later should remove any direct references to the `Microsoft.AspNetCore.Server.Kestrel.Https` NuGet package.

Category

ASP.NET Core

Affected APIs

None

Kestrel: Request trailer headers moved to new collection

In prior versions, Kestrel added HTTP/1.1 chunked trailer headers into the request headers collection when the request body was read to the end. This behavior caused concerns about ambiguity between headers and trailers. The decision was made to move the trailers to a new collection.

HTTP/2 request trailers were unavailable in ASP.NET Core 2.2 but are now also available in this new collection in ASP.NET Core 3.0.

New request extension methods have been added to access these trailers.

HTTP/1.1 trailers are available once the entire request body has been read.

HTTP/2 trailers are available once they're received from the client. The client won't send the trailers until the entire request body has been at least buffered by the server. You may need to read the request body to free up buffer space. Trailers are always available if you read the request body to the end. The trailers mark the end of the body.

Version introduced

3.0

Old behavior

Request trailer headers would be added to the `HttpRequest.Headers` collection.

New behavior

Request trailer headers aren't present in the `HttpRequest.Headers` collection. Use the following extension methods on `HttpRequest` to access them:

- `GetDeclaredTrailers()` - Gets the request "Trailer" header that lists which trailers to expect after the body.
- `SupportsTrailers()` - Indicates if the request supports receiving trailer headers.
- `CheckTrailersAvailable()` - Determines if the request supports trailers and if they're available for reading.
- `GetTrailer(string trailerName)` - Gets the requested trailing header from the response.

Reason for change

Trailers are a key feature in scenarios like gRPC. Merging the trailers in to request headers was confusing to users.

Recommended action

Use the trailer-related extension methods on `HttpRequest` to access trailers.

Category

ASP.NET Core

Affected APIs

[HttpRequest.Headers](#)

Kestrel: Transport abstractions removed and made public

As part of moving away from "pubternal" APIs, the Kestrel transport layer APIs are exposed as a public interface in the `Microsoft.AspNetCore.Connections.Abstractions` library.

Version introduced

3.0

Old behavior

- Transport-related abstractions were available in the `Microsoft.AspNetCore.Server.Kestrel.Transport.Abstractions` library.
- The `ListenOptions.NoDelay` property was available.

New behavior

- The `IConnectionListener` interface was introduced in the `Microsoft.AspNetCore.Connections.Abstractions` library to expose the most used functionality from the `...Transport.Abstractions` library.
- The `NoDelay` is now available in transport options (`LibuvTransportOptions` and `SocketTransportOptions`).
- `SchedulingMode` is no longer available.

Reason for change

ASP.NET Core 3.0 has moved away from "pubternal" APIs.

Recommended action

Category

ASP.NET Core

Affected APIs

None

Localization: ResourceManagerWithCultureStringLocalizer and WithCulture marked obsolete

The `ResourceManagerWithCultureStringLocalizer` class and `WithCulture` interface member are often sources of confusion for users of localization, especially when creating their own `IStringLocalizer` implementation. These items give the user the impression that an `IStringLocalizer` instance is "per-language, per-resource". In reality, the instances should only be "per-resource". The language searched for is determined by the `CultureInfo.CurrentCulture` at execution time. To eliminate the source of confusion, the APIs were marked as obsolete in ASP.NET Core 3.0 Preview 3. The APIs will be removed in a future release.

For context, see [dotnet/aspnetcore#3324](#). For discussion, see [dotnet/aspnetcore#7756](#).

Version introduced

3.0

Old behavior

Methods weren't marked as `Obsolete`.

New behavior

Methods are marked `Obsolete`.

Reason for change

The APIs represented a use case that isn't recommended. There was confusion about the design of localization.

Recommended action

The recommendation is to use `ResourceManagerStringLocalizer` instead. Let the culture be set by the `CurrentCulture`. If that isn't an option, create and use a copy of `ResourceManagerWithCultureStringLocalizer`.

Category

ASP.NET Core

Affected APIs

- `ResourceManagerWithCultureStringLocalizer`
 - `ResourceManagerStringLocalizer.WithCulture`
-

Logging: DebugLogger class made internal

Prior to ASP.NET Core 3.0, `DebugLogger`'s access modifier was `public`. In ASP.NET Core 3.0, the access modifier changed to `internal`.

Version introduced

3.0

Reason for change

The change is being made to:

- Enforce consistency with other logger implementations such as `ConsoleLogger`.
- Reduce the API surface.

Recommended action

Use the `AddDebug` `ILoggingBuilder` extension method to enable debug logging. `DebugLoggerProvider` is also still `public` in the event the service needs to be registered manually.

Category

ASP.NET Core

Affected APIs`Microsoft.Extensions.Logging.Debug.DebugLogger`**MVC: Async suffix trimmed from controller action names**

As part of addressing [dotnet/aspnetcore#4849](#), ASP.NET Core MVC trims the suffix `Async` from action names by default. Starting with ASP.NET Core 3.0, this change affects both routing and link generation.

Version introduced

3.0

Old behavior

Consider the following ASP.NET Core MVC controller:

```
public class ProductController : Controller
{
    public async IActionResult ListAsync()
    {
        var model = await DbContext.Products.ToListAsync();
        return View(model);
    }
}
```

The action is routable via `Product/ListAsync`. Link generation requires specifying the `Async` suffix. For example:

```
<a asp-controller="Product" asp-action="ListAsync">List</a>
```

New behavior

In ASP.NET Core 3.0, the action is routable via `Product>List`. Link generation code should omit the `Async` suffix. For example:

```
<a asp-controller="Product" asp-action="List">List</a>
```

This change doesn't affect names specified using the `[ActionName]` attribute. The new behavior can be disabled by setting `MvcOptions.SuppressAsyncSuffixInActionNames` to `false` in `Startup.ConfigureServices`:

```
services.AddMvc(options =>
{
    options.SuppressAsyncSuffixInActionNames = false;
});
```

Reason for change

By convention, asynchronous .NET methods are suffixed with `Async`. However, when a method defines an MVC action, it's undesirable to use the `Async` suffix.

Recommended action

If your app depends on MVC actions preserving the name's `Async` suffix, choose one of the following mitigations:

- Use the `[ActionName]` attribute to preserve the original name.
- Disable the renaming entirely by setting `MvcOptions.SuppressAsyncSuffixInActionNames` to `false` in `Startup.ConfigureServices`:

```
services.AddMvc(options =>
{
    options.SuppressAsyncSuffixInActionNames = false;
});
```

Category

ASP.NET Core

Affected APIs

None

MVC: JsonResult moved to Microsoft.AspNetCore.Mvc.Core

`JsonResult` has moved to the `Microsoft.AspNetCore.Mvc.Core` assembly. This type used to be defined in `Microsoft.AspNetCore.Mvc.Formatters.Json`. An assembly-level `[TypeForwardedTo]` attribute was added to `Microsoft.AspNetCore.Mvc.Formatters.Json` to address this issue for the majority of users. Apps that use third-party libraries may encounter issues.

Version introduced

3.0 Preview 6

Old behavior

An app using a 2.2-based library builds successfully.

New behavior

An app using a 2.2-based library fails compilation. An error containing a variation of the following text is provided:

```
The type 'JsonResult' exists in both 'Microsoft.AspNetCore.Mvc.Core, Version=3.0.0.0, Culture=neutral, PublicKeyToken=adb9793829ddae60' and 'Microsoft.AspNetCore.Mvc.Formatters.Json, Version=2.0.0.0, Culture=neutral, PublicKeyToken=adb9793829ddae60'
```

For an example of such an issue, see [dotnet/aspnetcore#7220](#).

Reason for change

Platform-level changes to the composition of ASP.NET Core as described at [aspnet/Announcements#325](#).

Recommended action

Libraries compiled against the 2.2 version of `Microsoft.AspNetCore.Mvc.Formatters.Json` may need to recompile to address the problem for all consumers. If affected, contact the library author. Request recompilation of the library to target ASP.NET Core 3.0.

Category

ASP.NET Core

Affected APIs

[Microsoft.AspNetCore.Mvc.JsonResult](#)

MVC: Precompilation tool deprecated

In ASP.NET Core 1.1, the `Microsoft.AspNetCore.Mvc.Razor.ViewCompilation` (MVC precompilation tool) package was introduced to add support for publish-time compilation of Razor files (`.cshtml` files). In ASP.NET Core 2.1, the [Razor SDK](#) was introduced to expand upon features of the precompilation tool. The Razor SDK added support for build- and publish-time compilation of Razor files. The SDK verifies the correctness of `.cshtml` files at build time while improving on app startup time. The Razor SDK is on by default, and no gesture is required to start using it.

In ASP.NET Core 3.0, the ASP.NET Core 1.1-era MVC precompilation tool was removed. Earlier package versions will continue receiving important bug and security fixes in the patch release.

Version introduced

3.0

Old behavior

The `Microsoft.AspNetCore.Mvc.Razor.ViewCompilation` package was used to pre-compile MVC Razor views.

New behavior

The Razor SDK natively supports this functionality. The `Microsoft.AspNetCore.Mvc.Razor.ViewCompilation` package is no longer updated.

Reason for change

The Razor SDK provides more functionality and verifies the correctness of `.cshtml` files at build time. The SDK also improves app startup time.

Recommended action

For users of ASP.NET Core 2.1 or later, update to use the native support for precompilation in the [Razor SDK](#). If bugs or missing features prevent migration to the Razor SDK, open an issue at [dotnet/aspnetcore](#).

Category

ASP.NET Core

Affected APIs

None

MVC: "Pubternal" types changed to internal

In ASP.NET Core 3.0, all "pubternal" types in MVC were updated to either be `public` in a supported namespace or `internal` as appropriate.

Change description

In ASP.NET Core, "pubternal" types are declared as `public` but reside in a `.Internal`-suffixed namespace. While these types are `public`, they have no support policy and are subject to breaking changes. Unfortunately, accidental use of these types has been common, resulting in breaking changes to these projects and limiting the ability to maintain the framework.

Version introduced

3.0

Old behavior

Some types in MVC were `public` but in a `.Internal` namespace. These types had no support policy and were subject to breaking changes.

New behavior

All such types are updated either to be `public` in a supported namespace or marked as `internal`.

Reason for change

Accidental use of the "pubternal" types has been common, resulting in breaking changes to these projects and limiting the ability to maintain the framework.

Recommended action

If you're using types that have become truly `public` and have been moved into a new, supported namespace, update your references to match the new namespaces.

If you're using types that have become marked as `internal`, you'll need to find an alternative. The previously "pubternal" types were never supported for public use. If there are specific types in these namespaces that are critical to your apps, file an issue at [dotnet/aspnetcore](#). Considerations may be made for making the requested types `public`.

Category

ASP.NET Core

Affected APIs

This change includes types in the following namespaces:

- `Microsoft.AspNetCore.Mvc.Cors.Internal`
- `Microsoft.AspNetCore.Mvc.DataAnnotations.Internal`
- `Microsoft.AspNetCore.Mvc.Formatters.Internal`
- `Microsoft.AspNetCore.Mvc.Formatters.Json.Internal`
- `Microsoft.AspNetCore.Mvc.Formatters.Xml.Internal`
- `Microsoft.AspNetCore.Mvc.Internal`
- `Microsoft.AspNetCore.Mvc.ModelBinding.Internal`
- `Microsoft.AspNetCore.Mvc.Razor.Internal`
- `Microsoft.AspNetCore.Mvc.RazorPages.Internal`
- `Microsoft.AspNetCore.Mvc.TagHelpers.Internal`
- `Microsoft.AspNetCore.Mvc.ViewFeatures.Internal`

MVC: Web API compatibility shim removed

Starting with ASP.NET Core 3.0, the `Microsoft.AspNetCore.Mvc.WebApiCompatShim` package is no longer available.

Change description

The `Microsoft.AspNetCore.Mvc.WebApiCompatShim` (WebApiCompatShim) package provides partial compatibility in ASP.NET Core with ASP.NET 4.x Web API 2 to simplify migrating existing Web API implementations to ASP.NET Core. However, apps using the WebApiCompatShim don't benefit from the API-related features shipping in recent ASP.NET Core releases. Such features include improved Open API specification generation, standardized error handling, and client code generation. To better focus the API efforts in 3.0, WebApiCompatShim was removed. Existing apps using the WebApiCompatShim should migrate to the newer `[ApiController]` model.

Version introduced

3.0

Reason for change

The Web API compatibility shim was a migration tool. It restricts user access to new functionality added in ASP.NET Core.

Recommended action

Remove usage of this shim and migrate directly to the similar functionality in ASP.NET Core itself.

Category

ASP.NET Core

Affected APIs

[Microsoft.AspNetCore.Mvc.WebApiCompatShim](#)

Razor: RazorTemplateEngine API removed

The `RazorTemplateEngine` API was removed and replaced with

```
Microsoft.AspNetCore.Razor.Language.RazorProjectEngine .
```

For discussion, see GitHub issue [dotnet/aspnetcore#25215](#).

Version introduced

3.0

Old behavior

A template engine can be created and used to parse and generate code for Razor files.

New behavior

`RazorProjectEngine` can be created and provided the same type of information as `RazorTemplateEngine` to parse and generate code for Razor files. `RazorProjectEngine` also provides extra levels of configuration.

Reason for change

`RazorTemplateEngine` was too tightly coupled to the existing implementations. This tight coupling led to more questions when trying to properly configure a Razor parsing/generation pipeline.

Recommended action

Use `RazorProjectEngine` instead of `RazorTemplateEngine`. Consider the following examples.

`Create and configure the RazorProjectEngine`

```
RazorProjectEngine projectEngine =
    RazorProjectEngine.Create(RazorConfiguration.Default,
        RazorProjectFileSystem.Create(@"C:\source\repos\ConsoleApp4\ConsoleApp4"),
        builder =>
    {
        builder.ConfigureClass((document, classNode) =>
        {
            classNode.ClassName = "MyClassName";

            // Can also configure other aspects of the class here.
        });

        // More configuration can go here
    });
}
```

`Generate code for a Razor file`

```
RazorProjectItem item = projectEngine.FileSystem.GetItem(
    @"C:\source\repos\ConsoleApp4\ConsoleApp4\Example.cshtml",
    FileKinds.Legacy);
RazorCodeDocument output = projectEngine.Process(item);

// Things available
RazorSyntaxTree syntaxTree = output.GetSyntaxTree();
DocumentIntermediateNode intermediateDocument =
    output.GetDocumentIntermediateNode();
RazorCSharpDocument csharpDocument = output.GetCSharpDocument();
```

Category

ASP.NET Core

Affected APIs

- `RazorTemplateEngine`
- `RazorTemplateEngineOptions`

Razor: Runtime compilation moved to a package

Support for runtime compilation of Razor views and Razor Pages has moved to a separate package.

Version introduced

3.0

Old behavior

Runtime compilation is available without needing additional packages.

New behavior

The functionality has been moved to the `Microsoft.AspNetCore.Mvc.Razor.RuntimeCompilation` package.

The following APIs were previously available in `Microsoft.AspNetCore.Mvc.Razor.RazorViewEngineOptions` to support runtime compilation. The APIs are now available via

`Microsoft.AspNetCore.Mvc.Razor.RuntimeCompilation.MvcRazorRuntimeCompilationOptions`.

- `RazorViewEngineOptions.FileProviders` is now `MvcRazorRuntimeCompilationOptions.FileProviders`
- `RazorViewEngineOptions.AdditionalCompilationReferences` is now `MvcRazorRuntimeCompilationOptions.AdditionalReferencePaths`

In addition, `Microsoft.AspNetCore.Mvc.Razor.RazorViewEngineOptions.AllowRecompilingViewsOnFileChange` has been removed. Recompilation on file changes is enabled by default by referencing the `Microsoft.AspNetCore.Mvc.Razor.RuntimeCompilation` package.

Reason for change

This change was necessary to remove the ASP.NET Core shared framework dependency on Roslyn.

Recommended action

Apps that require runtime compilation or recompilation of Razor files should take the following steps:

1. Add a reference to the `Microsoft.AspNetCore.Mvc.Razor.RuntimeCompilation` package.
2. Update the project's `Startup.ConfigureServices` method to include a call to `AddRazorRuntimeCompilation`.
For example:

```
services.AddMvc()
    .AddRazorRuntimeCompilation();
```

Category

ASP.NET Core

Affected APIs

`Microsoft.AspNetCore.Mvc.Razor.RazorViewEngineOptions`

Session state: Obsolete APIs removed

Obsolete APIs for configuring session cookies were removed. For more information, see [aspnet/Announcements#257](#).

Version introduced

3.0

Reason for change

This change enforces consistency across APIs for configuring features that use cookies.

Recommended action

Migrate usage of the removed APIs to their newer replacements. Consider the following example in

```
Startup.ConfigureServices :
```

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddSession(options =>
    {
        // Removed obsolete APIs
        options.CookieName = "SessionCookie";
        options.CookieDomain = "contoso.com";
        options.CookiePath = "/";
        options.CookieHttpOnly = true;
        options.CookieSecure = CookieSecurePolicy.Always;

        // new API
        options.Cookie.Name = "SessionCookie";
        options.Cookie.Domain = "contoso.com";
        options.Cookie.Path = "/";
        options.Cookie.HttpOnly = true;
        options.Cookie.SecurePolicy = CookieSecurePolicy.Always;
    });
}
```

Category

ASP.NET Core

Affected APIs

- `Microsoft.AspNetCore.Builder.SessionOptions.CookieDomain`
- `Microsoft.AspNetCore.Builder.SessionOptions.CookieHttpOnly`
- `Microsoft.AspNetCore.Builder.SessionOptions.CookieName`
- `Microsoft.AspNetCore.Builder.SessionOptions.CookiePath`
- `Microsoft.AspNetCore.Builder.SessionOptions.CookieSecure`

Shared framework: Assemblies removed from Microsoft.AspNetCore.App

Starting in ASP.NET Core 3.0, the ASP.NET Core shared framework (`Microsoft.AspNetCore.App`) only contains first-party assemblies that are fully developed, supported, and serviceable by Microsoft.

Change description

Think of the change as the redefining of boundaries for the ASP.NET Core "platform." The shared framework will be [source-buildable by anybody via GitHub](#) and will continue to offer the existing benefits of .NET Core shared frameworks to your apps. Some benefits include smaller deployment size, centralized patching, and faster startup time.

As part of the change, some notable breaking changes are introduced in `Microsoft.AspNetCore.App`.

Version introduced

3.0

Old behavior

Projects referenced `Microsoft.AspNetCore.App` via a `<PackageReference>` element in the project file.

Additionally, `Microsoft.AspNetCore.App` contained the following subcomponents:

- Json.NET (`Newtonsoft.Json`)
- Entity Framework Core (assemblies prefixed with `Microsoft.EntityFrameworkCore.`)
- Roslyn (`Microsoft.CodeAnalysis`)

New behavior

A reference to `Microsoft.AspNetCore.App` no longer requires a `<PackageReference>` element in the project file.

The .NET Core SDK supports a new element called `<FrameworkReference>`, which replaces the use of `<PackageReference>`.

For more information, see [dotnet/aspnetcore#3612](#).

Entity Framework Core ships as NuGet packages. This change aligns the shipping model with all other data access libraries on .NET. It provides Entity Framework Core the simplest path to continue innovating while supporting the various .NET platforms. The move of Entity Framework Core out of the shared framework has no impact on its status as a Microsoft-developed, supported, and serviceable library. The [.NET Core support policy](#) continues to cover it.

Json.NET and Entity Framework Core continue to work with ASP.NET Core. They won't, however, be included in the shared framework.

For more information, see [The future of JSON in .NET Core 3.0](#). Also see [the complete list of binaries](#) removed from the shared framework.

Reason for change

This change simplifies the consumption of `Microsoft.AspNetCore.App` and reduces the duplication between NuGet packages and shared frameworks.

For more information on the motivation for this change, see [this blog post](#).

Recommended action

Starting with ASP.NET Core 3.0, it is no longer necessary for projects to consume assemblies in `Microsoft.AspNetCore.App` as NuGet packages. To simplify the targeting and usage of the ASP.NET Core shared framework, many NuGet packages shipped since ASP.NET Core 1.0 are no longer produced. The APIs those packages provide are still available to apps by using a `<FrameworkReference>` to `Microsoft.AspNetCore.App`. Common API examples include Kestrel, MVC, and Razor.

This change doesn't apply to all binaries referenced via `Microsoft.AspNetCore.App` in ASP.NET Core 2.x. Notable exceptions include:

- `Microsoft.Extensions` libraries that continue to target .NET Standard are available as NuGet packages (see <https://github.com/dotnet/extensions>).
- APIs produced by the ASP.NET Core team that aren't part of `Microsoft.AspNetCore.App`. For example, the following components are available as NuGet packages:
 - Entity Framework Core
 - APIs that provide third-party integration
 - Experimental features
 - APIs with dependencies that couldn't fulfill the requirements to be in the shared framework
- Extensions to MVC that maintain support for Json.NET. An API is provided as a [NuGet package](#) to support using Json.NET and MVC. See the [ASP.NET Core migration guide for more details](#).
- The SignalR .NET client continues to support .NET Standard and ships as a [NuGet package](#). It's intended for use on many .NET runtimes, such as Xamarin and UWP.

For more information, see [Stop producing packages for shared framework assemblies in 3.0](#). For discussion, see [dotnet/aspnetcore#3757](#).

Category

ASP.NET Core

Affected APIs

- `Microsoft.CodeAnalysis`
- `Microsoft.EntityFrameworkCore`

Shared framework: Removed `Microsoft.AspNetCore.All`

Starting in ASP.NET Core 3.0, the `Microsoft.AspNetCore.All` metapackage and the matching `Microsoft.AspNetCore.All` shared framework are no longer produced. This package is available in ASP.NET Core 2.2 and will continue to receive servicing updates in ASP.NET Core 2.1.

Version introduced

3.0

Old behavior

Apps could use the `Microsoft.AspNetCore.All` metapackage to target the `Microsoft.AspNetCore.All` shared

framework on .NET Core.

New behavior

.NET Core 3.0 doesn't include a `Microsoft.AspNetCore.All` shared framework.

Reason for change

The `Microsoft.AspNetCore.All` metapackage included a large number of external dependencies.

Recommended action

Migrate your project to use the `Microsoft.AspNetCore.App` framework. Components that were previously available in `Microsoft.AspNetCore.All` are still available on NuGet. Those components are now deployed with your app instead of being included in the shared framework.

Category

ASP.NET Core

Affected APIs

None

SignalR: HandshakeProtocol.SuccessHandshakeData replaced

The `HandshakeProtocol.SuccessHandshakeData` field was removed and replaced with a helper method that generates a successful handshake response given a specific `IHubProtocol`.

Version introduced

3.0

Old behavior

`HandshakeProtocol.SuccessHandshakeData` was a `public static ReadOnlyMemory<byte>` field.

New behavior

`HandshakeProtocol.SuccessHandshakeData` has been replaced by a `static GetSuccessfulHandshake(IHubProtocol protocol)` method that returns a `ReadOnlyMemory<byte>` based on the specified protocol.

Reason for change

Additional fields were added to the handshake *response* that are non-constant and change depending on the selected protocol.

Recommended action

None. This type isn't designed for use from user code. It's `public`, so it can be shared between the SignalR server and client. It may also be used by customer SignalR clients written in .NET. **Users** of SignalR shouldn't be affected by this change.

Category

ASP.NET Core

Affected APIs

`HandshakeProtocol.SuccessHandshakeData`

SignalR: HubConnection ResetSendPing and ResetTimeout methods removed

The `ResetSendPing` and `ResetTimeout` methods were removed from the SignalR `HubConnection` API. These methods were originally intended only for internal use but were made public in ASP.NET Core 2.2. These methods won't be available starting in the ASP.NET Core 3.0 Preview 4 release. For discussion, see [dotnet/aspnetcore#8543](#).

Version introduced

3.0

Old behavior

APIs were available.

New behavior

APIs are removed.

Reason for change

These methods were originally intended only for internal use but were made public in ASP.NET Core 2.2.

Recommended action

Don't use these methods.

Category

ASP.NET Core

Affected APIs

- `HubConnection.ResetSendPing()`
 - `HubConnection.ResetTimeout()`
-

SignalR: HubConnectionContext constructors changed

SignalR's `HubConnectionContext` constructors changed to accept an options type, rather than multiple

parameters, to future-proof adding options. This change replaces two constructors with a single constructor that accepts an options type.

Version introduced

3.0

Old behavior

`HubConnectionContext` has two constructors:

```
public HubConnectionContext(ConnectionContext connectionContext, TimeSpan keepAliveInterval, ILoggerFactory loggerFactory);
public HubConnectionContext(ConnectionContext connectionContext, TimeSpan keepAliveInterval, ILoggerFactory loggerFactory, TimeSpan clientTimeoutInterval);
```

New behavior

The two constructors were removed and replaced with one constructor:

```
public HubConnectionContext(ConnectionContext connectionContext, HubConnectionContextOptions contextOptions, ILoggerFactory loggerFactory)
```

Reason for change

The new constructor uses a new options object. Consequently, the features of `HubConnectionContext` can be expanded in the future without making more constructors and breaking changes.

Recommended action

Instead of using the following constructor:

```
HubConnectionContext connectionContext = new HubConnectionContext(
    connectionContext,
    keepAliveInterval: TimeSpan.FromSeconds(15),
    loggerFactory,
    clientTimeoutInterval: TimeSpan.FromSeconds(15));
```

Use the following constructor:

```
HubConnectionContextOptions contextOptions = new HubConnectionContextOptions()
{
    KeepAliveInterval = TimeSpan.FromSeconds(15),
    ClientTimeoutInterval = TimeSpan.FromSeconds(15)
};
HubConnectionContext connectionContext = new HubConnectionContext(connectionContext, contextOptions,
    loggerFactory);
```

Category

ASP.NET Core

Affected APIs

- [HubConnectionContext\(ConnectionContext, TimeSpan, ILoggerFactory\)](#)
- [HubConnectionContext\(ConnectionContext, TimeSpan, ILoggerFactory, TimeSpan\)](#)

SignalR: JavaScript client package name changed

In ASP.NET Core 3.0 Preview 7, the SignalR JavaScript client package name changed from `@aspnet/signalr` to `@microsoft/signalr`. The name change reflects the fact that SignalR is useful in more than just ASP.NET Core apps, thanks to the Azure SignalR Service.

To react to this change, change references in your `package.json` files, `require` statements, and ECMAScript `import` statements. No API will change as part of this rename.

For discussion, see [dotnet/aspnetcore#11637](#).

Version introduced

3.0

Old behavior

The client package was named `@aspnet/signalr`.

New behavior

The client package is named `@microsoft/signalr`.

Reason for change

The name change clarifies that SignalR is useful beyond ASP.NET Core apps, thanks to the Azure SignalR Service.

Recommended action

Switch to the new package `@microsoft/signalr`.

Category

ASP.NET Core

Affected APIs

None

SignalR: UseSignalR and UseConnections methods marked obsolete

The methods `UseConnections` and `UseSignalR` and the classes `ConnectionsRouteBuilder` and `HubRouteBuilder` are marked as obsolete in ASP.NET Core 3.0.

Version introduced

3.0

Old behavior

SignalR hub routing was configured using `UseSignalR` or `UseConnections`.

New behavior

The old way of configuring routing has been obsoleted and replaced with endpoint routing.

Reason for change

Middleware is being moved to the new endpoint routing system. The old way of adding middleware is being obsoleted.

Recommended action

Replace `UseSignalR` with `UseEndpoints`:

Old code:

```
app.UseSignalR(routes =>
{
    routes.MapHub<SomeHub>("/path");
});
```

New code:

```
app.UseEndpoints(endpoints =>
{
    endpoints.MapHub<SomeHub>("/path");
});
```

Category

ASP.NET Core

Affected APIs

- [Microsoft.AspNetCore.Builder.ConnectionsApplicationBuilderExtensions.UseConnections\(IApplicationBuilder, Action<ConnectionsRouteBuilder>\)](#)
- [Microsoft.AspNetCore.Builder.SignalRApplicationBuilderExtensions.UseSignalR\(IApplicationBuilder, Action<HubRouteBuilder>\)](#)
- [Microsoft.AspNetCore.Http.Connections.ConnectionsRouteBuilder](#)
- [Microsoft.AspNetCore.SignalR.HubRouteBuilder](#)

SPAs: SpaServices and NodeServices marked obsolete

The contents of the following NuGet packages have all been unnecessary since ASP.NET Core 2.1. Consequently, the following packages are being marked as obsolete:

- [Microsoft.AspNetCore.SpaServices](#)
- [Microsoft.AspNetCore.NodeServices](#)

For the same reason, the following npm modules are being marked as deprecated:

- [aspnet-angular](#)
- [aspnet-prerendering](#)
- [aspnet-webpack](#)
- [aspnet-webpack-react](#)
- [domain-task](#)

The preceding packages and npm modules will later be removed in .NET 5.

Version introduced

3.0

Old behavior

The deprecated packages and npm modules were intended to integrate ASP.NET Core with various Single-Page App (SPA) frameworks. Such frameworks include Angular, React, and React with Redux.

New behavior

A new integration mechanism exists in the [Microsoft.AspNetCore.SpaServices.Extensions](#) NuGet package. The package remains the basis of the Angular and React project templates since ASP.NET Core 2.1.

Reason for change

ASP.NET Core supports integration with various Single-Page App (SPA) frameworks, including Angular, React,

and React with Redux. Initially, integration with these frameworks was accomplished with ASP.NET Core-specific components that handled scenarios like server-side prerendering and integration with Webpack. As time went on, industry standards changed. Each of the SPA frameworks released their own standard command-line interfaces. For example, Angular CLI and create-react-app.

When ASP.NET Core 2.1 was released in May 2018, the team responded to the change in standards. A newer and simpler way to integrate with the SPA frameworks' own toolchains was provided. The new integration mechanism exists in the package `Microsoft.AspNetCore.SpaServices.Extensions` and remains the basis of the Angular and React project templates since ASP.NET Core 2.1.

To clarify that the older ASP.NET Core-specific components are irrelevant and not recommended:

- The pre-2.1 integration mechanism is marked as obsolete.
- The supporting npm packages are marked as deprecated.

Recommended action

If you're using these packages, update your apps to use the functionality:

- In the `Microsoft.AspNetCore.SpaServices.Extensions` package.
- Provided by the SPA frameworks you're using

To enable features like server-side prerendering and hot module reload, see the documentation for the corresponding SPA framework. The functionality in `Microsoft.AspNetCore.SpaServices.Extensions` is *not* obsolete and will continue to be supported.

Category

ASP.NET Core

Affected APIs

- `Microsoft.AspNetCore.Builder.SpaRouteExtensions`
- `Microsoft.AspNetCore.Builder.WebpackDevMiddleware`
- `Microsoft.AspNetCore.NodeServices.EmbeddedResourceReader`
- `Microsoft.AspNetCore.NodeServices.INodeServices`
- `Microsoft.AspNetCore.NodeServices.NodeServicesFactory`
- `Microsoft.AspNetCore.NodeServices.NodeServicesOptions`
- `Microsoft.AspNetCore.NodeServices.StringAsTempFile`
- `Microsoft.AspNetCore.NodeServices.HostingModels.INodeInstance`
- `Microsoft.AspNetCore.NodeServices.HostingModels.NodeInvocationException`
- `Microsoft.AspNetCore.NodeServices.HostingModels.NodeInvocationInfo`
- `Microsoft.AspNetCore.NodeServices.HostingModels.NodeServicesOptionsExtensions`
- `Microsoft.AspNetCore.NodeServices.HostingModels.OutOfProcessNodeInstance`
- `Microsoft.AspNetCore.SpaServices.Prerendering.ISpaPrerenderer`
- `Microsoft.AspNetCore.SpaServices.Prerendering.ISpaPrerendererBuilder`
- `Microsoft.AspNetCore.SpaServices.Prerendering.JavaScriptModuleExport`
- `Microsoft.AspNetCore.SpaServices.Prerendering.Prefixer`
- `Microsoft.AspNetCore.SpaServices.Prerendering.PrefixerBuilder`
- `Microsoft.AspNetCore.SpaServices.Prerendering.RenderToStringResult`
- `Microsoft.AspNetCore.SpaServices.Webpack.WebpackDevMiddlewareOptions`
- `Microsoft.Extensions.DependencyInjection.NodeServicesServiceCollectionExtensions`
- `Microsoft.Extensions.DependencyInjection.PrerenderingServiceCollectionExtensions`

SPAs: `SpaServices` and `NodeServices` no longer fall back to console logger

`Microsoft.AspNetCore.SpaServices` and `Microsoft.AspNetCore.NodeServices` won't display console logs unless logging is configured.

Version introduced

3.0

Old behavior

`Microsoft.AspNetCore.SpaServices` and `Microsoft.AspNetCore.NodeServices` used to automatically create a console logger when logging isn't configured.

New behavior

`Microsoft.AspNetCore.SpaServices` and `Microsoft.AspNetCore.NodeServices` won't display console logs unless

logging is configured.

Reason for change

There's a need to align with how other ASP.NET Core packages implement logging.

Recommended action

If the old behavior is required, to configure console logging, add

```
services.AddLogging(builder => builder.AddConsole())
```

Category

ASP.NET Core

Affected APIs

None

Target framework: .NET Framework support dropped

Starting with ASP.NET Core 3.0, .NET Framework is an unsupported target framework.

Change description

.NET Framework 4.8 is the last major version of .NET Framework. New ASP.NET Core apps should be built on .NET Core. Starting with the .NET Core 3.0 release, you can think of ASP.NET Core 3.0 as being part of .NET Core.

Customers using ASP.NET Core with .NET Framework can continue in a fully supported fashion using the [2.1 LTS release](#). Support and servicing for 2.1 continues until at least August 21, 2021. This date is three years after declaration of the LTS release per the [.NET Support Policy](#). Support for ASP.NET Core 2.1 packages on .NET Framework will extend indefinitely, similar to the [servicing policy for other package-based ASP.NET frameworks](#).

For more information about porting from .NET Framework to .NET Core, see [Porting to .NET Core](#).

`Microsoft.Extensions` packages (such as logging, dependency injection, and configuration) and Entity Framework Core aren't affected. They'll continue to support .NET Standard.

For more information on the motivation for this change, see [the original blog post](#).

Version introduced

3.0

Old behavior

ASP.NET Core apps could run on either .NET Core or .NET Framework.

New behavior

ASP.NET Core apps can only be run on .NET Core.

Recommended action

Take one of the following actions:

- Keep your app on ASP.NET Core 2.1.
- Migrate your app and dependencies to .NET Core.

Category

ASP.NET Core

Affected APIs

None

Core .NET libraries

- APIs that report version now report product and not file version
- Custom EncoderFallbackBuffer instances cannot fall back recursively
- Floating point formatting and parsing behavior changes
- Floating-point parsing operations no longer fail or throw an `OverflowException`
- `InvalidOperationException` moved to another assembly
- Replacing ill-formed UTF-8 byte sequences follows Unicode guidelines
- `TypeDescriptionProviderAttribute` moved to another assembly
- `ZipArchiveEntry` no longer handles archives with inconsistent entry sizes
- `FieldInfo.SetValue` throws exception for static, init-only fields
- Passing `GroupCollection` to extension methods taking `IEnumerable<T>` requires disambiguation

APIs that report version now report product and not file version

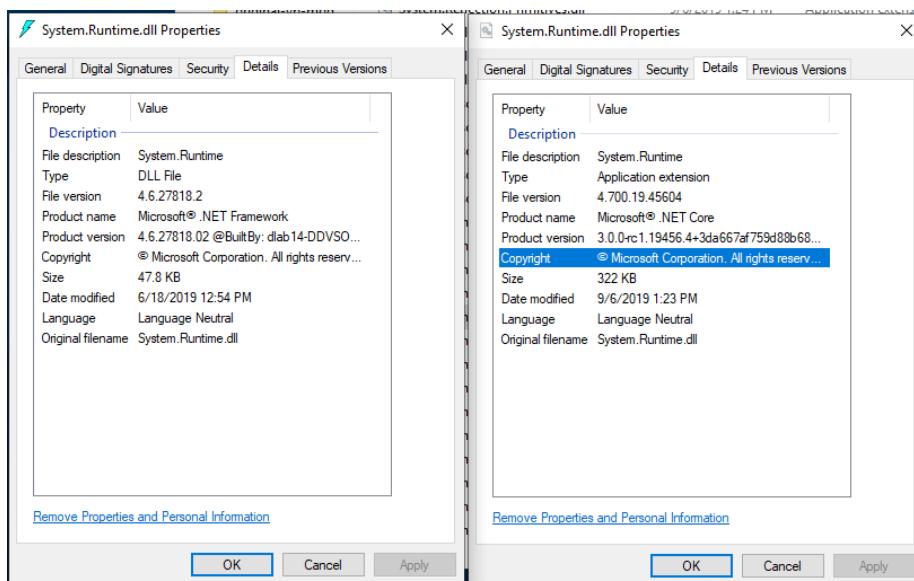
Many of the APIs that return versions in .NET Core now return the product version rather than the file version.

Change description

In .NET Core 2.2 and previous versions, methods such as `Environment.Version`, `RuntimelInformation.FrameworkDescription`, and the file properties dialog for .NET Core assemblies reflect the file version. Starting with .NET Core 3.0, they reflect the product version.

The following figure illustrates the difference in version information for the `System.Runtime.dll` assembly for

.NET Core 2.2 (on the left) and .NET Core 3.0 (on the right) as displayed by the Windows Explorer file properties dialog.



Version introduced

3.0

Recommended action

None. This change should make version detection intuitive rather than obtuse.

Category

Core .NET libraries

Affected APIs

- [Environment.Version](#)
- [RuntimeInformation.FrameworkDescription](#)

Custom EncoderFallbackBuffer instances cannot fall back recursively

Custom [EncoderFallbackBuffer](#) instances cannot fall back recursively. The implementation of [EncoderFallbackBuffer.GetNextChar\(\)](#) must result in a character sequence that is convertible to the destination encoding. Otherwise, an exception occurs.

Change description

During a character-to-byte transcoding operation, the runtime detects ill-formed or nonconvertible UTF-16 sequences and provides those characters to the [EncoderFallbackBuffer.Fallback](#) method. The [Fallback](#) method determines which characters should be substituted for the original nonconvertible data, and these characters are drained by calling [EncoderFallbackBuffer.GetNextChar](#) in a loop.

The runtime then attempts to transcode these substitution characters to the target encoding. If this operation succeeds, the runtime continues transcoding from where it left off in the original input string.

Previously, custom implementations of [EncoderFallbackBuffer.GetNextChar\(\)](#) can return character sequences that are not convertible to the destination encoding. If the substituted characters cannot be transcoded to the target encoding, the runtime invokes the [EncoderFallbackBuffer.Fallback](#) method once again with the substitution characters, expecting the [EncoderFallbackBuffer.GetNextChar\(\)](#) method to return a new substitution sequence. This process continues until the runtime eventually sees a well-formed, convertible substitution, or until a maximum recursion count is reached.

Starting with .NET Core 3.0, custom implementations of [EncoderFallbackBuffer.GetNextChar\(\)](#) must return character sequences that are convertible to the destination encoding. If the substituted characters cannot be transcoded to the target encoding, an [ArgumentException](#) is thrown. The runtime will no longer make recursive calls into the [EncoderFallbackBuffer](#) instance.

This behavior only applies when all three of the following conditions are met:

- The runtime detects an ill-formed UTF-16 sequence or a UTF-16 sequence that cannot be converted to the target encoding.
- A custom [EncoderFallback](#) has been specified.
- The custom [EncoderFallback](#) attempts to substitute a new ill-formed or nonconvertible UTF-16 sequence.

Version introduced

3.0

Recommended action

Most developers needn't take any action.

If an application uses a custom [EncoderFallback](#) and [EncoderFallbackBuffer](#) class, ensure the implementation of

`EncoderFallbackBuffer.Fallback` populates the fallback buffer with well-formed UTF-16 data that is directly convertible to the target encoding when the `Fallback` method is first invoked by the runtime.

Category

Core .NET libraries

Affected APIs

- `EncoderFallbackBuffer.Fallback`
- `EncoderFallbackBuffer.GetNextChar()`

Floating-point formatting and parsing behavior changed

Floating-point parsing and formatting behavior (by the `Double` and `Single` types) are now [IEEE-compliant](#). This ensures that the behavior of floating-point types in .NET matches that of other IEEE-compliant languages. For example, `double.Parse("SomeLiteral")` should always match what C# produces for `double x = SomeLiteral`.

Change description

In .NET Core 2.2 and earlier versions, formatting with `Double.ToString` and `Single.ToString`, and parsing with `Double.Parse`, `Double.TryParse`, `Single.Parse`, and `Single.TryParse` are not IEEE-compliant. As a result, it's impossible to guarantee that a value will roundtrip with any supported standard or custom format string. For some inputs, the attempt to parse a formatted value can fail, and for others, the parsed value doesn't equal the original value.

Starting with .NET Core 3.0, floating-point parsing and formatting operations are IEEE 754-compliant.

The following table shows two code snippets and how the output changes between .NET Core 2.2 and .NET Core 3.1.

CODE SNIPPET	OUTPUT ON .NET CORE 2.2	OUTPUT ON .NET CORE 3.1
<code>Console.WriteLine((-0.0).ToString());</code>	0	-0
<code>var value = -3.123456789123456789; Console.WriteLine(value == double.Parse(value.ToString()));</code>	False	True

For more information, see the [Floating-point parsing and formatting improvements in .NET Core 3.0](#) blog post.

Version introduced

3.0

Recommended action

The [Potential impact to existing code](#) section of the [Floating-point parsing and formatting improvements in .NET Core 3.0](#) blog post suggests some changes you can make to your code if you want to maintain the previous behavior.

- For some differences in formatting, you can get behavior equivalent to the previous behavior by specifying a different format string.
- For differences in parsing, there's no mechanism to fall back to the previous behavior.

Category

Core .NET libraries

Affected APIs

- `Double.ToString`
- `Single.ToString`
- `Double.Parse`
- `Double.TryParse`
- `Single.Parse`
- `Single.TryParse`

Floating-point parsing operations no longer fail or throw an `OverflowException`

The floating-point parsing methods no longer throw an `OverflowException` or return `false` when they parse a string whose numeric value is outside the range of the `Single` or `Double` floating-point type.

Change description

In .NET Core 2.2 and earlier versions, the `Double.Parse` and `Single.Parse` methods throw an `OverflowException` for values that outside the range of their respective type. The `Double.TryParse` and `Single.TryParse` methods return `false` for the string representations of out-of-range numeric values.

Starting with .NET Core 3.0, the `Double.Parse`, `Double.TryParse`, `Single.Parse`, and `Single.TryParse` methods no longer fail when parsing out-of-range numeric strings. Instead, the `Double` parsing methods return `Double.PositiveInfinity` for values that exceed `Double.MaxValue`, and they return `Double.NegativeInfinity` for values that are less than `Double.MinValue`. Similarly, the `Single` parsing methods return `Single.PositiveInfinity` for values that exceed `Single.MaxValue`, and they return `Single.NegativeInfinity` for values that are less than

`Single.MinValue`.

This change was made for improved IEEE 754:2008 compliance.

Version introduced

3.0

Recommended action

This change can affect your code in either of two ways:

- Your code depends on the handler for the `OverflowException` to execute when an overflow occurs. In this case, you should remove the `catch` statement and place any necessary code in an `If` statement that tests whether `Double.IsInfinity` or `Single.IsInfinity` is `true`.
- Your code assumes that floating-point values are not `Infinity`. In this case, you should add the necessary code to check for floating-point values of `PositiveInfinity` and `NegativeInfinity`.

Category

Core .NET libraries

Affected APIs

- `Double.Parse`
- `Double.TryParse`
- `Single.Parse`
- `Single.TryParse`

InvalidAsynchronousStateException moved to another assembly

The `InvalidAsynchronousStateException` class has been moved.

Change description

In .NET Core 2.2 and earlier versions, the `InvalidAsynchronousStateException` class is found in the `System.ComponentModel.TypeConverter` assembly.

Starting with .NET Core 3.0, it is found in the `System.ComponentModel.Primitives` assembly.

Version introduced

3.0

Recommended action

This change only affects applications that use reflection to load the `InvalidAsynchronousStateException` by calling a method such as `Assembly.GetType` or an overload of `Activator.CreateInstance` that assumes the type is in a particular assembly. If that is the case, update the assembly referenced in the method call to reflect the type's new assembly location.

Category

Core .NET libraries

Affected APIs

None.

Replacing ill-formed UTF-8 byte sequences follows Unicode guidelines

When the `UTF8Encoding` class encounters an ill-formed UTF-8 byte sequence during a byte-to-character transcoding operation, it replaces that sequence with a ' REPLACEMENT CHARACTER' character in the output string. .NET Core 3.0 differs from previous versions of .NET Core and the .NET Framework by following the Unicode best practice for performing this replacement during the transcoding operation.

This is part of a larger effort to improve UTF-8 handling throughout .NET, including by the new `System.Text.Unicode.Utf8` and `System.Text.Rune` types. The `UTF8Encoding` type was given improved error handling mechanics so that it produces output consistent with the newly introduced types.

Change description

Starting with .NET Core 3.0, when transcoding bytes to characters, the `UTF8Encoding` class performs character substitution based on Unicode best practices. The substitution mechanism used is described by [The Unicode Standard, Version 12.0, Sec. 3.9 \(PDF\)](#) in the heading titled *U+FFFD Substitution of Maximal Subparts*.

This behavior *only* applies when the input byte sequence contains ill-formed UTF-8 data. Additionally, if the `UTF8Encoding` instance has been constructed with `throwOnInvalidBytes: true`, the `UTF8Encoding` instance will continue to throw on invalid input rather than perform U+FFFD replacement. For more information about the `UTF8Encoding` constructor, see [UTF8Encoding\(Boolean, Boolean\)](#).

The following table illustrates the impact of this change with an invalid 3-byte input:

ILL-FORMED 3-BYTE INPUT	OUTPUT BEFORE .NET CORE 3.0	OUTPUT STARTING WITH .NET CORE 3.0
[ED A0 90]	[FFFD FFFD] (2-character output)	[FFFD FFFD FFFD] (3-character output)

The 3-char output is the preferred output, according to *Table 3-9* of the previously linked Unicode Standard PDF.

Version introduced

3.0

Recommended action

No action is required on the part of the developer.

Category

Core .NET libraries

Affected APIs

- [UTF8Encoding.GetCharCount](#)
- [UTF8Encoding.GetChars](#)
- [UTF8Encoding.GetString\(Byte\[\], Int32, Int32\)](#)

TypeDescriptionProviderAttribute moved to another assembly

The [TypeDescriptionProviderAttribute](#) class has been moved.

Change description

In .NET Core 2.2 and earlier versions, The [TypeDescriptionProviderAttribute](#) class is found in the [System.ComponentModel.TypeConverter](#) assembly.

Starting with .NET Core 3.0, it is found in the [System.ObjectModel](#) assembly.

Version introduced

3.0

Recommended action

This change only affects applications that use reflection to load the [TypeDescriptionProviderAttribute](#) type by calling a method such as [Assembly.GetType](#) or an overload of [Activator.CreateInstance](#) that assumes the type is in a particular assembly. If that is the case, the assembly referenced in the method call should be updated to reflect the type's new assembly location.

Category

Windows Forms

Affected APIs

None.

ZipArchiveEntry no longer handles archives with inconsistent entry sizes

Zip archives list both compressed size and uncompressed size in the central directory and local header. The entry data itself also indicates its size. In .NET Core 2.2 and earlier versions, these values were never checked for consistency. Starting with .NET Core 3.0, they now are.

Change description

In .NET Core 2.2 and earlier versions, [ZipArchiveEntry.Open\(\)](#) succeeds even if the local header disagrees with the central header of the zip file. Data is decompressed until the end of the compressed stream is reached, even if its length exceeds the uncompressed file size listed in the central directory/local header.

Starting with .NET Core 3.0, the [ZipArchiveEntry.Open\(\)](#) method checks that local header and central header agree on compressed and uncompressed sizes of an entry. If they do not, the method throws an [InvalidOperationException](#) if the archive's local header and/or data descriptor list sizes that disagree with the central directory of the zip file. When reading an entry, decompressed data is truncated to the uncompressed file size listed in the header.

This change was made to ensure that a [ZipArchiveEntry](#) correctly represents the size of its data and that only that amount of data is read.

Version introduced

3.0

Recommended action

Repackage any zip archive that exhibits these problems.

Category

Core .NET libraries

Affected APIs

- [ZipArchiveEntry.Open\(\)](#)
- [ZipFileExtensions.ExtractToDirectory](#)
- [ZipFileExtensions.ExtractToFile](#)
- [ZipFile.ExtractToDirectory](#)

FieldInfo.SetValue throws exception for static, init-only fields

Starting in .NET Core 3.0, an exception is thrown when you attempt to set a value on a static, [InitOnly](#) field by calling [System.Reflection.FieldInfo.SetValue](#).

Change description

In .NET Framework and versions of .NET Core prior to 3.0, you could set the value of a static field that's constant after it is initialized ([readonly in C#](#)) by calling `System.Reflection.FieldInfo.SetValue`. However, setting such a field in this way resulted in unpredictable behavior based on the target framework and optimization settings.

In .NET Core 3.0 and later versions, when you call `SetValue` on a static, `InitOnly` field, a `System.FieldAccessException` exception is thrown.

TIP

An `InitOnly` field is one that can only be set at the time it's declared or in the constructor for the containing class. In other words, it's constant after it is initialized.

Version introduced

3.0

Recommended action

Initialize static, `InitOnly` fields in a static constructor. This applies to both dynamic and non-dynamic types.

Alternatively, you can remove the `FieldAttributes.InitOnly` attribute from the field, and then call `FieldInfo.SetValue`.

Category

Core .NET libraries

Affected APIs

- `FieldInfo.SetValue(Object, Object)`
- `FieldInfo.SetValue(Object, Object, BindingFlags, CultureInfo)`

Passing GroupCollection to extension methods taking `IEnumerable<T>` requires disambiguation

When calling an extension method that takes an `IEnumerable<T>` on a `GroupCollection`, you must disambiguate the type using a cast.

Change description

Starting in .NET Core 3.0, `System.Text.RegularExpressions.GroupCollection` implements `IEnumerable<KeyValuePair<String, Group>>` in addition to the other types it implements, including `IEnumerable<Group>`. This results in ambiguity when calling an extension method that takes an `IEnumerable<T>`. If you call such an extension method on a `GroupCollection` instance, for example, `Enumerable.Count`, you'll see the following compiler error:

`CS1061: 'GroupCollection' does not contain a definition for 'Count' and no accessible extension method 'Count' accepting a first argument of type 'GroupCollection' could be found (are you missing a using directive or an assembly reference?)`

In previous versions of .NET, there was no ambiguity and no compiler error.

Version introduced

3.0

Reason for change

This was an [unintentional breaking change](#). Because it has been like this for some time, we don't plan to revert it. In addition, such a change would itself be breaking.

Recommended action

For `GroupCollection` instances, disambiguate calls to extension methods that accept an `IEnumerable<T>` with a cast.

```
// Without a cast - causes CS1061.  
match.Groups.Count(_ => true)  
  
// With a disambiguating cast.  
((IEnumerable<Group>)m.Groups).Count(_ => true);
```

Category

Core .NET libraries

Affected APIs

Any extension method that accepts an `IEnumerable<T>` is affected. For example:

- `System.Collections.Immutable.ImmutableArray.TolImmutableArray<TSource>(IEnumerable<TSource>)`
- `System.Collections.Immutable.ImmutableDictionary.TolImmutableDictionary`
- `System.Collections.Immutable.ImmutableHashSet.TolImmutableHashSet`
- `System.Collections.Immutable.ImmutableList.TolImmutableList<TSource>(IEnumerable<TSource>)`
- `System.Collections.Immutable.ImmutableSortedDictionary.TolImmutableSortedDictionary`
- `System.Collections.Immutable.ImmutableSortedSet.TolImmutableSortedSet`
- `System.Data.DataTableExtensions.Copy ToDataTable`

- Most of the `System.Linq.Enumerable` methods, for example, `System.Linq.Enumerable.Count`
- `System.Linq.ParallelEnumerable.AsParallel`
- `System.Linq.Queryable.AsQueryable`

Cryptography

- `BEGIN TRUSTED CERTIFICATE` syntax no longer supported on Linux
- `EnvelopedCms` defaults to AES-256 encryption
- Minimum size for RSAOpenSsl key generation has increased
- .NET Core 3.0 prefers OpenSSL 1.1.x to OpenSSL 1.0.x
- `CryptoStream.Dispose` transforms final block only when writing

"BEGIN TRUSTED CERTIFICATE" syntax no longer supported for root certificates on Linux

Root certificates on Linux and other Unix-like systems (but not macOS) can be presented in two forms: the standard `BEGIN CERTIFICATE` PEM header, and the OpenSSL-specific `BEGIN TRUSTED CERTIFICATE` PEM header. The latter syntax allows for additional configuration that has caused compatibility issues with .NET Core's `System.Security.Cryptography.X509Certificates.X509Chain` class. `BEGIN TRUSTED CERTIFICATE` root certificate contents are no longer loaded by the chain engine starting in .NET Core 3.0.

Change description

Previously, both the `BEGIN CERTIFICATE` and `BEGIN TRUSTED CERTIFICATE` syntaxes were used to populate the root trust list. If the `BEGIN TRUSTED CERTIFICATE` syntax was used and additional options were specified in the file, `X509Chain` may have reported that the chain trust was explicitly disallowed (`X509ChainStatusFlags.ExplicitDistrust`). However, if the certificate was also specified with the `BEGIN CERTIFICATE` syntax in a previously loaded file, the chain trust was allowed.

Starting in .NET Core 3.0, `BEGIN TRUSTED CERTIFICATE` contents are no longer read. If the certificate is not also specified via a standard `BEGIN CERTIFICATE` syntax, the `X509Chain` reports that the root is not trusted (`X509ChainStatusFlags.UntrustedRoot`).

Version introduced

3.0

Recommended action

Most applications are unaffected by this change, but applications that cannot see both root certificate sources because of permissions problems may experience unexpected `UntrustedRoot` errors after upgrading.

Many Linux distributions (or distros) write root certificates into two locations: a one-certificate-per-file directory, and a one-file concatenation. On some distros, the one-certificate-per-file directory uses the `BEGIN TRUSTED CERTIFICATE` syntax while the file concatenation uses the standard `BEGIN CERTIFICATE` syntax. Ensure that any custom root certificates are added as `BEGIN CERTIFICATE` in at least one of these locations, and that both locations can be read by your application.

The typical directory is `/etc/ssl/certs/` and the typical concatenated file is `/etc/ssl/cert.pem`. Use the command `openssl version -d` to determine the platform-specific root, which may differ from `/etc/ssl/`. For example, on Ubuntu 18.04, the directory is `/usr/lib/ssl/certs/` and the file is `/usr/lib/ssl/cert.pem`. However, `/usr/lib/ssl/certs/` is a symlink to `/etc/ssl/certs/` and `/usr/lib/ssl/cert.pem` does not exist.

```
$ openssl version -d
OPENSSLDIR: "/usr/lib/ssl"
$ ls -al /usr/lib/ssl
total 12
drwxr-xr-x  3 root root 4096 Dec 12 17:10 .
drwxr-xr-x 73 root root 4096 Feb 20 15:18 ..
lrwxrwxrwx  1 root root   14 Mar 27  2018 certs -> /etc/ssl/certs
drwxr-xr-x  2 root root 4096 Dec 12 17:10 misc
lrwxrwxrwx  1 root root   20 Nov 12 16:58 openssl.cnf -> /etc/ssl/openssl.cnf
lrwxrwxrwx  1 root root   16 Mar 27  2018 private -> /etc/ssl/private
```

Category

Cryptography

Affected APIs

- `System.Security.Cryptography.X509Certificates.X509Chain`

EnvelopedCms defaults to AES-256 encryption

The default symmetric encryption algorithm used by `EnvelopedCms` has changed from TripleDES to AES-256.

Change description

In previous versions, when `EnvelopedCms` is used to encrypt data without specifying a symmetric encryption algorithm via a constructor overload, the data is encrypted with the TripleDES/3DES/3DEA/DES3-EDE algorithm.

Starting with .NET Core 3.0 (via version 4.6.0 of the `System.Security.Cryptography.Pkcs` NuGet package), the default algorithm has been changed to AES-256 for algorithm modernization and to improve the security of default options. If a message recipient certificate has a (non-EC) Diffie-Hellman public key, the encryption

operation may fail with a [CryptographicException](#) due to limitations in the underlying platform.

In the following sample code, the data is encrypted with TripleDES if running on .NET Core 2.2 or earlier. If running on .NET Core 3.0 or later, it's encrypted with AES-256.

```
EnvelopedCms cms = new EnvelopedCms(content);
cms.Encrypt(recipient);
return cms.Encode();
```

Version introduced

3.0

Recommended action

If you are negatively impacted by the change, you can restore TripleDES encryption by explicitly specifying the encryption algorithm identifier in an [EnvelopedCms](#) constructor that includes a parameter of type [AlgorithmIdentifier](#), such as:

```
Oid tripleDesOid = new Oid("1.2.840.113549.3.7", null);
AlgorithmIdentifier tripleDesIdentifier = new AlgorithmIdentifier(tripleDesOid);
EnvelopedCms cms = new EnvelopedCms(content, tripleDesIdentifier);

cms.Encrypt(recipient);
return cms.Encode();
```

Category

Cryptography

Affected APIs

- [EnvelopedCms\(\)](#)
- [EnvelopedCms\(ContentInfo\)](#)
- [EnvelopedCms\(SubjectIdentifierType, ContentInfo\)](#)

Minimum size for RSAOpenSsl key generation has increased

The minimum size for generating new RSA keys on Linux has increased from 384-bit to 512-bit.

Change description

Starting with .NET Core 3.0, the minimum legal key size reported by the [LegalKeySizes](#) property on RSA instances from [RSA.Create](#), [RSAOpenSsl](#), and [RSACryptoServiceProvider](#) on Linux has increased from 384 to 512.

As a result, in .NET Core 2.2 and earlier versions, a method call such as `RSA.Create(384)` succeeds. In .NET Core 3.0 and later versions, the method call `RSA.Create(384)` throws an exception indicating the size is too small.

This change was made because OpenSSL, which performs the cryptographic operations on Linux, raised its minimum between versions 1.0.2 and 1.1.0. .NET Core 3.0 prefers OpenSSL 1.1.x to 1.0.x, and the minimum reported version was raised to reflect this new higher dependency limitation.

Version introduced

3.0

Recommended action

If you call any of the affected APIs, ensure that the size of any generated keys is not less than the provider minimum.

NOTE

384-bit RSA is already considered insecure (as is 512-bit RSA). Modern recommendations, such as [NIST Special Publication 800-57 Part 1 Revision 4](#), suggest 2048-bit as the minimum size for newly generated keys.

Category

Cryptography

Affected APIs

- [AsymmetricAlgorithm.LegalKeySizes](#)
- [RSA.Create](#)
- [RSAOpenSsl](#)
- [RSACryptoServiceProvider](#)

.NET Core 3.0 prefers OpenSSL 1.1.x to OpenSSL 1.0.x

.NET Core for Linux, which works across multiple Linux distributions, can support both OpenSSL 1.0.x and OpenSSL 1.1.x. .NET Core 2.1 and .NET Core 2.2 look for 1.0.x first, then fall back to 1.1.x; .NET Core 3.0 looks for 1.1.x first. This change was made to add support for new cryptographic standards.

This change may impact libraries or applications that do platform interop with the OpenSSL-specific interop types in .NET Core.

Change description

In .NET Core 2.2 and earlier versions, the runtime prefers loading OpenSSL 1.0.x over 1.1.x. This means that the `IntPtr` and `SafeHandle` types for interop with OpenSSL are used with `libcrypto.so.1.0.0 / libcrypto.so.1.0 / libcrypto.so.10` by preference.

Starting with .NET Core 3.0, the runtime prefers loading OpenSSL 1.1.x over OpenSSL 1.0.x, so the `IntPtr` and `SafeHandle` types for interop with OpenSSL are used with `libcrypto.so.1.1 / libcrypto.so.11 / libcrypto.so.1.1.0 / libcrypto.so.1.1.1` by preference. As a result, libraries and applications that interoperate with OpenSSL directly may have incompatible pointers with the .NET Core-exposed values when upgrading from .NET Core 2.1 or .NET Core 2.2.

Version introduced

3.0

Recommended action

Libraries and applications that do direct operations with OpenSSL need to be careful to ensure they are using the same version of OpenSSL as the .NET Core runtime.

All libraries or applications that use `IntPtr` or `SafeHandle` values from the .NET Core cryptographic types directly with OpenSSL should compare the version of the library they use with the new `SafeEvpPKeyHandle.OpenSslVersion` property to ensure the pointers are compatible.

Category

Cryptography

Affected APIs

- [SafeEvpPKeyHandle](#)
- [RSAOpenSsl\(IntPtr\)](#)
- [RSAOpenSsl\(SafeEvpPKeyHandle\)](#)
- [RSAOpenSsl.DuplicateKeyHandle\(\)](#)
- [DSAOpenSsl\(IntPtr\)](#)
- [DSAOpenSsl\(SafeEvpPKeyHandle\)](#)
- [DSAOpenSsl.DuplicateKeyHandle\(\)](#)
- [ECDsaOpenSsl\(IntPtr\)](#)
- [ECDsaOpenSsl\(SafeEvpPKeyHandle\)](#)
- [ECDsaOpenSsl.DuplicateKeyHandle\(\)](#)
- [ECDiffieHellmanOpenSsl\(IntPtr\)](#)
- [ECDiffieHellmanOpenSsl\(SafeEvpPKeyHandle\)](#)
- [ECDiffieHellmanOpenSsl.DuplicateKeyHandle\(\)](#)
- [X509Certificate.Handle](#)

CryptoStream.Dispose transforms final block only when writing

The `CryptoStream.Dispose` method, which is used to finish `CryptoStream` operations, no longer attempts to transform the final block when reading.

Change description

In previous .NET versions, if a user performed an incomplete read when using `CryptoStream` in `Read` mode, the `Dispose` method could throw an exception (for example, when using AES with padding). The exception was thrown because the final block was attempted to be transformed but the data was incomplete.

In .NET Core 3.0 and later versions, `Dispose` no longer tries to transform the final block when reading, which allows for incomplete reads.

Reason for change

This change enables incomplete reads from the crypto stream when a network operation is canceled, without the need to catch an exception.

Version introduced

3.0

Recommended action

Most apps should not be affected by this change.

If your application previously caught an exception in case of an incomplete read, you can delete that `catch` block. If your app used transforming of the final block in hashing scenarios, you might need to ensure that the entire stream is read before it's disposed.

Category

Cryptography

Affected APIs

- [System.Security.Cryptography.CryptoStream.Dispose](#)

Globalization

- "C" locale maps to the invariant locale

"C" locale maps to the invariant locale

.NET Core 2.2 and earlier versions depend on the default ICU behavior, which maps the "C" locale to the en_US_POSIX locale. The en_US_POSIX locale has an undesirable collation behavior, because it doesn't support case-insensitive string comparisons. Because some Linux distributions set the "C" locale as the default locale, users were experiencing unexpected behavior.

Change description

Starting with .NET Core 3.0, the "C" locale mapping has changed to use the Invariant locale instead of en_US_POSIX. The "C" locale to Invariant mapping is also applied to Windows for consistency.

Mapping "C" to en_US_POSIX culture caused customer confusion, because en_US_POSIX doesn't support case insensitive sorting/searching string operations. Because the "C" locale is used as a default locale in some of the Linux distros, customers experienced this undesired behavior on these operating systems.

Version introduced

3.0

Recommended action

Nothing specific more than the awareness of this change. This change affects only applications that use the "C" locale mapping.

Category

Globalization

Affected APIs

All collation and culture APIs are affected by this change.

MSBuild

- Resource manifest file name change

Resource manifest file name change

Starting in .NET Core 3.0, in the default case, MSBuild generates a different manifest file name for resource files.

Version introduced

3.0

Change description

Prior to .NET Core 3.0, if no `LogicalName`, `ManifestResourceName`, or `DependentUpon` metadata was specified for an `EmbeddedResource` item in the project file, MSBuild generated a manifest file name in the pattern `<RootNamespace>.<ResourceFilePathFromProjectRoot>.resources`. If `RootNamespace` is not defined in the project file, it defaults to the project name. For example, the generated manifest name for a resource file named `Form1.resx` in the root project directory was `MyProject.Form1.resources`.

Starting in .NET Core 3.0, if a resource file is colocated with a source file of the same name (for example, `Form1.resx` and `Form1.cs`), MSBuild uses type information from the source file to generate the manifest file name in the pattern `<Namespace>.<ClassName>.resources`. The namespace and class name are extracted from the first type in the colocated source file. For example, the generated manifest name for a resource file named `Form1.resx` that's colocated with a source file named `Form1.cs` is `MyNamespace.Form1.resources`. The key thing to note is that the first part of the file name is different to prior versions of .NET Core (`MyNamespace` instead of `MyProject`).

NOTE

If you have `LogicalName`, `ManifestResourceName`, or `DependentUpon` metadata specified on an `EmbeddedResource` item in the project file, then this change does not affect that resource file.

This breaking change was introduced with the addition of the `EmbeddedResourceUseDependentUponConvention` property to .NET Core projects. By default, resource files aren't explicitly listed in a .NET Core project file, so they have no `DependentUpon` metadata to specify how to name the generated `.resources` file. When `EmbeddedResourceUseDependentUponConvention` is set to `true`, which is the default, MSBuild looks for a colocated source file and extracts a namespace and class name from that file. If you set `EmbeddedResourceUseDependentUponConvention` to `false`, MSBuild generates the manifest name according to the previous behavior, which combines `RootNamespace` and the relative file path.

Recommended action

In most cases, no action is required on the part of the developer, and your app should continue to work. However, if this change breaks your app, you can either:

- Change your code to expect the new manifest name.

- Opt out of the new naming convention by setting `EmbeddedResourceUseDependentUponConvention` to `false` in your project file.

```
<PropertyGroup>
  <EmbeddedResourceUseDependentUponConvention>false</EmbeddedResourceUseDependentUponConvention>
</PropertyGroup>
```

Category

MSBuild

Affected APIs

N/A

Networking

- Default value of `HttpRequestMessage.Version` changed to 1.1

Default value of `HttpRequestMessage.Version` changed to 1.1

The default value of the `System.Net.Http.HttpRequestMessage.Version` property has changed from 2.0 to 1.1.

Version introduced

3.0

Change description

In .NET Core 1.0 through 2.0, the default value of the `System.Net.Http.HttpRequestMessage.Version` property is 1.1. Starting with .NET Core 2.1, it was changed to 2.1.

Starting with .NET Core 3.0, the default version number returned by the `System.Net.Http.HttpRequestMessage.Version` property is once again 1.1.

Recommended action

Update your code if it depends on the `System.Net.Http.HttpRequestMessage.Version` property returning a default value of 2.0.

Category

Networking

Affected APIs

- `System.Net.Http.HttpRequestMessage.Version`

What's new in .NET Core 2.2

9/20/2022 • 3 minutes to read • [Edit Online](#)

.NET Core 2.2 includes enhancements in application deployment, event handling for runtime services, authentication to Azure SQL databases, JIT compiler performance, and code injection prior to the execution of the `Main` method.

New deployment mode

Starting with .NET Core 2.2, you can deploy [framework-dependent executables](#), which are `.exe` files instead of `.dll` files. Functionally similar to framework-dependent deployments, framework-dependent executables (FDE) still rely on the presence of a shared system-wide version of .NET Core to run. Your app contains only your code and any third-party dependencies. Unlike framework-dependent deployments, FDEs are platform-specific.

This new deployment mode has the distinct advantage of building an executable instead of a library, which means you can run your app directly without invoking `dotnet` first.

Core

Handling events in runtime services

You may often want to monitor your application's use of runtime services, such as the GC, JIT, and ThreadPool, to understand how they impact your application. On Windows systems, this is commonly done by monitoring the ETW events of the current process. While this continues to work well, it's not always possible to use ETW if you're running in a low-privilege environment or on Linux or macOS.

Starting with .NET Core 2.2, CoreCLR events can now be consumed using the [System.Diagnostics.Tracing.EventListener](#) class. These events describe the behavior of such runtime services as GC, JIT, ThreadPool, and interop. These are the same events that are exposed as part of the CoreCLR ETW provider. This allows for applications to consume these events or use a transport mechanism to send them to a telemetry aggregation service. You can see how to subscribe to events in the following code sample:

```

internal sealed class SimpleEventListener : EventListener
{
    // Called whenever an EventSource is created.
    protected override void OnEventSourceCreated(EventSource eventSource)
    {
        // Watch for the .NET runtime EventSource and enable all of its events.
        if (eventSource.Name.Equals("Microsoft-Windows-DotNETRuntime"))
        {
            EnableEvents(eventSource, EventLevel.Verbose, (EventKeywords)(-1));
        }
    }

    // Called whenever an event is written.
    protected override void OnEventWritten(EventWrittenEventArgs eventData)
    {
        // Write the contents of the event to the console.
        Console.WriteLine($"ThreadID = {eventData.OSThreadId} ID = {eventData.EventId} Name =
{eventData.EventName}");
        for (int i = 0; i < eventData.Payload.Count; i++)
        {
            string payloadString = eventData.Payload[i]?.ToString() ?? string.Empty;
            Console.WriteLine($" \tName = \"{eventData.PayloadNames[i]}\" Value = \"{payloadString}\"");
        }
        Console.WriteLine("\n");
    }
}

```

In addition, .NET Core 2.2 adds the following two properties to the [EventWrittenEventArgs](#) class to provide additional information about ETW events:

- [EventWrittenEventArgs.OSThreadId](#)
- [EventWrittenEventArgs.TimeStamp](#)

Data

AAD authentication to Azure SQL databases with the `SqlConnection.AccessToken` property

Starting with .NET Core 2.2, an access token issued by Azure Active Directory can be used to authenticate to an Azure SQL database. To support access tokens, the [AccessToken](#) property has been added to the [SqlConnection](#) class. To take advantage of AAD authentication, download version 4.6 of the System.Data.SqlClient NuGet package. In order to use the feature, you can obtain the access token value using the [Active Directory Authentication Library for .NET](#) contained in the [Microsoft.IdentityModel.Clients.ActiveDirectory](#) NuGet package.

JIT compiler improvements

Tiered compilation remains an opt-in feature

In .NET Core 2.1, the JIT compiler implemented a new compiler technology, *tiered compilation*, as an opt-in feature. The goal of tiered compilation is improved performance. One of the important tasks performed by the JIT compiler is optimizing code execution. For little-used code paths, however, the compiler may spend more time optimizing code than the runtime spends executing unoptimized code. Tiered compilation introduces two stages in JIT compilation:

- A **first tier**, which generates code as quickly as possible.
- A **second tier**, which generates optimized code for those methods that are executed frequently. The second tier of compilation is performed in parallel for enhanced performance.

For information on the performance improvement that can result from tiered compilation, see [Announcing .NET Core 2.2 Preview 2](#).

For information about opting in to tiered compilation, see [Jit compiler improvements](#) in [What's new in .NET Core 2.1](#).

Runtime

Injecting code prior to executing the Main method

Starting with .NET Core 2.2, you can use a startup hook to inject code prior to running an application's Main method. Startup hooks make it possible for a host to customize the behavior of applications after they have been deployed without needing to recompile or change the application.

We expect hosting providers to define custom configuration and policy, including settings that potentially influence the load behavior of the main entry point, such as the [`System.Runtime.Loader.AssemblyLoadContext`](#) behavior. The hook can be used to set up tracing or telemetry injection, to set up callbacks for handling, or to define other environment-dependent behavior. The hook is separate from the entry point, so that user code doesn't need to be modified.

See [Host startup hook](#) for more information.

See also

- [What's new in .NET Core 3.1](#)
- [What's new in ASP.NET Core 2.2](#)
- [New features in EF Core 2.2](#)

What's new in .NET Core 2.1

9/20/2022 • 10 minutes to read • [Edit Online](#)

.NET Core 2.1 includes enhancements and new features in the following areas:

- [Tooling](#)
- [Roll forward](#)
- [Deployment](#)
- [Windows Compatibility Pack](#)
- [JIT compilation improvements](#)
- [API changes](#)

Tooling

The .NET Core 2.1 SDK (v 2.1.300), the tooling included with .NET Core 2.1, includes the following changes and enhancements:

Build performance improvements

A major focus of .NET Core 2.1 is improving build-time performance, particularly for incremental builds. These performance improvements apply to both command-line builds using `dotnet build` and to builds in Visual Studio. Some individual areas of improvement include:

- For package asset resolution, resolving only assets used by a build rather than all assets.
- Caching of assembly references.
- Use of long-running SDK build servers, which are processes that span across individual `dotnet build` invocations. They eliminate the need to JIT-compile large blocks of code every time `dotnet build` is run. Build server processes can be automatically terminated with the following command:

```
dotnet buildserver shutdown
```

New CLI commands

A number of tools that were available only on a per project basis using `DotnetCliToolReference` are now available as part of the .NET Core SDK. These tools include:

- `dotnet watch` provides a file system watcher that waits for a file to change before executing a designated set of commands. For example, the following command automatically rebuilds the current project and generates verbose output whenever a file in it changes:

```
dotnet watch -- --verbose build
```

Note the `--` option that precedes the `--verbose` option. It delimits the options passed directly to the `dotnet watch` command from the arguments that are passed to the child `dotnet` process. Without it, the `--verbose` option applies to the `dotnet watch` command, not the `dotnet build` command.

For more information, see [Develop ASP.NET Core apps using dotnet watch](#).

- `dotnet dev-certs` generates and manages certificates used during development in ASP.NET Core applications.

- `dotnet user-secrets` manages the secrets in a user secret store in ASP.NET Core applications.
- `dotnet sql-cache` creates a table and indexes in a Microsoft SQL Server database to be used for distributed caching.
- `dotnet ef` is a tool for managing databases, `DbContext` objects, and migrations in Entity Framework Core applications. For more information, see [EF Core .NET Command-line Tools](#).

Global Tools

.NET Core 2.1 supports *Global Tools* -- that is, custom tools that are available globally from the command line. The extensibility model in previous versions of .NET Core made custom tools available on a per project basis only by using `DotnetCliToolReference`.

To install a Global Tool, you use the `dotnet tool install` command. For example:

```
dotnet tool install -g dotnetsay
```

Once installed, the tool can be run from the command line by specifying the tool name. For more information, see [.NET Core Global Tools overview](#).

Tool management with the `dotnet tool` command

In .NET Core 2.1 SDK, all tools operations use the `dotnet tool` command. The following options are available:

- `dotnet tool install` to install a tool.
- `dotnet tool update` to uninstall and reinstall a tool, which effectively updates it.
- `dotnet tool list` to list currently installed tools.
- `dotnet tool uninstall` to uninstall currently installed tools.

Roll forward

All .NET Core applications starting with .NET Core 2.0 automatically roll forward to the latest *minor version* installed on a system.

Starting with .NET Core 2.0, if the version of .NET Core that an application was built with is not present at run time, the application automatically runs against the latest installed *minor version* of .NET Core. In other words, if an application is built with .NET Core 2.0, and .NET Core 2.0 is not present on the host system but .NET Core 2.1 is, the application runs with .NET Core 2.1.

IMPORTANT

This roll-forward behavior doesn't apply to preview releases. By default, it also doesn't apply to major releases, but this can be changed with the settings below.

You can modify this behavior by changing the setting for the roll-forward on no candidate shared framework.

The available settings are:

- `0` - disable minor version roll-forward behavior. With this setting, an application built for .NET Core 2.0.0 will roll forward to .NET Core 2.0.1, but not to .NET Core 2.2.0 or .NET Core 3.0.0.
- `1` - enable minor version roll-forward behavior. This is the default value for the setting. With this setting, an application built for .NET Core 2.0.0 will roll forward to either .NET Core 2.0.1 or .NET Core 2.2.0, depending on which one is installed, but it will not roll forward to .NET Core 3.0.0.
- `2` - enable minor and major version roll-forward behavior. If set, even different major versions are

considered, so an application built for .NET Core 2.0.0 will roll forward to .NET Core 3.0.0.

You can modify this setting in any of three ways:

- Set the `DOTNET_ROLL_FORWARD_ON_NO_CANDIDATE_FX` environment variable to the desired value.
- Add the following line with the desired value to the `.runtimeconfig.json` file:

```
"rollForwardOnNoCandidateFx" : 0
```

- When using the [.NET Core CLI](#), add the following option with the desired value to a .NET Core command such as `run`:

```
dotnet run --rollForwardOnNoCandidateFx=0
```

Patch version roll forward is independent of this setting and is done after any potential minor or major version roll forward is applied.

Deployment

Self-contained application servicing

`dotnet publish` now publishes self-contained applications with a serviced runtime version. When you publish a self-contained application with the .NET Core 2.1 SDK (v 2.1.300), your application includes the latest serviced runtime version known by that SDK. When you upgrade to the latest SDK, you'll publish with the latest .NET Core runtime version. This applies for .NET Core 1.0 runtimes and later.

Self-contained publishing relies on runtime versions on NuGet.org. You do not need to have the serviced runtime on your machine.

Using the .NET Core 2.0 SDK, self-contained applications are published with the .NET Core 2.0.0 runtime unless a different version is specified via the `RuntimeFrameworkVersion` property. With this new behavior, you'll no longer need to set this property to select a higher runtime version for a self-contained application. The easiest approach going forward is to always publish with .NET Core 2.1 SDK (v 2.1.300).

For more information, see [Self-contained deployment runtime roll forward](#).

Windows Compatibility Pack

When you port existing code from the .NET Framework to .NET Core, you can use the [Windows Compatibility Pack](#). It provides access to 20,000 more APIs than are available in .NET Core. These APIs include types in the `System.Drawing` namespace, the `EventLog` class, WMI, Performance Counters, Windows Services, and the Windows registry types and members.

JIT compiler improvements

.NET Core incorporates a new JIT compiler technology called *tiered compilation* (also known as *adaptive optimization*) that can significantly improve performance. Tiered compilation is an opt-in setting.

One of the important tasks performed by the JIT compiler is optimizing code execution. For little-used code paths, however, the compiler may spend more time optimizing code than the runtime spends running unoptimized code. Tiered compilation introduces two stages in JIT compilation:

- A **first tier**, which generates code as quickly as possible.
- A **second tier**, which generates optimized code for those methods that are executed frequently. The

second tier of compilation is performed in parallel for enhanced performance.

You can opt into tiered compilation in either of two ways.

- To use tiered compilation in all projects that use the .NET Core 2.1 SDK, set the following environment variable:

```
COMPlus_TieredCompilation="1"
```

- To use tiered compilation on a per-project basis, add the `<TieredCompilation>` property to the `<PropertyGroup>` section of the MSBuild project file, as the following example shows:

```
<PropertyGroup>
    <!-- other property definitions -->

    <TieredCompilation>true</TieredCompilation>
</PropertyGroup>
```

API changes

`Span<T>` and `Memory<T>`

.NET Core 2.1 includes some new types that make working with arrays and other types of memory much more efficient. The new types include:

- `System.Span<T>` and `System.ReadOnlySpan<T>`.
- `System.Memory<T>` and `System.ReadOnlyMemory<T>`.

Without these types, when passing such items as a portion of an array or a section of a memory buffer, you have to make a copy of some portion of the data before passing it to a method. These types provide a virtual view of that data that eliminates the need for the additional memory allocation and copy operations.

The following example uses a `Span<T>` and `Memory<T>` instance to provide a virtual view of 10 elements of an array.

```
using System;

class Program
{
    static void Main()
    {
        int[] numbers = new int[100];
        for (int i = 0; i < 100; i++)
        {
            numbers[i] = i * 2;
        }

        var part = new Span<int>(numbers, start: 10, length: 10);
        foreach (var value in part)
            Console.Write($"{value} ");
    }
}

// The example displays the following output:
//      20 22 24 26 28 30 32 34 36 38
```

```

Module Program
Sub Main()
    Dim numbers As Integer() = New Integer(99) {}

    For i As Integer = 0 To 99
        numbers(i) = i * 2
    Next

    Dim part = New Memory(Of Integer)(numbers, start:=10, length:=10)

    For Each value In part.Span
        Console.WriteLine($"{value} ")
    Next
End Sub
End Module
' The example displays the following output:
'      20  22  24  26  28  30  32  34  36  38

```

Brotli compression

.NET Core 2.1 adds support for Brotli compression and decompression. Brotli is a general-purpose lossless compression algorithm that is defined in [RFC 7932](#) and is supported by most web browsers and major web servers. You can use the stream-based [System.IO.Compression.BrotliStream](#) class or the high-performance span-based [System.IO.Compression.BrotliEncoder](#) and [System.IO.Compression.BrotliDecoder](#) classes. The following example illustrates compression with the [BrotliStream](#) class:

```

public static Stream DecompressWithBrotli(Stream toDecompress)
{
    MemoryStream decompressedStream = new MemoryStream();
    using (BrotliStream decompressionStream = new BrotliStream(toDecompress, CompressionMode.Decompress))
    {
        decompressionStream.CopyTo(decompressedStream);
    }
    decompressedStream.Position = 0;
    return decompressedStream;
}

```

```

Public Function DecompressWithBrotli(toDecompress As Stream) As Stream
    Dim decompressedStream As New MemoryStream()
    Using decompressionStream As New BrotliStream(toDecompress, CompressionMode.Decompress)
        decompressionStream.CopyTo(decompressedStream)
    End Using
    decompressedStream.Position = 0
    Return decompressedStream
End Function

```

The [BrotliStream](#) behavior is the same as [DeflateStream](#) and [GZipStream](#), which makes it easy to convert code that calls these APIs to [BrotliStream](#).

New cryptography APIs and cryptography improvements

.NET Core 2.1 includes numerous enhancements to the cryptography APIs:

- [System.Security.Cryptography.Pkcs.SignedCms](#) is available in the [System.Security.Cryptography.Pkcs](#) package. The implementation is the same as the [SignedCms](#) class in the .NET Framework.
- New overloads of the [X509Certificate.GetCertHash](#) and [X509Certificate.GetCertHashString](#) methods accept a hash algorithm identifier to enable callers to get certificate thumbprint values using algorithms other than SHA-1.
- New [Span<T>](#)-based cryptography APIs are available for hashing, HMAC, cryptographic random number

generation, asymmetric signature generation, asymmetric signature processing, and RSA encryption.

- The performance of [System.Security.Cryptography.Rfc2898DeriveBytes](#) has improved by about 15% by using a [Span<T>](#)-based implementation.
- The new [System.Security.Cryptography.CryptographicOperations](#) class includes two new methods:
 - [FixedTimeEquals](#) takes a fixed amount of time to return for any two inputs of the same length, which makes it suitable for use in cryptographic verification to avoid contributing to timing side-channel information.
 - [ZeroMemory](#) is a memory-clearing routine that cannot be optimized.
- The static [RandomNumberGenerator.Fill](#) method fills a [Span<T>](#) with random values.
- The [System.Security.Cryptography.Pkcs.EnvelopedCms](#) is now supported on Linux and macOS.
- Elliptic-Curve Diffie-Hellman (ECDH) is now available in the [System.Security.Cryptography.ECDiffieHellman](#) class family. The surface area is the same as in the .NET Framework.
- The instance returned by [RSA.Create](#) can encrypt or decrypt with OAEP using a SHA-2 digest, as well as generate or validate signatures using RSA-PSS.

Sockets improvements

.NET Core includes a new type, [System.Net.Http.SocketsHttpHandler](#), and a rewritten [System.Net.Http.HttpMessageHandler](#), that form the basis of higher-level networking APIs. [System.Net.Http.SocketsHttpHandler](#), for example, is the basis of the [HttpClient](#) implementation. In previous versions of .NET Core, higher-level APIs were based on native networking implementations.

The sockets implementation introduced in .NET Core 2.1 has a number of advantages:

- A significant performance improvement when compared with the previous implementation.
- Elimination of platform dependencies, which simplifies deployment and servicing.
- Consistent behavior across all .NET Core platforms.

[SocketsHttpHandler](#) is the default implementation in .NET Core 2.1. However, you can configure your application to use the older [HttpClientHandler](#) class by calling the [AppContext.SetSwitch](#) method:

```
AppContext.SetSwitch("System.Net.Http.UseSocketsHttpHandler", false);
```

```
AppContext.SetSwitch("System.Net.Http.UseSocketsHttpHandler", False)
```

You can also use an environment variable to opt out of using sockets implementations based on [SocketsHttpHandler](#). To do this, set the `DOTNET_SYSTEM_NET_HTTP_USESOCKETSHTTPHANDLER` to either `false` or 0.

On Windows, you can also choose to use [System.Net.Http.WinHttpHandler](#), which relies on a native implementation, or the [SocketsHttpHandler](#) class by passing an instance of the class to the [HttpClient](#) constructor.

On Linux and macOS, you can only configure [HttpClient](#) on a per-process basis. On Linux, you need to deploy [libcurl](#) if you want to use the old [HttpClient](#) implementation. (It is installed with .NET Core 2.0.)

Breaking changes

For information about breaking changes, see [Breaking changes for migration from version 2.0 to 2.1](#).

See also

- [What's new in .NET Core 3.1](#)
- [New features in EF Core 2.1](#)
- [What's new in ASP.NET Core 2.1](#)

Breaking changes in .NET Core 2.1

9/20/2022 • 6 minutes to read • [Edit Online](#)

If you're migrating to version 2.1 of .NET Core, the breaking changes listed in this article may affect your app.

Core .NET libraries

- [Path APIs don't throw an exception for invalid characters](#)
- [Private fields added to built-in struct types](#)
- [OpenSSL versions on macOS](#)

Path APIs don't throw an exception for invalid characters

APIs that involve file paths no longer validate path characters or throw an [ArgumentException](#) if an invalid character is found.

Change description

In .NET Framework and .NET Core 1.0 - 2.0, the methods listed in the [Affected APIs](#) section throw an [ArgumentException](#) if the path argument contains an invalid path character. Starting in .NET Core 2.1, these methods no longer check for [invalid path characters](#) or throw an exception if an invalid character is found.

Reason for change

Aggressive validation of path characters blocks some cross-platform scenarios. This change was introduced so that .NET does not try to replicate or predict the outcome of operating system API calls. For more information, see the [System.IO in .NET Core 2.1 sneak peek](#) blog post.

Version introduced

.NET Core 2.1

Recommended action

If your code relied on these APIs to check for invalid characters, you can add a call to [Path.GetInvalidPathChars](#).

Affected APIs

- [System.IO.Directory.CreateDirectory](#)
- [System.IO.Directory.Delete](#)
- [System.IO.Directory.EnumerateDirectories](#)
- [System.IO.Directory.EnumerateFiles](#)
- [System.IO.Directory.EnumerateFileSystemEntries](#)
- [System.IO.Directory.GetCreationTime\(String\)](#)
- [System.IO.Directory.GetCreationTimeUtc\(String\)](#)
- [System.IO.Directory.GetDirectories](#)
- [System.IO.Directory.GetDirectoryRoot\(String\)](#)
- [System.IO.Directory.GetFiles](#)
- [System.IO.Directory.GetFileSystemEntries](#)
- [System.IO.Directory.GetLastAccessTime\(String\)](#)
- [System.IO.Directory.GetLastAccessTimeUtc\(String\)](#)
- [System.IO.Directory.GetLastWriteTime\(String\)](#)
- [System.IO.Directory.GetLastWriteTimeUtc\(String\)](#)
- [System.IO.Directory.GetParent\(String\)](#)
- [System.IO.Directory.Move\(String, String\)](#)

- [System.IO.Directory.SetCreationTime\(String, DateTime\)](#)
- [System.IO.Directory.SetCreationTimeUtc\(String, DateTime\)](#)
- [System.IO.Directory.SetCurrentDirectory\(String\)](#)
- [System.IO.Directory.SetLastAccessTime\(String, DateTime\)](#)
- [System.IO.Directory.SetLastAccessTimeUtc\(String, DateTime\)](#)
- [System.IO.Directory.SetLastWriteTime\(String, DateTime\)](#)
- [System.IO.Directory.SetLastWriteTimeUtc\(String, DateTime\)](#)
- [System.IO.DirectoryInfo ctor](#)
- [System.IO.Directory.GetDirectories](#)
- [System.IO.Directory.GetFiles](#)
- [System.IO.DirectoryInfo.GetFileSystemInfos](#)
- [System.IO.File.AppendAllText](#)
- [System.IO.File.AppendAllTextAsync](#)
- [System.IO.File.Copy](#)
- [System.IO.File.Create](#)
- [System.IO.File.CreateText](#)
- [System.IO.File.Decrypt](#)
- [System.IO.File.Delete](#)
- [System.IO.File.Encrypt](#)
- [System.IO.File.GetAttributes\(String\)](#)
- [System.IO.File.GetCreationTime\(String\)](#)
- [System.IO.File.GetCreationTimeUtc\(String\)](#)
- [System.IO.File.GetLastAccessTime\(String\)](#)
- [System.IO.File.GetLastAccessTimeUtc\(String\)](#)
- [System.IO.File.GetLastWriteTime\(String\)](#)
- [System.IO.File.GetLastWriteTimeUtc\(String\)](#)
- [System.IO.File.Move](#)
- [System.IO.File.Open](#)
- [System.IO.File.OpenRead\(String\)](#)
- [System.IO.File.OpenText\(String\)](#)
- [System.IO.File.OpenWrite\(String\)](#)
- [System.IO.File.ReadAllBytes\(String\)](#)
- [System.IO.File.ReadAllBytesAsync\(String, CancellationToken\)](#)
- [System.IO.File.ReadAllLines\(String\)](#)
- [System.IO.File.ReadAllLinesAsync\(String, CancellationToken\)](#)
- [System.IO.File.ReadAllText\(String\)](#)
- [System.IO.File.ReadAllTextAsync\(String, CancellationToken\)](#)
- [System.IO.File.SetAttributes\(String, FileAttributes\)](#)
- [System.IO.File.SetCreationTime\(String, DateTime\)](#)
- [System.IO.File.SetCreationTimeUtc\(String, DateTime\)](#)
- [System.IO.File.SetLastAccessTime\(String, DateTime\)](#)
- [System.IO.File.SetLastAccessTimeUtc\(String, DateTime\)](#)
- [System.IO.File.SetLastWriteTime\(String, DateTime\)](#)
- [System.IO.File.SetLastWriteTimeUtc\(String, DateTime\)](#)
- [System.IO.File.WriteAllBytes\(String, Byte\[\]\)](#)
- [System.IO.File.WriteAllBytesAsync\(String, Byte\[\], CancellationToken\)](#)

- [System.IO.File.WriteAllText](#)
- [System.IO.File.WriteAllTextAsync](#)
- [System.IO.File.WriteAllText](#)
- [System.IO.FileInfo ctor](#)
- [System.IO.FileInfo.CopyTo](#)
- [System.IO.FileInfo.MoveTo](#)
- [System.IO.FileStream ctor](#)
- [System.IO.Path.GetFullPath\(String\)](#)
- [System.IO.Path.IsPathRooted\(String\)](#)
- [System.IO.Path.GetPathRoot\(String\)](#)
- [System.IO.Path.ChangeExtension\(String, String\)](#)
- [System.IO.Path.GetDirectoryName\(String\)](#)
- [System.IO.Path.GetExtension\(String\)](#)
- [System.IO.Path.HasExtension\(String\)](#)
- [System.IO.Path.Combine](#)

See also

- [System.IO in .NET Core 2.1 sneak peek](#)

Private fields added to built-in struct types

Private fields were added to [certain struct types](#) in [reference assemblies](#). As a result, in C#, those struct types must always be instantiated by using the [new operator](#) or [default literal](#).

Change description

In .NET Core 2.0 and previous versions, some provided struct types, for example, [ConsoleKeyInfo](#), could be instantiated without using the [new](#) operator or [default literal](#) in C#. This was because the [reference assemblies](#) used by the C# compiler didn't contain the private fields for the structs. All private fields for .NET struct types are added to the reference assemblies starting in .NET Core 2.1.

For example, the following C# code compiles in .NET Core 2.0, but not in .NET Core 2.1:

```
ConsoleKeyInfo key; // Struct type

if (key.ToString() == "y")
{
    Console.WriteLine("Yes!");
}
```

In .NET Core 2.1, the previous code results in the following compiler error: [CS0165 - Use of unassigned local variable 'key'](#)

Version introduced

2.1

Recommended action

Instantiate struct types by using the [new](#) operator or [default literal](#).

For example:

```
ConsoleKeyInfo key = new ConsoleKeyInfo(); // Struct type.

if (key.ToString() == "y")
    Console.WriteLine("Yes!");
```

```
ConsoleKeyInfo key = default; // Struct type.  
  
if (key.ToString() == "y")  
    Console.WriteLine("Yes!");
```

Category

Core .NET libraries

Affected APIs

- [System.ArraySegment<T>.Enumerator](#)
- [System.ArraySegment<T>](#)
- [System.Boolean](#)
- [System.Buffers.MemoryHandle](#)
- [System.Buffers.StandardFormat](#)
- [System.Byte](#)
- [System.Char](#)
- [System.Collections.DictionaryEntry](#)
- [System.Collections.Generic.Dictionary<TKey,TValue>.Enumerator](#)
- [System.Collections.Generic.Dictionary<TKey,TValue>.KeyCollection.Enumerator](#)
- [System.Collections.Generic.Dictionary<TKey,TValue>.ValueCollection.Enumerator](#)
- [System.Collections.Generic.HashSet<T>.Enumerator](#)
- [System.Collections.Generic.KeyValuePair<TKey,TValue>](#)
- [System.Collections.Generic.LinkedList<T>.Enumerator](#)
- [System.Collections.Generic.List<T>.Enumerator](#)
- [System.Collections.Generic.Queue<T>.Enumerator](#)
- [System.Collections.Generic.SortedDictionary<TKey,TValue>.Enumerator](#)
- [System.Collections.Generic.SortedDictionary<TKey,TValue>.KeyCollection.Enumerator](#)
- [System.Collections.Generic.SortedDictionary<TKey,TValue>.ValueCollection.Enumerator](#)
- [System.Collections.Generic.SortedSet<T>.Enumerator](#)
- [System.Collections.Generic.Stack<T>.Enumerator](#)
- [System.Collections.Immutable.ImmutableArray<T>.Enumerator](#)
- [System.Collections.Immutable.ImmutableArray<T>](#)
- [System.Collections.Immutable.ImmutableDictionary<TKey,TValue>.Enumerator](#)
- [System.Collections.Immutable.ImmutableHashSet<T>.Enumerator](#)
- [System.Collections.Immutable.ImmutableList<T>.Enumerator](#)
- [System.Collections.Immutable.ImmutableQueue<T>.Enumerator](#)
- [System.Collections.Immutable.ImmutableSortedDictionary<TKey,TValue>.Enumerator](#)
- [System.Collections.Immutable.ImmutableSortedSet<T>.Enumerator](#)
- [System.Collections.Immutable.ImmutableStack<T>.Enumerator](#)
- [System.Collections.Specialized.BitVector32.Section](#)
- [System.Collections.Specialized.BitVector32](#)
- [LazyMemberInfo](#)
- [System.ComponentModel.Design.Serialization.MemberRelationship](#)
- [System.ConsoleKeyInfo](#)
- [System.Data.SqlTypes.SqlBinary](#)
- [System.Data.SqlTypes.SqlBoolean](#)
- [System.Data.SqlTypes.SqlByte](#)
- [System.Data.SqlTypes.SqlDateTime](#)

- [System.Data.SqlTypes.SqlDecimal](#)
- [System.Data.SqlTypes.SqlDouble](#)
- [System.Data.SqlTypes.SqlGuid](#)
- [System.Data.SqlTypes.SqlInt16](#)
- [System.Data.SqlTypes.SqlInt32](#)
- [System.Data.SqlTypes.SqlInt64](#)
- [System.Data.SqlTypes.SqlMoney](#)
- [System.Data.SqlTypes.SqlSingle](#)
- [System.Data.SqlTypes.SqlString](#)
- [System.DateTime](#)
- [System.DateTimeOffset](#)
- [System.Decimal](#)
- [System.Diagnostics.CounterSample](#)
- [System.Diagnostics.SymbolStore.SymbolToken](#)
- [System.Diagnostics.Tracing.EventSourceEventArgs](#)
- [System.Diagnostics.Tracing.EventSourceOptions](#)
- [System.Double](#)
- [System.Drawing.CharacterRange](#)
- [System.Drawing.Point](#)
- [System.Drawing.PointF](#)
- [System.Drawing.Rectangle](#)
- [System.Drawing.RectangleF](#)
- [System.Drawing.Size](#)
- [System.Drawing.SizeF](#)
- [System.Guid](#)
- [System.HashCode](#)
- [System.Int16](#)
- [System.Int32](#)
- [System.Int64](#)
- [System.IntPtr](#)
- [System.IO.Pipelines.FlushResult](#)
- [System.IO.Pipelines.ReadResult](#)
- [System.IO.WaitForChangedResult](#)
- [System.Memory<T>](#)
- [System.ModuleHandle](#)
- [System.Net.Security.SslApplicationProtocol](#)
- [System.Net.Sockets.IPPacketInformation](#)
- [System.Net.Sockets.SocketInformation](#)
- [System.Net.Sockets.UdpReceiveResult](#)
- [System.Net.WebSockets.ValueWebSocketReceiveResult](#)
- [System.Nullable<T>](#)
- [System.Numerics.BigInteger](#)
- [System.Numerics.Complex](#)
- [System.Numerics.Vector<T>](#)
- [System.ReadOnlyMemory<T>](#)
- [System.ReadOnlySpan<T>.Enumerator](#)

- [System.ReadOnlySpan<T>](#)
- [System.Reflection.CustomAttributeNamedArgument](#)
- [System.Reflection.CustomAttributeTypedArgument](#)
- [System.Reflection.Emit.Label](#)
- [System.Reflection.Emit.OpCode](#)
- [System.Reflection.Metadata.ArrayShape](#)
- [System.Reflection.Metadata.AssemblyDefinition](#)
- [System.Reflection.Metadata.AssemblyDefinitionHandle](#)
- [System.Reflection.Metadata.AssemblyFile](#)
- [System.Reflection.Metadata.AssemblyFileHandle](#)
- [System.Reflection.Metadata.AssemblyFileHandleCollection.Enumerator](#)
- [System.Reflection.Metadata.AssemblyFileHandleCollection](#)
- [System.Reflection.Metadata.AssemblyReference](#)
- [System.Reflection.Metadata.AssemblyReferenceHandle](#)
- [System.Reflection.Metadata.AssemblyReferenceHandleCollection.Enumerator](#)
- [System.Reflection.Metadata.AssemblyReferenceHandleCollection](#)
- [System.Reflection.Metadata.Blob](#)
- [System.Reflection.Metadata.BlobBuilder.Blobs](#)
- [System.Reflection.Metadata.BlobContentId](#)
- [System.Reflection.Metadata.BlobHandle](#)
- [System.Reflection.Metadata.BlobReader](#)
- [System.Reflection.Metadata.BlobWriter](#)
- [System.Reflection.Metadata.Constant](#)
- [System.Reflection.Metadata.ConstantHandle](#)
- [System.Reflection.Metadata.CustomAttribute](#)
- [System.Reflection.Metadata.CustomAttributeHandle](#)
- [System.Reflection.Metadata.CustomAttributeHandleCollection.Enumerator](#)
- [System.Reflection.Metadata.CustomAttributeHandleCollection](#)
- [System.Reflection.Metadata.CustomAttributeNamedArgument< TType >](#)
- [System.Reflection.Metadata.CustomAttributeTypedArgument< TType >](#)
- [System.Reflection.Metadata.CustomAttributeValue< TType >](#)
- [System.Reflection.Metadata.CustomDebugInformation](#)
- [System.Reflection.Metadata.CustomDebugInformationHandle](#)
- [System.Reflection.Metadata.CustomDebugInformationHandleCollection.Enumerator](#)
- [System.Reflection.Metadata.CustomDebugInformationHandleCollection](#)
- [System.Reflection.Metadata.DeclarativeSecurityAttribute](#)
- [System.Reflection.Metadata.DeclarativeSecurityAttributeHandle](#)
- [System.Reflection.Metadata.DeclarativeSecurityAttributeHandleCollection.Enumerator](#)
- [System.Reflection.Metadata.DeclarativeSecurityAttributeHandleCollection](#)
- [System.Reflection.Metadata.Document](#)
- [System.Reflection.Metadata.DocumentHandle](#)
- [System.Reflection.Metadata.DocumentHandleCollection.Enumerator](#)
- [System.Reflection.Metadata.DocumentHandleCollection](#)
- [System.Reflection.Metadata.DocumentNameBlobHandle](#)
- [System.Reflection.Metadata.Ecma335.ArrayShapeEncoder](#)
- [System.Reflection.Metadata.Ecma335.BlobEncoder](#)

- [System.Reflection.Metadata.Ecma335.CustomAttributeArrayTypeEncoder](#)
- [System.Reflection.Metadata.Ecma335.CustomAttributeElementTypeEncoder](#)
- [System.Reflection.Metadata.Ecma335.CustomAttributeNamedArgumentsEncoder](#)
- [System.Reflection.Metadata.Ecma335.CustomModifiersEncoder](#)
- [System.Reflection.Metadata.Ecma335.EditAndContinueLogEntry](#)
- [System.Reflection.Metadata.Ecma335.ExceptionRegionEncoder](#)
- [System.Reflection.Metadata.Ecma335.FixedArgumentsEncoder](#)
- [System.Reflection.Metadata.Ecma335.GenericTypeArgumentsEncoder](#)
- [System.Reflection.Metadata.Ecma335.InstructionEncoder](#)
- [System.Reflection.Metadata.Ecma335.LabelHandle](#)
- [System.Reflection.Metadata.Ecma335.LiteralEncoder](#)
- [System.Reflection.Metadata.Ecma335.LiteralsEncoder](#)
- [System.Reflection.Metadata.Ecma335.LocalVariablesEncoder](#)
- [System.Reflection.Metadata.Ecma335.LocalVariableTypeEncoder](#)
- [System.Reflection.Metadata.Ecma335.MethodBodyStreamEncoder.MethodBody](#)
- [System.Reflection.Metadata.Ecma335.MethodBodyStreamEncoder](#)
- [System.Reflection.Metadata.Ecma335.MethodSignatureEncoder](#)
- [System.Reflection.Metadata.Ecma335.NamedArgumentsEncoder](#)
- [System.Reflection.Metadata.Ecma335.NamedArgumentTypeEncoder](#)
- [System.Reflection.Metadata.Ecma335.NameEncoder](#)
- [System.Reflection.Metadata.Ecma335.ParametersEncoder](#)
- [System.Reflection.Metadata.Ecma335.ParameterTypeEncoder](#)
- [System.Reflection.Metadata.Ecma335.PermissionSetEncoder](#)
- [System.Reflection.Metadata.Ecma335.ReturnTypeEncoder](#)
- [System.Reflection.Metadata.Ecma335.ScalarEncoder](#)
- [System.Reflection.Metadata.Ecma335.SignatureDecoder< TType, TGenericContext >](#)
- [System.Reflection.Metadata.Ecma335.SignatureTypeEncoder](#)
- [System.Reflection.Metadata.Ecma335.VectorEncoder](#)
- [System.Reflection.Metadata.EntityHandle](#)
- [System.Reflection.Metadata.EventAccessors](#)
- [System.Reflection.Metadata.EventDefinition](#)
- [System.Reflection.Metadata.EventDefinitionHandle](#)
- [System.Reflection.Metadata.EventDefinitionHandleCollection.Enumerator](#)
- [System.Reflection.Metadata.EventDefinitionHandleCollection](#)
- [System.Reflection.Metadata.ExceptionRegion](#)
- [System.Reflection.Metadata.ExportedType](#)
- [System.Reflection.Metadata.ExportedTypeHandle](#)
- [System.Reflection.Metadata.ExportedTypeHandleCollection.Enumerator](#)
- [System.Reflection.Metadata.ExportedTypeHandleCollection](#)
- [System.Reflection.Metadata.FieldDefinition](#)
- [System.Reflection.Metadata.FieldDefinitionHandle](#)
- [System.Reflection.Metadata.FieldDefinitionHandleCollection.Enumerator](#)
- [System.Reflection.Metadata.FieldDefinitionHandleCollection](#)
- [System.Reflection.Metadata.GenericParameter](#)
- [System.Reflection.Metadata.GenericParameterConstraint](#)
- [System.Reflection.Metadata.GenericParameterConstraintHandle](#)

- [System.Reflection.Metadata.GenericParameterConstraintHandleCollection.Enumerator](#)
- [System.Reflection.Metadata.GenericParameterConstraintHandleCollection](#)
- [System.Reflection.Metadata.GenericParameterHandle](#)
- [System.Reflection.Metadata.GenericParameterHandleCollection.Enumerator](#)
- [System.Reflection.Metadata.GenericParameterHandleCollection](#)
- [System.Reflection.Metadata.GuidHandle](#)
- [System.Reflection.Metadata.Handle](#)
- [System.Reflection.Metadata.ImportDefinition](#)
- [System.Reflection.Metadata.ImportDefinitionCollection.Enumerator](#)
- [System.Reflection.Metadata.ImportDefinitionCollection](#)
- [System.Reflection.Metadata.ImportScope](#)
- [System.Reflection.Metadata.ImportScopeCollection.Enumerator](#)
- [System.Reflection.Metadata.ImportScopeCollection](#)
- [System.Reflection.Metadata.ImportScopeHandle](#)
- [System.Reflection.Metadata.InterfaceImplementation](#)
- [System.Reflection.Metadata.InterfaceImplementationHandle](#)
- [System.Reflection.Metadata.InterfaceImplementationHandleCollection.Enumerator](#)
- [System.Reflection.Metadata.InterfaceImplementationHandleCollection](#)
- [System.Reflection.Metadata.LocalConstant](#)
- [System.Reflection.Metadata.LocalConstantHandle](#)
- [System.Reflection.Metadata.LocalConstantHandleCollection.Enumerator](#)
- [System.Reflection.Metadata.LocalConstantHandleCollection](#)
- [System.Reflection.Metadata.LocalScope](#)
- [System.Reflection.Metadata.LocalScopeHandle](#)
- [System.Reflection.Metadata.LocalScopeHandleCollection.ChildrenEnumerator](#)
- [System.Reflection.Metadata.LocalScopeHandleCollection.Enumerator](#)
- [System.Reflection.Metadata.LocalScopeHandleCollection](#)
- [System.Reflection.Metadata.LocalVariable](#)
- [System.Reflection.Metadata.LocalVariableHandle](#)
- [System.Reflection.Metadata.LocalVariableHandleCollection.Enumerator](#)
- [System.Reflection.Metadata.LocalVariableHandleCollection](#)
- [System.Reflection.Metadata.ManifestResource](#)
- [System.Reflection.Metadata.ManifestResourceHandle](#)
- [System.Reflection.Metadata.ManifestResourceHandleCollection.Enumerator](#)
- [System.Reflection.Metadata.ManifestResourceHandleCollection](#)
- [System.Reflection.Metadata.MemberReference](#)
- [System.Reflection.Metadata.MemberReferenceHandle](#)
- [System.Reflection.Metadata.MemberReferenceHandleCollection.Enumerator](#)
- [System.Reflection.Metadata.MemberReferenceHandleCollection](#)
- [System.Reflection.Metadata.MetadataStringComparer](#)
- [System.Reflection.Metadata.MethodDebugInformation](#)
- [System.Reflection.Metadata.MethodDebugInformationHandle](#)
- [System.Reflection.Metadata.MethodDebugInformationHandleCollection.Enumerator](#)
- [System.Reflection.Metadata.MethodDebugInformationHandleCollection](#)
- [System.Reflection.Metadata.MethodDefinition](#)
- [System.Reflection.Metadata.MethodDefinitionHandle](#)

- [System.Reflection.Metadata.MethodDefinitionHandleCollection.Enumerator](#)
- [System.Reflection.Metadata.MethodDefinitionHandleCollection](#)
- [System.Reflection.Metadata.MethodImplementation](#)
- [System.Reflection.Metadata.MethodImplementationHandle](#)
- [System.Reflection.Metadata.MethodImplementationHandleCollection.Enumerator](#)
- [System.Reflection.Metadata.MethodImplementationHandleCollection](#)
- [System.Reflection.Metadata.MethodImport](#)
- [System.Reflection.Metadata.MethodSignature<TType>](#)
- [System.Reflection.Metadata.MethodSpecification](#)
- [System.Reflection.Metadata.MethodSpecificationHandle](#)
- [System.Reflection.Metadata.ModuleDefinition](#)
- [System.Reflection.Metadata.ModuleDefinitionHandle](#)
- [System.Reflection.Metadata.ModuleReference](#)
- [System.Reflection.Metadata.ModuleReferenceHandle](#)
- [System.Reflection.Metadata.NamespaceDefinition](#)
- [System.Reflection.Metadata.NamespaceDefinitionHandle](#)
- [System.Reflection.Metadata.Parameter](#)
- [System.Reflection.Metadata.ParameterHandle](#)
- [System.Reflection.Metadata.ParameterHandleCollection.Enumerator](#)
- [System.Reflection.Metadata.ParameterHandleCollection](#)
- [System.Reflection.Metadata.PropertyAccessors](#)
- [System.Reflection.Metadata.PropertyDefinition](#)
- [System.Reflection.Metadata.PropertyDefinitionHandle](#)
- [System.Reflection.Metadata.PropertyDefinitionHandleCollection.Enumerator](#)
- [System.Reflection.Metadata.PropertyDefinitionHandleCollection](#)
- [System.Reflection.Metadata.ReservedBlob<THandle>](#)
- [System.Reflection.Metadata.SequencePoint](#)
- [System.Reflection.Metadata.SequencePointCollection.Enumerator](#)
- [System.Reflection.Metadata.SequencePointCollection](#)
- [System.Reflection.Metadata.SignatureHeader](#)
- [System.Reflection.Metadata.StandaloneSignature](#)
- [System.Reflection.Metadata.StandaloneSignatureHandle](#)
- [System.Reflection.Metadata.StringHandle](#)
- [System.Reflection.Metadata.TypeDefinition](#)
- [System.Reflection.Metadata.TypeDefinitionHandle](#)
- [System.Reflection.Metadata.TypeDefinitionHandleCollection.Enumerator](#)
- [System.Reflection.Metadata.TypeDefinitionHandleCollection](#)
- [System.Reflection.Metadata.TypeLayout](#)
- [System.Reflection.Metadata.TypeReference](#)
- [System.Reflection.Metadata.TypeReferenceHandle](#)
- [System.Reflection.Metadata.TypeReferenceHandleCollection.Enumerator](#)
- [System.Reflection.Metadata.TypeReferenceHandleCollection](#)
- [System.Reflection.Metadata.TypeSpecification](#)
- [System.Reflection.Metadata.TypeSpecificationHandle](#)
- [System.Reflection.Metadata.UserStringHandle](#)
- [System.Reflection.ParameterModifier](#)

- [System.Reflection.PortableExecutable.CodeViewDebugDirectoryData](#)
- [System.Reflection.PortableExecutable.DebugDirectoryEntry](#)
- [System.Reflection.PortableExecutable.PEMemoryBlock](#)
- [System.Reflection.PortableExecutable.SectionHeader](#)
- [System.Runtime.CompilerServices.AsyncTaskMethodBuilder<TResult>](#)
- [System.Runtime.CompilerServices.AsyncTaskMethodBuilder](#)
- [System.Runtime.CompilerServices.AsyncValueTaskMethodBuilder<TResult>](#)
- [System.Runtime.CompilerServices.AsyncValueTaskMethodBuilder](#)
- [System.Runtime.CompilerServices.AsyncVoidMethodBuilder](#)
- [System.Runtime.CompilerServices.ConfiguredTaskAwaitable<TResult>.ConfiguredTaskAwaiter](#)
- [System.Runtime.CompilerServices.ConfiguredTaskAwaitable<TResult>](#)
- [System.Runtime.CompilerServices.ConfiguredTaskAwaitable.ConfiguredTaskAwaiter](#)
- [System.Runtime.CompilerServices.ConfiguredTaskAwaitable](#)
- [System.Runtime.CompilerServices.ConfiguredValueTaskAwaitable<TResult>](#)
- [System.Runtime.CompilerServices.ConfiguredValueTaskAwaitable<TResult>.ConfiguredValueTaskAwaiter](#)
- [System.Runtime.CompilerServices.TaskAwaiter<TResult>](#)
- [System.Runtime.CompilerServices.TaskAwaiter](#)
- [System.Runtime.CompilerServices.ValueTaskAwaiter<TResult>](#)
- [System.Runtime.CompilerServices.ValueTaskAwaiter<TResult>](#)
- [System.Runtime.InteropServices.ArrayWithOffset](#)
- [System.Runtime.InteropServices.GCHandle](#)
- [System.Runtime.InteropServices.HandleRef](#)
- [System.Runtime.InteropServices.OSPlatform](#)
- [System.Runtime.InteropServices.WindowsRuntime.EventRegistrationToken](#)
- [System.Runtime.Serialization.SerializationEntry](#)
- [System.Runtime.Serialization.StreamingContext](#)
- [System.RuntimeArgumentHandle](#)
- [System.RuntimeFieldHandle](#)
- [System.RuntimeMethodHandle](#)
- [System.RuntimeTypeHandle](#)
- [System.SByte](#)
- [System.Security.Cryptography.CngProperty](#)
- [System.Security.Cryptography.ECCurve](#)
- [System.Security.Cryptography.HashAlgorithmName](#)
- [System.Security.Cryptography.X509Certificates.X509ChainStatus](#)
- [System.Security.Cryptography.Xml.X509IssuerSerial](#)
- [System.ServiceProcess.SessionChangeDescription](#)
- [System.Single](#)
- [System.Span<T>.Enumerator](#)
- [System.Span<T>](#)
- [System.Threading.AsyncFlowControl](#)
- [System.Threading.AsyncLocalValueChangedArgs<T>](#)
- [System.Threading.CancellationToken](#)
- [System.Threading.CancellationTokenRegistration](#)
- [System.Threading.LockCookie](#)
- [System.Threading.SpinLock](#)

- [System.Threading.SpinWait](#)
- [System.Threading.Tasks.Dataflow.DataflowMessageHeader](#)
- [System.Threading.Tasks.ParallelLoopResult](#)
- [System.Threading.Tasks.ValueTask<TResult>](#)
- [System.TimeSpan](#)
- [System.TimeZoneInfo.TransitionTime](#)
- [System.Transactions.TransactionOptions](#)
- [System.TypedReference](#)
- [System.TypedReference](#)
- [System.UInt16](#)
- [System.UInt32](#)
- [System.UInt64](#)
- [System.UIntPtr](#)
- [System.Windows.Forms.ColorDialog.Color](#)
- [System.Windows.Media.Animation.KeyTime](#)
- [System.Windows.Media.Animation.RepeatBehavior](#)
- [System.Xml.Serialization.XmlDeserializationEvents](#)
- [Windows.Foundation.Point](#)
- [Windows.Foundation.Rect](#)
- [Windows.Foundation.Size](#)
- [Windows.UI.Color](#)
- [Windows.UI.Xaml.Controls.Primitives.GeneratorPosition](#)
- [Windows.UI.Xaml.CornerRadius](#)
- [Windows.UI.Xaml.Duration](#)
- [Windows.UI.Xaml.GridLength](#)
- [Windows.UI.Xaml.Media.Matrix](#)
- [Windows.UI.Xaml.Media.Media3D.Matrix3D](#)
- [Windows.UI.Xaml.Thickness](#)

OpenSSL versions on macOS

The .NET Core 3.0 and later runtimes on macOS now prefer OpenSSL 1.1.x versions to OpenSSL 1.0.x versions for the [AesCcm](#), [AesGcm](#), [DSAOpenSsl](#), [ECDiffieHellmanOpenSsl](#), [ECDsaOpenSsl](#), [RSAOpenSsl](#), and [SafeEvpPKeyHandle](#) types.

The .NET Core 2.1 runtime now supports OpenSSL 1.1.x versions, but still prefers OpenSSL 1.0.x versions.

Change description

Previously, the .NET Core runtime used OpenSSL 1.0.x versions on macOS for types that interact with OpenSSL. The most recent OpenSSL 1.0.x version, OpenSSL 1.0.2, is now out of support. To keep types that use OpenSSL on supported versions of OpenSSL, the .NET Core 3.0 and later runtimes now use newer versions of OpenSSL on macOS.

With this change, the behavior for the .NET Core runtimes on macOS is as follows:

- The .NET Core 3.0 and later version runtimes use OpenSSL 1.1.x, if available, and fall back to OpenSSL 1.0.x only if there's no 1.1.x version available.

For callers that use the OpenSSL interop types with custom P/Invokes, follow the guidance in the [SafeEvpPKeyHandle.OpenSslVersion](#) remarks. Your app may crash if you don't check the [OpenSslVersion](#) value.

- The .NET Core 2.1 runtime uses OpenSSL 1.0.x, if available, and falls back to OpenSSL 1.1.x if there's no 1.0.x version available.

The 2.1 runtime prefers the earlier version of OpenSSL because the [SafeEvpPKeyHandle.OpenSslVersion](#) property does not exist in .NET Core 2.1, so the OpenSSL version cannot be reliably determined at run time.

Version introduced

- .NET Core 2.1.16
- .NET Core 3.0.3
- .NET Core 3.1.2

Recommended action

- Uninstall OpenSSL version 1.0.2 if it's no longer needed.
- Install OpenSSL 1.1.x if you use the [AesCcm](#), [AesGcm](#), [DSAOpenSsl](#), [ECDiffieHellmanOpenSsl](#), [ECDsaOpenSsl](#), [RSAOpenSsl](#), or [SafeEvpPKeyHandle](#) types.
- If you use the OpenSSL interop types with custom P/Invokes, follow the guidance in the [SafeEvpPKeyHandle.OpenSslVersion](#) remarks.

Category

Core .NET libraries

Affected APIs

- [System.Security.Cryptography.AesCcm](#)
- [System.Security.Cryptography.AesGcm](#)
- [System.Security.Cryptography.DSAOpenSsl](#)
- [System.Security.Cryptography.ECDiffieHellmanOpenSsl](#)
- [System.Security.Cryptography.ECDsaOpenSsl](#)
- [System.Security.Cryptography.RSAOpenSsl](#)
- [System.Security.Cryptography.SafeEvpPKeyHandle](#)

MSBuild

- [Project tools now included in SDK](#)

Project tools now included in SDK

The .NET Core 2.1 SDK now includes common CLI tooling, and you no longer need to reference these tools from the project.

Change description

In .NET Core 2.0, projects reference external .NET tools with the `<DotNetCliToolReference>` project setting. In .NET Core 2.1, some of these tools are included with the .NET Core SDK, and the setting is no longer needed. If you include references to these tools in your project, you'll receive an error similar to the following: **The tool 'Microsoft.EntityFrameworkCore.Tools.DotNet' is now included in the .NET Core SDK.**

Tools now included in .NET Core 2.1 SDK:

<DOTNETCLITOOLREFERENCE> VALUE	TOOL
<code>Microsoft.DotNet.Watcher.Tools</code>	<code>dotnet-watch</code>
<code>Microsoft.Extensions.SecretManager.Tools</code>	<code>dotnet-user-secrets</code>

<DOTNETCLITOOLREFERENCE> VALUE	TOOL
Microsoft.Extensions.Caching.SqlConfig.Tools	dotnet-sql-cache
Microsoft.EntityFrameworkCore.Tools.DotNet	dotnet-ef

Version introduced

.NET Core SDK 2.1.300

Recommended action

Remove the <DotNetCliToolReference> setting from your project.

Category

MSBuild

Affected APIs

N/A

What's new in .NET Core 2.0

9/20/2022 • 6 minutes to read • [Edit Online](#)

.NET Core 2.0 includes enhancements and new features in the following areas:

- [Tooling](#)
- [Language support](#)
- [Platform improvements](#)
- [API changes](#)
- [Visual Studio integration](#)
- [Documentation improvements](#)

Tooling

dotnet restore runs implicitly

In previous versions of .NET Core, you had to run the `dotnet restore` command to download dependencies immediately after you created a new project with the `dotnet new` command, as well as whenever you added a new dependency to your project.

You don't have to run `dotnet restore` because it's run implicitly by all commands that require a restore to occur, such as `dotnet new`, `dotnet build`, `dotnet run`, `dotnet test`, `dotnet publish`, and `dotnet pack`. To disable implicit restore, use the `--no-restore` option.

The `dotnet restore` command is still useful in certain scenarios where explicitly restoring makes sense, such as [continuous integration builds in Azure DevOps Services](#) or in build systems that need to explicitly control when the restore occurs.

For information about how to manage NuGet feeds, see the [dotnet restore documentation](#).

You can also disable the automatic invocation of `dotnet restore` by passing the `--no-restore` switch to the `new`, `run`, `build`, `publish`, `pack`, and `test` commands.

Retargeting to .NET Core 2.0

If the .NET Core 2.0 SDK is installed, projects that target .NET Core 1.x can be retargeted to .NET Core 2.0.

To retarget to .NET Core 2.0, edit your project file by changing the value of the `<TargetFramework>` element (or the `<TargetFrameworks>` element if you have more than one target in your project file) from 1.x to 2.0:

```
<PropertyGroup>
  <TargetFramework>netcoreapp2.0</TargetFramework>
</PropertyGroup>
```

You can also retarget .NET Standard libraries to .NET Standard 2.0 the same way:

```
<PropertyGroup>
  <TargetFramework>netstandard2.0</TargetFramework>
</PropertyGroup>
```

For more information about migrating your project to .NET Core 2.0, see [Migrating from ASP.NET Core 1.x to ASP.NET Core 2.0](#).

Language support

In addition to supporting C# and F#, .NET Core 2.0 also supports Visual Basic.

Visual Basic

With version 2.0, .NET Core now supports Visual Basic 2017. You can use Visual Basic to create the following project types:

- .NET Core console apps
- .NET Core class libraries
- .NET Standard class libraries
- .NET Core unit test projects
- .NET Core xUnit test projects

For example, to create a Visual Basic "Hello World" application, do the following steps from the command line:

1. Open a console window, create a directory for your project, and make it the current directory.
2. Enter the command `dotnet new console -lang vb`.

The command creates a project file with a `.vbproj` file extension, along with a Visual Basic source code file named `Program.vb`. This file contains the source code to write the string "Hello World!" to the console window.

3. Enter the command `dotnet run`. The [.NET Core CLI](#) automatically compiles and executes the application, which displays the message "Hello World!" in the console window.

Support for C# 7.1

.NET Core 2.0 supports C# 7.1, which adds a number of new features, including:

- The `Main` method, the application entry point, can be marked with the `async` keyword.
- Inferred tuple names.
- Default expressions.

Platform improvements

.NET Core 2.0 includes a number of features that make it easier to install .NET Core and to use it on supported operating systems.

.NET Core for Linux is a single implementation

.NET Core 2.0 offers a single Linux implementation that works on multiple Linux distributions. .NET Core 1.x required that you download a distribution-specific Linux implementation.

You can also develop apps that target Linux as a single operating system. .NET Core 1.x required that you target each Linux distribution separately.

Support for the Apple cryptographic libraries

.NET Core 1.x on macOS required the OpenSSL toolkit's cryptographic library. .NET Core 2.0 uses the Apple cryptographic libraries and doesn't require OpenSSL, so you no longer need to install it.

API changes and library support

Support for .NET Standard 2.0

.NET Standard defines a versioned set of APIs that must be available on .NET implementations that comply with that version of the standard. .NET Standard is targeted at library developers. It aims to guarantee the functionality that is available to a library that targets a version of .NET Standard on each .NET implementation.

.NET Core 1.x supports .NET Standard version 1.6; .NET Core 2.0 supports the latest version, .NET Standard 2.0. For more information, see [.NET Standard](#).

.NET Standard 2.0 includes over 20,000 more APIs than were available in .NET Standard 1.6. Much of this expanded surface area results from incorporating APIs that are common to the .NET Framework and Xamarin into .NET Standard.

.NET Standard 2.0 class libraries can also reference .NET Framework class libraries, provided that they call APIs that are present in .NET Standard 2.0. No recompilation of the .NET Framework libraries is required.

For a list of the APIs that have been added to .NET Standard since its last version, .NET Standard 1.6, see [.NET Standard 2.0 vs. 1.6](#).

Expanded surface area

The total number of APIs available on .NET Core 2.0 has more than doubled in comparison with .NET Core 1.1.

And with the [Windows Compatibility Pack](#) porting from .NET Framework has also become much simpler.

Support for .NET Framework libraries

.NET Core code can reference existing .NET Framework libraries, including existing NuGet packages. Note that the libraries must use APIs that are found in .NET Standard.

Visual Studio integration

Visual Studio 2017 version 15.3 and in some cases Visual Studio for Mac offer a number of significant enhancements for .NET Core developers.

Retargeting .NET Core apps and .NET Standard libraries

If the .NET Core 2.0 SDK is installed, you can retarget .NET Core 1.x projects to .NET Core 2.0 and .NET Standard 1.x libraries to .NET Standard 2.0.

To retarget your project in Visual Studio, you open the **Application** tab of the project's properties dialog and change the **Target framework** value to **.NET Core 2.0** or **.NET Standard 2.0**. You can also change it by right-clicking on the project and selecting the **Edit *.csproj** file option. For more information, see the [Tooling](#) section earlier in this topic.

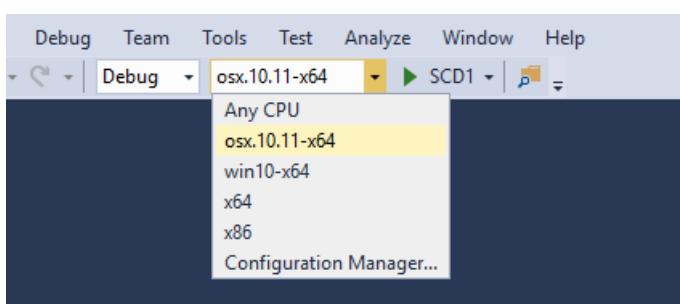
Live Unit Testing support for .NET Core

Whenever you modify your code, Live Unit Testing automatically runs any affected unit tests in the background and displays the results and code coverage live in the Visual Studio environment. .NET Core 2.0 now supports Live Unit Testing. Previously, Live Unit Testing was available only for .NET Framework applications.

For more information, see [Live Unit Testing with Visual Studio](#) and the [Live Unit Testing FAQ](#).

Better support for multiple target frameworks

If you're building a project for multiple target frameworks, you can now select the target platform from the top-level menu. In the following figure, a project named SCD1 targets 64-bit macOS X 10.11 (`osx.10.11-x64`) and 64-bit Windows 10/Windows Server 2016 (`win10-x64`). You can select the target framework before selecting the project button, in this case to run a debug build.



Side-by-side support for .NET Core SDKs

You can now install the .NET Core SDK independently of Visual Studio. This makes it possible for a single version of Visual Studio to build projects that target different versions of .NET Core. Previously, Visual Studio and the .NET Core SDK were tightly coupled; a particular version of the SDK accompanied a particular version of Visual Studio.

Documentation improvements

.NET Application Architecture

[.NET Application Architecture](#) gives you access to a set of e-books that provide guidance, best practices, and sample applications when using .NET to build:

- [Microservices and Docker containers](#)
- [Web applications with ASP.NET](#)
- [Mobile applications with Xamarin](#)
- [Applications that are deployed to the Cloud with Azure](#)

See also

- [What's new in ASP.NET Core 2.0](#)

What's new in .NET Standard

9/20/2022 • 3 minutes to read • [Edit Online](#)

.NET Standard is a formal specification that defines a versioned set of APIs that must be available on .NET implementations that comply with that version of the standard. .NET Standard is targeted at library developers. A library that targets a .NET Standard version can be used on any .NET or Xamarin implementation that supports that version of the standard.

.NET Standard is included with the .NET SDK. It's also included with Visual Studio if you select the .NET workload.

.NET Standard 2.1 is the last version of .NET Standard that will be released. For more information, see [.NET 5+ and .NET Standard](#).

Supported .NET implementations

.NET Standard 2.1 is supported by the following .NET implementations:

- .NET Core 3.0 or later (including .NET 5 and later)
- Mono 6.4 or later
- Xamarin.iOS 12.16 or later
- Xamarin.Android 10.0 or later

.NET Standard 2.0 is supported by the following .NET implementations:

- .NET Core 2.0 or later (including .NET 5 and later)
- .NET Framework 4.6.1 or later
- Mono 5.4 or later
- Xamarin.iOS 10.14 or later
- Xamarin.Mac 3.8 or later
- Xamarin.Android 8.0 or later
- Universal Windows Platform 10.0.16299 or later

What's new in .NET Standard 2.1

.NET Standard 2.1 adds many APIs to the standard. Some of them are new APIs, and others are existing APIs that help to converge the .NET implementations even further. For a list of the APIs that have been added to .NET Standard 2.1, see [.NET Standard 2.1 vs 2.0](#).

For more information, see the [Announcing .NET Standard 2.1](#) blog post.

What's new in .NET Standard 2.0

.NET Standard 2.0 includes the following new features.

A vastly expanded set of APIs

Through version 1.6, .NET Standard included a comparatively small subset of APIs. Among those excluded were many APIs that were commonly used in .NET Framework or Xamarin. This complicates development, since it requires that developers find suitable replacements for familiar APIs when they develop applications and libraries that target multiple .NET implementations. .NET Standard 2.0 addresses this limitation by adding over 20,000 more APIs than were available in .NET Standard 1.6, the previous version of the standard. For a list of the APIs that have been added to .NET Standard 2.0, see [.NET Standard 2.0 vs 1.6](#).

Some of the additions to the [System](#) namespace in .NET Standard 2.0 include:

- Support for the [AppDomain](#) class.
- Better support for working with arrays from additional members in the [Array](#) class.
- Better support for working with attributes from additional members in the [Attribute](#) class.
- Better calendar support and additional formatting options for [DateTime](#) values.
- Additional [Decimal](#) rounding functionality.
- Additional functionality in the [Environment](#) class.
- Enhanced control over the garbage collector through the [GC](#) class.
- Enhanced support for string comparison, enumeration, and normalization in the [String](#) class.
- Support for daylight saving adjustments and transition times in the [TimeZoneInfo.AdjustmentRule](#) and [TimeZoneInfo.TransitionTime](#) classes.
- Significantly enhanced functionality in the [Type](#) class.
- Better support for deserialization of exception objects by adding an exception constructor with [SerializationInfo](#) and [StreamingContext](#) parameters.

Support for .NET Framework libraries

Many libraries target .NET Framework rather than .NET Standard. However, most of the calls in those libraries are to APIs that are included in .NET Standard 2.0. Starting with .NET Standard 2.0, you can access .NET Framework libraries from a .NET Standard library by using a [compatibility shim](#). This compatibility layer is transparent to developers; you don't have to do anything to take advantage of .NET Framework libraries.

The single requirement is that the APIs called by the .NET Framework class library must be included in .NET Standard 2.0.

Support for Visual Basic

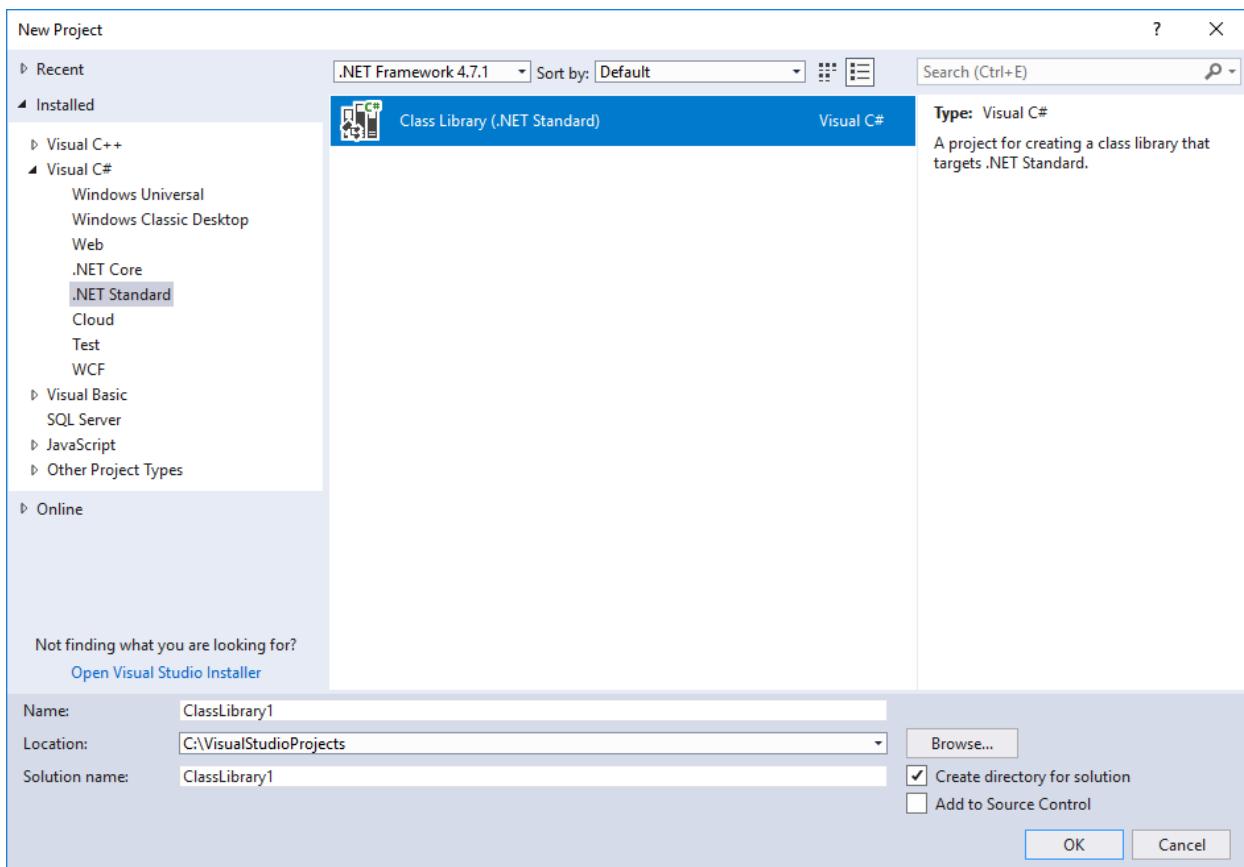
You can now develop .NET Standard libraries in Visual Basic. Visual Studio 2019 and Visual Studio 2017 version 15.3 or later with the .NET Core workload installed include a .NET Standard Class Library template. For Visual Basic developers who use other development tools and environments, you can use the [dotnet new](#) command to create a .NET Standard Library project. For more information, see the [Tooling support for .NET Standard libraries](#).

Tooling support for .NET Standard libraries

With the release of .NET Core 2.0 and .NET Standard 2.0, both Visual Studio 2017 and the [.NET CLI](#) include tooling support for creating .NET Standard libraries.

If you install Visual Studio with the [.NET Core cross-platform development](#) workload, you can create a .NET Standard 2.0 library project by using a project template, as the following figure shows:

- [C#](#)
- [Visual Basic](#)



If you're using the .NET CLI, the following `dotnet new` command creates a class library project that targets .NET Standard 2.0:

```
dotnet new classlib
```

See also

- [.NET Standard](#)
- [Introducing .NET Standard](#)
- [Download the .NET SDK](#)

Tools and diagnostics in .NET

9/20/2022 • 2 minutes to read • [Edit Online](#)

In this article, you'll learn about the various tools available to .NET developers. With .NET, you have a robust software development kit (SDK) that includes a command-line interface (CLI). The .NET CLI enables many of the features throughout the .NET-ready integrated development environments (IDEs). This article also provides resources to productivity capabilities, such as .NET CLI tools for diagnosing performance issues, memory leaks, high CPU, deadlocks, and tooling support for code analysis.

.NET SDK

The .NET SDK includes both the .NET Runtime and the .NET CLI. You can download the [.NET SDK](#) for Windows, Linux, macOS, or Docker. For more information, see [.NET SDK overview](#).

.NET CLI

The .NET CLI is a cross-platform toolchain for developing, building, running, and publishing .NET applications. The .NET CLI is included with the .NET SDK. For more information, see [.NET CLI overview](#).

IDEs

You can write .NET applications in [Visual Studio Code](#), [Visual Studio](#), or [Visual Studio for Mac](#).

Additional tools

In addition to the more common tools, .NET also provides tools for specific scenarios. Some of the use cases include uninstalling the .NET SDK or the .NET Runtime, retrieving Windows Communication Foundation (WCF) metadata, generating proxy source code, and serializing XML. For more information, see [.NET additional tools overview](#).

Diagnostics and instrumentation

As a .NET developer, you can make use of common performance diagnostic tools to monitor app performance, profile apps with tracing, collect performance metrics, and analyze dump files. You collect performance metrics with event counters, and use profiling tools to gain insights into how apps perform. For more information, see [.NET diagnostics tools](#).

Code analysis

The .NET compiler platform (Roslyn) analyzers inspect your C# or Visual Basic code for code quality and code style issues. For more information, see [.NET source code analysis overview](#).

Package Validation

The .NET SDK allows library developers to validate that their packages are consistent and well formed. For more information, see [.NET SDK package validation](#).

What is the .NET SDK?

9/20/2022 • 2 minutes to read • [Edit Online](#)

The .NET SDK is a set of libraries and tools that allow developers to create .NET applications and libraries. It contains the following components that are used to build and run applications:

- The [.NET CLI](#).
- The [.NET runtime and libraries](#).
- The `dotnet` driver.

How to install the .NET SDK

As with any tooling, the first thing is to get the tools on your machine. Depending on your scenario, you can install the SDK using one of the following methods:

- Use the native installers.
- Use the installation shell script.

The native installers are primarily meant for developers' machines. The SDK is distributed using each supported platform's native install mechanism, such as DEB packages on Ubuntu or MSI bundles on Windows. These installers install and set up the environment as needed for the user to use the SDK immediately after the install. However, they also require administrative privileges on the machine. You can find the SDK to install on the [.NET downloads](#) page.

Install scripts, on the other hand, don't require administrative privileges. However, they also don't install any prerequisites on the machine; you need to install all of the prerequisites manually. The scripts are meant mostly for setting up build servers or when you wish to install the tools without admin privileges (do note the prerequisites caveat above). You can find more information in the [install script reference](#) article. If you're interested in how to set up the SDK on your CI build server, see [Use the .NET SDK in Continuous Integration \(CI\) environments](#).

By default, the SDK installs in a "side-by-side" (SxS) manner, which means multiple versions can coexist at any given time on a single machine. For information about how the version gets picked when you're running CLI commands, see [Select the .NET version to use](#).

See also

- [.NET downloads](#)
- [.NET CLI overview](#)
- [.NET versioning overview](#)
- [How to remove the .NET runtime and SDK](#)
- [Select the .NET version to use](#)

.NET environment variables

9/20/2022 • 11 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 3.1 SDK and later versions

In this article, you'll learn about the environment variables used by .NET SDK, .NET CLI, and .NET runtime. Some environment variables are used by the .NET runtime, while others are only used by the .NET SDK and .NET CLI. Some environment variables are used by all.

.NET runtime environment variables

`DOTNET_SYSTEM_NET_HTTP_*`

There are several global HTTP environment variable settings:

- `DOTNET_SYSTEM_NET_HTTP_ENABLEACTIVITYPROPAGATION`
 - Indicates whether or not to enable activity propagation of the diagnostic handler for global HTTP settings.
- `DOTNET_SYSTEM_NET_HTTP_SOCKETSHTTPHANDLER_HTTP2SUPPORT`
 - When set to `false` or `0`, disables HTTP/2 support, which is enabled by default.
- `DOTNET_SYSTEM_NET_HTTP_SOCKETSHTTPHANDLER_HTTP3SUPPORT`
 - When set to `true` or `1`, enables HTTP/3 support, which is disabled by default.
- `DOTNET_SYSTEM_NET_HTTP_SOCKETSHTTPHANDLER_HTTP2FLOWCONTROL_DISABLEDYYNAMICWINDOWSIZING`
 - When set to `false` or `0`, overrides the default and disables the HTTP/2 dynamic window scaling algorithm.
- `DOTNET_SYSTEM_NET_HTTP_SOCKETSHTTPHANDLER_FLOWCONTROL_MAXSTREAMWINDOWSIZE`
 - Defaults to 16 MB. When overridden, the maximum size of the HTTP/2 stream receive window cannot be less than 65,535.
- `DOTNET_SYSTEM_NET_HTTP_SOCKETSHTTPHANDLER_FLOWCONTROL_STREAMWINDOWSCALETHRESHOLDMULTIPLIER`
 - Defaults to 1.0. When overridden, higher values result in a shorter window but slower downloads.
Can't be less than 0.

`DOTNET_SYSTEM_GLOBALIZATION_*`

- `DOTNET_SYSTEM_GLOBALIZATION_INVARIANT` : See [set invariant mode](#).
- `DOTNET_SYSTEM_GLOBALIZATION_PREDEFINED_CULTURES_ONLY` : Specifies whether to load only predefined cultures.
- `DOTNET_SYSTEM_GLOBALIZATION_APPLOCALICU` : Indicates whether to use the app-local International Components of Unicode (ICU). For more information, see [App-local ICU](#).

Set invariant mode

Applications can enable the invariant mode in any of the following ways:

1. In the project file:

```
<PropertyGroup>
    <InvariantGlobalization>true</InvariantGlobalization>
</PropertyGroup>
```

2. In the `runtimedataconfig.json` file:

```
{  
    "runtimeOptions": {  
        "configProperties": {  
            "System.Globalization.Invariant": true  
        }  
    }  
}
```

3. By setting environment variable value `DOTNET_SYSTEM_GLOBALIZATION_INVARIANT` to `true` or `1`.

IMPORTANT

A value set in the project file or *runtimeconfig.json* has a higher priority than the environment variable.

For more information, see [.NET Globalization Invariant Mode](#).

`DOTNET_SYSTEM_GLOBALIZATION_USENLS`

This applies to Windows only. For globalization to use National Language Support (NLS), set

`DOTNET_SYSTEM_GLOBALIZATION_USENLS` to either `true` or `1`. To not use it, set `DOTNET_SYSTEM_GLOBALIZATION_USENLS` to either `false` or `0`.

`DOTNET_SYSTEM_NET_SOCKETS_*`

This section focuses on two `System.Net.Sockets` environment variables:

- `DOTNET_SYSTEM_NET_SOCKETS_INLINE_COMPLETIONS`
- `DOTNET_SYSTEM_NET_SOCKETS_THREAD_COUNT`

Socket continuations are dispatched to the `System.Threading.ThreadPool` from the event thread. This avoids continuations blocking the event handling. To allow continuations to run directly on the event thread, set

`DOTNET_SYSTEM_NET_SOCKETS_INLINE_COMPLETIONS` to `1`. It's disabled by default.

NOTE

This setting can make performance worse if there is expensive work that will end up holding onto the IO thread for longer than needed. Test to make sure this setting helps performance.

Using TechEmpower benchmarks that generate a lot of small socket reads and writes under a very high load, a single socket engine is capable of keeping busy up to thirty x64 and eight Arm64 CPU cores. The vast majority of real-life scenarios will never generate such a huge load (hundreds of thousands of requests per second), and having a single producer is almost always enough. However, to be sure that extreme loads can be handled, you can use `DOTNET_SYSTEM_NET_SOCKETS_THREAD_COUNT` to override the calculated value. When not overridden, the following value is used:

- When `DOTNET_SYSTEM_NET_SOCKETS_INLINE_COMPLETIONS` is `1`, the `Environment.ProcessorCount` value is used.
- When `DOTNET_SYSTEM_NET_SOCKETS_INLINE_COMPLETIONS` is not `1`, `RuntimelInformation.ProcessArchitecture` is evaluated:
 - When Arm or Arm64 the cores per engine value is set to `8`, otherwise `30`.
- Using the determined cores per engine, the maximum value of either `1` or `Environment.ProcessorCount` over the cores per engine.

`DOTNET_SYSTEM_NET_DISABLEIPV6`

Helps determine whether or not Internet Protocol version 6 (IPv6) is disabled. When set to either `true` or `1`, IPv6 is disabled unless otherwise specified in the `System.AppContext`.

`DOTNET_SYSTEM_NET_HTTP_USESOCKETSHTTPHANDLER`

You can use one of the following mechanisms to configure a process to use the older `HttpClientHandler`:

From code, use the `ApplicationContext` class:

```
ApplicationContext.SetSwitch("System.Net.Http.UseSocketsHttpHandler", false);
```

The `ApplicationContext` switch can also be set by a config file. For more information configuring switches, see [ApplicationContext for library consumers](#).

The same can be achieved via the environment variable `DOTNET_SYSTEM_NET_HTTP_USESOCKETSHTTPHANDLER`. To opt-out, set the value to either `false` or `0`.

NOTE

Starting in .NET 5, this setting to use `HttpClientHandler` is no longer available.

`DOTNET_Jit*` and `DOTNET_GC*`

There are two stressing-related features for the JIT and JIT-generated GC information: JIT Stress and GC Hole Stress. These features provide a way during development to discover edge cases and more "real world" scenarios without having to develop complex applications. The following environment variables are available:

- `DOTNET_JitStress`
- `DOTNET_JitStressModeNamesOnly`
- `DOTNET_GCStress`

JIT stress

Enabling JIT Stress can be done in several ways. Set `DOTNET_JitStress` to a non-zero integer value to generate varying levels of JIT optimizations based on a hash of the method's name. To apply all optimizations set `DOTNET_JitStress=2`, for example. Another way to enable JIT Stress is by setting `DOTNET_JitStressModeNamesOnly=1` and then requesting the stress modes, space-delimited, in the `DOTNET_JitStressModeNames` variable.

As an example, consider:

```
DOTNET_JitStressModeNames=STRESS_USE_CMOV STRESS_64RSLT_MUL STRESS_LCL_FLDS
```

GC Hole stress

Enabling GC Hole Stress causes GCs to always occur in specific locations and that helps to track down GC holes. GC Hole Stress can be enabled using the `DOTNET_GCStress` environment variable.

For more information, see [Investigating JIT and GC Hole stress](#).

JIT memory barriers

The code generator for Arm64 allows all `MemoryBarriers` instructions to be removed by setting `DOTNET_JitNoMemoryBarriers` to `1`.

`DOTNET_RUNNING_IN_CONTAINER` and `DOTNET_RUNNING_IN_CONTAINERS`

The official .NET images (Windows and Linux) set the well-known environment variables:

- `DOTNET_RUNNING_IN_CONTAINER`
- `DOTNET_RUNNING_IN_CONTAINERS`

These values are used to determine when your ASP.NET Core workloads are running in the context of a

container.

`DOTNET_SYSTEM_CONSOLE_ALLOW_ANSI_COLOR_REDIRECTION`

When `Console.IsOutputRedirected` is `true`, you can emit ANSI color code by setting

`DOTNET_SYSTEM_CONSOLE_ALLOW_ANSI_COLOR_REDIRECTION` to either `1` or `true`.

`DOTNET_SYSTEM_DIAGNOSTICS` and related variables

- `DOTNET_SYSTEM_DIAGNOSTICS_DEFAULTACTIVITYIDFORMATISHIERARCHIAL`: When `1` or `true`, the default *Activity Id* format is hierarchical.
- `DOTNET_SYSTEM_RUNTIME_CACHING_TRACING`: When running as Debug, tracing can be enabled when this is `true`.

`DOTNET_DiagnosticPorts`

Configures alternate endpoints where diagnostic tools can communicate with the .NET runtime. See the [Diagnostic Port documentation](#) for more information.

`DOTNET_DefaultDiagnosticPortSuspend`

Configures the runtime to pause during startup and wait for the *Diagnostics IPC ResumeStartup* command from the specified diagnostic port when set to 1. Defaults to 0. See the [Diagnostic Port documentation](#) for more information.

`DOTNET_EnableDiagnostics`

When set to `1`, enables debugging, profiling, and other diagnostics via the [Diagnostic Port](#). Defaults to 1.

EventPipe variables

See [EventPipe environment variables](#) for more information.

- `DOTNET_EnableEventPipe`: When set to `1`, enables tracing via EventPipe.
- `DOTNET_EventPipeOutputPath`: The output path where the trace will be written.
- `DOTNET_EventPipeOutputStreaming`: When set to `1`, enables streaming to the output file while the app is running. By default trace information is accumulated in a circular buffer and the contents are written at app shutdown.

.NET SDK and CLI environment variables

`DOTNET_ROOT`, `DOTNET_ROOT(x86)`

Specifies the location of the .NET runtimes, if they are not installed in the default location. The default location on Windows is `C:\Program Files\dotnet`. The default location on Linux and macOS is `/usr/local/share/dotnet`. This environment variable is used only when running apps via generated executables (apphosts). `DOTNET_ROOT(x86)` is used instead when running a 32-bit executable on a 64-bit OS.

`NUGET_PACKAGES`

The global packages folder. If not set, it defaults to `~/.nuget/packages` on Unix or `%userprofile%\.nuget\packages` on Windows.

`DOTNET_SERVICING`

Specifies the location of the servicing index to use by the shared host when loading the runtime.

`DOTNET_NOLOGO`

Specifies whether .NET welcome and telemetry messages are displayed on the first run. Set to `true` to mute these messages (values `true`, `1`, or `yes` accepted) or set to `false` to allow them (values `false`, `0`, or `no` accepted). If not set, the default is `false` and the messages will be displayed on the first run. This flag does not affect telemetry (see `DOTNET_CLI_TELEMETRY_OPTOUT` for opting out of sending telemetry).

`DOTNET_CLI_PERF_LOG`

Specifies whether performance details about the current CLI session are logged. Enabled when set to `1`, `true`, or `yes`. This is disabled by default.

`DOTNET_GENERATE_APNET_CERTIFICATE`

Specifies whether to generate an ASP.NET Core certificate. The default value is `true`, but this can be overridden by setting this environment variable to either `0`, `false`, or `no`.

`DOTNET_ADD_GLOBAL_TOOLS_TO_PATH`

Specifies whether to add global tools to the `PATH` environment variable. The default is `true`. To not add global tools to the path, set to `0`, `false`, or `no`.

`DOTNET_CLI_TELEMETRY_OPTOUT`

Specifies whether data about the .NET tools usage is collected and sent to Microsoft. Set to `true` to opt-out of the telemetry feature (values `true`, `1`, or `yes` accepted). Otherwise, set to `false` to opt into the telemetry features (values `false`, `0`, or `no` accepted). If not set, the default is `false` and the telemetry feature is active.

`DOTNET_SKIP_FIRST_TIME_EXPERIENCE`

If `DOTNET_SKIP_FIRST_TIME_EXPERIENCE` is set to `true`, the `NuGetFallbackFolder` won't be expanded to disk and a shorter welcome message and telemetry notice will be shown.

`DOTNET_MULTILEVEL_LOOKUP`

Specifies whether the .NET runtime, shared framework, or SDK are resolved from the global location. If not set, it defaults to 1 (logical `true`). Set the value to 0 (logical `false`) to not resolve from the global location and have isolated .NET installations. For more information about multi-level lookup, see [Multi-level SharedFX Lookup](#).

NOTE

This environment variable only applies to applications that target .NET 6 and earlier versions. Starting in .NET 7, .NET only looks for frameworks in one location. For more information, see [Multi-level lookup is disabled](#).

`DOTNET_ROLL_FORWARD`

Determines roll forward behavior. For more information, see the `--roll-forward` option for the `dotnet` command.

`DOTNET_ROLL_FORWARD_TO_PRERELEASE`

If set to `1` (enabled), enables rolling forward to a pre-release version from a release version. By default (`0` - disabled), when a release version of .NET runtime is requested, roll-forward will only consider installed release versions.

For more information, see the `--roll-forward` option for the `dotnet` command

`DOTNET_ROLL_FORWARD_ON_NO_CANDIDATE_FX`

Disables minor version roll forward, if set to `0`. This setting is superseded in .NET Core 3.0 by `DOTNET_ROLL_FORWARD`. The new settings should be used instead.

`DOTNET_CLI_UI_LANGUAGE`

Sets the language of the CLI UI using a locale value such as `en-us`. The supported values are the same as for Visual Studio. For more information, see the section on changing the installer language in the [Visual Studio installation documentation](#). The .NET resource manager rules apply, so you don't have to pick an exact match—you can also pick descendants in the `CultureInfo` tree. For example, if you set it to `fr-CA`, the CLI will find and use the `fr` translations. If you set it to a language that is not supported, the CLI falls back to English.

`DOTNET_DISABLE_GUI_ERRORS`

For GUI-enabled generated executables - disables dialog popup, which normally shows for certain classes of errors. It only writes to `stderr` and exits in those cases.

`DOTNET_ADDITIONAL_DEPS`

Equivalent to CLI option `--additional-deps`.

`DOTNET_RUNTIME_ID`

Overrides the detected RID.

`DOTNET_SHARED_STORE`

Location of the "shared store" which assembly resolution falls back to in some cases.

`DOTNET_STARTUP_HOOKS`

List of assemblies to load and execute startup hooks from.

`DOTNET_BUNDLE_EXTRACT_BASE_DIR`

Specifies a directory to which a single-file application is extracted before it is executed.

For more information, see [Single-file executables](#).

`DOTNET_CLI_CONTEXT_*`

- `DOTNET_CLI_CONTEXT_VERBOSE` : To enable a verbose context, set to `true`.
- `DOTNET_CLI_CONTEXT_ANSI_PASS_THRU` : To enable an ANSI pass-through, set to `true`.

`DOTNET_CLI_WORKLOAD_UPDATE_NOTIFY_DISABLE`

Disables background download of advertising manifests for workloads. Default is `false` - not disabled. If set to `true`, downloading is disabled. For more information, see [Advertising manifests](#).

`DOTNET_CLI_WORKLOAD_UPDATE_NOTIFY_INTERVAL_HOURS`

Specifies the minimum number of hours between background downloads of advertising manifests for workloads. Default is `24` - no more frequently than once a day. For more information, see [Advertising manifests](#).

`COREHOST_TRACE`

Controls diagnostics tracing from the hosting components, such as `dotnet.exe`, `hostfxr`, and `hostpolicy`.

- `COREHOST_TRACE=[0/1]` - default is `0` - tracing disabled. If set to `1`, diagnostics tracing is enabled.
- `COREHOST_TRACEFILE=<file path>` - has an effect only if tracing is enabled by setting `COREHOST_TRACE=1`. When set, the tracing information is written to the specified file; otherwise, the trace information is written to `stderr`.
- `COREHOST_TRACE_VERBOSITY=[1/2/3/4]` - default is `4`. The setting is used only when tracing is enabled via `COREHOST_TRACE=1`.
 - `4` - all tracing information is written
 - `3` - only informational, warning, and error messages are written
 - `2` - only warning and error messages are written
 - `1` - only error messages are written

The typical way to get detailed trace information about application startup is to set `COREHOST_TRACE=1` and `COREHOST_TRACEFILE=host_trace.txt` and then run the application. A new file `host_trace.txt` will be created in the current directory with the detailed information.

SuppressNETCoreSdkPreviewMessage

If set to `true`, invoking `dotnet` won't produce a warning when a preview SDK is being used.

Configure MSBuild in the .NET CLI

To execute MSBuild out-of-process, set the `DOTNET_CLI_RUN_MSBUILD_OUTOFPROC` environment variable to either `1`, `true`, or `yes`. By default, MSBuild will execute in-proc. To force MSBuild to use an external working node long-living process for building projects, set `DOTNET_CLI_USE_MSBUILDNOINPROCNODE` to `1`, `true`, or `yes`. This will set the `MSBUILDNOINPROCNODE` environment variable to `1`, which is referred to as *MSBuild Server V1*, as the entry process forwards most of the work to it.

DOTNET_MSBUILD_SDK_RESOLVER_*

These are overrides that are used to force the resolved SDK tasks and targets to come from a given base directory and report a given version to MSBuild, which may be `null` if unknown. One key use case for this is to test SDK tasks and targets without deploying them by using the .NET Core SDK.

- `DOTNET_MSBUILD_SDK_RESOLVER_SDKS_DIR` : Overrides the .NET SDK directory.
- `DOTNET_MSBUILD_SDK_RESOLVER_SDKS_VER` : Overrides the .NET SDK version.
- `DOTNET_MSBUILD_SDK_RESOLVER_CLI_DIR` : Overrides the `dotnet.exe` directory path.

DOTNET_NEW_PREFERRED_LANG

Configures the default programming language for the `dotnet new` command when the `-lang|--language` switch is omitted. The default value is `c#`. Valid values are `c#`, `F#`, or `vb`. For more information, see [dotnet new](#).

dotnet watch environment variables

For information about `dotnet watch` settings that are available as environment variables, see [dotnet watch environment variables](#).

See also

- [dotnet command](#)
- [Runtime Configuration Files](#)
- [.NET runtime configuration settings](#)

dotnet-install scripts reference

9/20/2022 • 7 minutes to read • [Edit Online](#)

Name

`dotnet-install.ps1` | `dotnet-install.sh` - Script used to install the .NET SDK and the shared runtime.

Synopsis

Windows:

```
dotnet-install.ps1 [-Architecture <ARCHITECTURE>] [-AzureFeed]
  [-Channel <CHANNEL>] [-DryRun] [-FeedCredential]
  [-InstallDir <DIRECTORY>] [-JsonFile <JSONFILE>]
  [-NoCdn] [-NoPath] [-ProxyAddress] [-ProxyBypassList <LIST_OF_URLS>]
  [-ProxyUseDefaultCredentials] [-Quality <QUALITY>] [-Runtime <RUNTIME>]
  [-SkipNonVersionedFiles] [-UncachedFeed] [-Verbose]
  [-Version <VERSION>]

Get-Help ./dotnet-install.ps1
```

Linux/macOS:

```
dotnet-install.sh [--architecture <ARCHITECTURE>] [--azure-feed]
  [--channel <CHANNEL>] [--dry-run] [--feed-credential]
  [--install-dir <DIRECTORY>] [--jsonfile <JSONFILE>]
  [--no-cdn] [--no-path] [--quality <QUALITY>]
  [--runtime <RUNTIME>] [--runtime-id <RID>]
  [--skip-non-versioned-files] [--uncached-feed] [--verbose]
  [--version <VERSION>]

dotnet-install.sh --help
```

The bash script also reads PowerShell switches, so you can use PowerShell switches with the script on Linux/macOS systems.

Description

The `dotnet-install` scripts perform a non-admin installation of the .NET SDK, which includes the .NET CLI and the shared runtime. There are two scripts:

- A PowerShell script that works on Windows.
- A bash script that works on Linux/macOS.

NOTE

.NET collects telemetry data. To learn more and how to opt out, see [.NET SDK telemetry](#).

Purpose

The intended use of the scripts is for Continuous Integration (CI) scenarios, where:

- The SDK needs to be installed without user interaction and without admin rights.

- The SDK installation doesn't need to persist across multiple CI runs.

The typical sequence of events:

- CI is triggered.
- CI installs the SDK using one of these scripts.
- CI finishes its work and clears temporary data including the SDK installation.

To set up a development environment or to run apps, use the installers rather than these scripts.

Recommended version

We recommend that you use the stable version of the scripts:

- Bash (Linux/macOS): <https://dot.net/v1/dotnet-install.sh>
- PowerShell (Windows): <https://dot.net/v1/dotnet-install.ps1>

Script behavior

Both scripts have the same behavior. They download the ZIP/tarball file from the CLI build drops and proceed to install it in either the default location or in a location specified by `-InstallDir|--install-dir`.

By default, the installation scripts download the SDK and install it. If you wish to only obtain the shared runtime, specify the `-Runtime|--runtime` argument.

By default, the script adds the install location to the \$PATH for the current session. Override this default behavior by specifying the `-NoPath|--no-path` argument. The script doesn't set the `DOTNET_ROOT` environment variable.

Before running the script, install the required [dependencies](#).

You can install a specific version using the `-Version|--version` argument. The version must be specified as a three-part version number, such as `2.1.0`. If the version isn't specified, the script installs the `latest` version.

The install scripts do not update the registry on Windows. They just download the zipped binaries and copy them to a folder. If you want registry key values to be updated, use the .NET installers.

Options

- `-Architecture|--architecture <ARCHITECTURE>`

Architecture of the .NET binaries to install. Possible values are `<auto>`, `amd64`, `x64`, `x86`, `arm64`, `arm`, and `s390x`. The default value is `<auto>`, which represents the currently running OS architecture.

- `-AzureFeed|--azure-feed`

For internal use only. Allows using a different storage to download SDK archives from. This parameter is only used if `--no-cdn` is false. The default is <https://dotnetcli.azureedge.net/dotnet>.

- `-Channel|--channel <CHANNEL>`

Specifies the source channel for the installation. The possible values are:

- `Current` - Most current release.
- `LTS` - Long-Term Support channel (most current supported release).
- Two-part version in A.B format, representing a specific release (for example, `3.1` or `6.0`).
- Three-part version in A.B.Cxx format, representing a specific SDK release (for example, `6.0.1xx` or `6.0.2xx`). Available since the 5.0 release.

The `version` parameter overrides the `channel` parameter when any version other than `latest` is used.

The default value is `LTS`. For more information on .NET support channels, see the [.NET Support Policy](#)

page.

- `-DryRun|--dry-run`

If set, the script won't perform the installation. Instead, it displays what command line to use to consistently install the currently requested version of the .NET CLI. For example, if you specify version `latest`, it displays a link with the specific version so that this command can be used deterministically in a build script. It also displays the binary's location if you prefer to install or download it yourself.

- `-FeedCredential|--feed-credential`

Used as a query string to append to the Azure feed. It allows changing the URL to use non-public blob storage accounts.

- `--help`

Prints out help for the script. Applies only to bash script. For PowerShell, use

```
Get-Help ./dotnet-install.ps1 .
```

- `-InstallDir|--install-dir <DIRECTORY>`

Specifies the installation path. The directory is created if it doesn't exist. The default value is `%LocalAppData%\Microsoft\dotnet` on Windows and `$HOME/.dotnet` on Linux/macOS. Binaries are placed directly in this directory.

- `-JsonFile|--jsonfile <JSONFILE>`

Specifies a path to a [global.json](#) file that will be used to determine the SDK version. The *global.json* file must have a value for `sdk:version`.

- `-NoCdn|--no-cdn`

Disables downloading from the [Azure Content Delivery Network \(CDN\)](#) and uses the uncached feed directly.

- `-NoPath|--no-path`

If set, the installation folder isn't exported to the path for the current session. By default, the script modifies the PATH, which makes the .NET CLI available immediately after install.

- `-ProxyAddress`

If set, the installer uses the proxy when making web requests. (Only valid for Windows.)

- `-ProxyBypassList <LIST_OF_URLS>`

If set with `ProxyAddress`, provides a list of comma-separated urls that will bypass the proxy. (Only valid for Windows.)

- `ProxyUseDefaultCredentials`

If set, the installer uses the credentials of the current user when using proxy address. (Only valid for Windows.)

- `-Quality|--quality <QUALITY>`

Downloads the latest build of the specified quality in the channel. The possible values are: `daily`, `signed`, `validated`, `preview`, `GA`. Works only in combination with `channel`. Not applicable for current and LTS channels and will be ignored if one of those channels is used.

For an SDK installation, use `channel` in `A.B` or `A.B.Cxx` format. For a runtime installation, use `channel`

in `A.B` format.

Don't use both `version` and `quality` parameters. When `quality` is specified, the script determines the proper version on its own.

Available since the 5.0 release.

- `-Runtime|--runtime <RUNTIME>`

Installs just the shared runtime, not the entire SDK. The possible values are:

- `dotnet` - the `Microsoft.NETCore.App` shared runtime.
- `aspnetcore` - the `Microsoft.AspNetCore.App` shared runtime.
- `windowsdesktop` - the `Microsoft.WindowsDesktop.App` shared runtime.

- `--os <OPERATING_SYSTEM>`

Specifies the operating system for which the tools are being installed. Possible values are: `osx`, `linux`, `linux-musl`, `freebsd`.

The parameter is optional and should only be used when it's required to override the operating system that is detected by the script.

- `-SharedRuntime|--shared-runtime`

NOTE

This parameter is obsolete and may be removed in a future version of the script. The recommended alternative is the `-Runtime|--runtime` option.

Installs just the shared runtime bits, not the entire SDK. This option is equivalent to specifying

`-Runtime|--runtime dotnet`.

- `-SkipNonVersionedFiles|--skip-non-versioned-files`

Skips installing non-versioned files, such as `dotnet.exe`, if they already exist.

- `-UncachedFeed|--uncached-feed`

For internal use only. Allows using a different storage to download SDK archives from. This parameter is only used if `--no-cdn` is true.

- `-Verbose|--verbose`

Displays diagnostics information.

- `-Version|--version <VERSION>`

Represents a specific build version. The possible values are:

- `Latest` - Latest build on the channel (used with the `-Channel` option).
- Three-part version in X.Y.Z format representing a specific build version; supersedes the `-channel` option. For example: `2.0.0-preview2-006120`.

If not specified, `-Version` defaults to `latest`.

Examples

- Install the latest long-term supported (LTS) version to the default location:

Windows:

```
./dotnet-install.ps1 -Channel LTS
```

macOS/Linux:

```
./dotnet-install.sh --channel LTS
```

- Install the latest preview version of the 6.0.1xx SDK to the specified location:

Windows:

```
./dotnet-install.ps1 -Channel 6.0.1xx -Quality preview -InstallDir C:\cli
```

macOS/Linux:

```
./dotnet-install.sh --channel 6.0.1xx --quality preview --install-dir ~/cli
```

- Install the 6.0.0 version of the shared runtime:

Windows:

```
./dotnet-install.ps1 -Runtime dotnet -Version 6.0.0
```

macOS/Linux:

```
./dotnet-install.sh --runtime dotnet --version 6.0.0
```

- Obtain script and install the 6.0.2 version behind a corporate proxy (Windows only):

```
Invoke-WebRequest 'https://dot.net/v1/dotnet-install.ps1' -Proxy $env:HTTP_PROXY -  
ProxyUseDefaultCredentials -OutFile 'dotnet-install.ps1';  
./dotnet-install.ps1 -InstallDir '~/.dotnet' -Version '6.0.2' -Runtime 'dotnet' -ProxyAddress  
$env:HTTP_PROXY -ProxyUseDefaultCredentials;
```

- Obtain script and install .NET CLI one-liner examples:

Windows:

```
# Run a separate PowerShell process because the script calls exit, so it will end the current  
PowerShell session.  
&powershell -NoProfile -ExecutionPolicy unrestricted -Command "  
[Net.ServicePointManager]::SecurityProtocol = [Net.SecurityProtocolType]::Tls12; &  
([scriptblock]::Create((Invoke-WebRequest -UseBasicParsing 'https://dot.net/v1/dotnet-install.ps1')))  
<additional install-script args>"
```

macOS/Linux:

```
curl -sSL https://dot.net/v1/dotnet-install.sh | bash /dev/stdin <additional install-script args>
```

Set environment variables

Manually installing .NET doesn't add the environment variables system-wide, and generally only works for the session in which .NET was installed. There are two environment variables you should set for your operating system:

- **DOTNET_ROOT**

This variable is set to the folder .NET was installed to, such as `$HOME/.dotnet` for Linux and macOS, and `$HOME\.dotnet` in PowerShell for Windows.

- **PATH**

This variable should include both the `DOTNET_ROOT` folder and the user's `.dotnet/tools` folder. Generally this is `$HOME/.dotnet/tools` on Linux and macOS, and `$HOME\.dotnet\tools` in PowerShell on Windows.

TIP

For Linux and macOS, use the `echo` command to set the variables in your shell profile, such as `.bashrc`.

```
echo 'export DOTNET_ROOT=$HOME/.dotnet' >> ~/.bashrc
echo 'export PATH=$PATH:$DOTNET_ROOT:$DOTNET_ROOT/tools' >> ~/.bashrc
```

See also

- [.NET releases](#)
- [.NET Runtime and SDK download archive](#)

global.json overview

9/20/2022 • 8 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 3.1 SDK and later versions

The *global.json* file allows you to define which .NET SDK version is used when you run .NET CLI commands. Selecting the .NET SDK version is independent from specifying the runtime version a project targets. The .NET SDK version indicates which version of the .NET CLI is used. This article explains how to select the SDK version by using *global.json*.

If you always want to use the latest SDK version that is installed on your machine, no *global.json* file is needed. In CI (continuous integration) scenarios, however, you typically want to specify an acceptable range for the SDK version that is used. The *global.json* file has a `rollForward` feature that provides flexible ways to specify an acceptable range of versions. For example, the following *global.json* file selects 6.0.300 or any later [feature band](#) or [patch](#) for 6.0 that is installed on the machine:

```
{  
  "sdk": {  
    "version": "6.0.300",  
    "rollForward": "latestFeature"  
  }  
}
```

The .NET SDK looks for a *global.json* file in the current working directory (which isn't necessarily the same as the project directory) or one of its parent directories.

For information about specifying the runtime version instead of the SDK version, see [Target frameworks](#).

global.json schema

sdk

Type: `object`

Specifies information about the .NET SDK to select.

version

- Type: `string`

The version of the .NET SDK to use.

This field:

- Doesn't have wildcard support; that is, you must specify the full version number.
- Doesn't support version ranges.

allowPrerelease

- Type: `boolean`
- Available since: .NET Core 3.0 SDK.

Indicates whether the SDK resolver should consider prerelease versions when selecting the SDK version to use.

If you don't set this value explicitly, the default value depends on whether you're running from Visual Studio:

- If you're **not** in Visual Studio, the default value is `true`.

- If you are in Visual Studio, it uses the prerelease status requested. That is, if you're using a Preview version of Visual Studio or you set the **Use previews of the .NET SDK** option (under Tools > Options > Environment > Preview Features), the default value is `true`. Otherwise, the default value is `false`.

`rollForward`

- Type: `string`
- Available since: .NET Core 3.0 SDK.

The roll-forward policy to use when selecting an SDK version, either as a fallback when a specific SDK version is missing or as a directive to use a higher version. A [version](#) must be specified with a `rollForward` value, unless you're setting it to `latestMajor`. The default roll forward behavior is determined by the [matching rules](#).

To understand the available policies and their behavior, consider the following definitions for an SDK version in the format `x.y.znn`:

- `x` is the major version.
- `y` is the minor version.
- `z` is the feature band.
- `nn` is the patch version.

The following table shows the possible values for the `rollForward` key:

VALUE	BEHAVIOR
<code>patch</code>	Uses the specified version. If not found, rolls forward to the latest patch level. If not found, fails.
<code>feature</code>	This value is the legacy behavior from the earlier versions of the SDK. Uses the latest patch level for the specified major, minor, and feature band. If not found, rolls forward to the next higher feature band within the same major/minor and uses the latest patch level for that feature band. If not found, fails.
<code>minor</code>	Uses the latest patch level for the specified major, minor, and feature band. If not found, rolls forward to the next higher feature band within the same major/minor version and uses the latest patch level for that feature band. If not found, rolls forward to the next higher minor and feature band within the same major and uses the latest patch level for that feature band. If not found, fails.

VALUE	BEHAVIOR
major	Uses the latest patch level for the specified major, minor, and feature band. If not found, rolls forward to the next higher feature band within the same major/minor version and uses the latest patch level for that feature band. If not found, rolls forward to the next higher minor and feature band within the same major and uses the latest patch level for that feature band. If not found, rolls forward to the next higher major, minor, and feature band and uses the latest patch level for that feature band. If not found, fails.
latestPatch	Uses the latest installed patch level that matches the requested major, minor, and feature band with a patch level and that is greater or equal than the specified value. If not found, fails.
latestFeature	Uses the highest installed feature band and patch level that matches the requested major and minor with a feature band and patch level that is greater or equal than the specified value. If not found, fails.
latestMinor	Uses the highest installed minor, feature band, and patch level that matches the requested major with a minor, feature band, and patch level that is greater or equal than the specified value. If not found, fails.
latestMajor	Uses the highest installed .NET SDK with a version that is greater or equal than the specified value. If not found, fail.
disable	Doesn't roll forward. Exact match required.

msbuild-sdks

Type: `object`

Lets you control the project SDK version in one place rather than in each individual project. For more information, see [How project SDKs are resolved](#).

Examples

The following example shows how to not use prerelease versions:

```
{
  "sdk": {
    "allowPrerelease": false
  }
}
```

The following example shows how to use the highest version installed that's greater or equal than the specified version. The JSON shown disallows any SDK version earlier than 2.2.200 and allows 2.2.200 or any later version, including 3.0.xxx and 3.1.xxx.

```
{  
  "sdk": {  
    "version": "2.2.200",  
    "rollForward": "latestMajor"  
  }  
}
```

The following example shows how to use the exact specified version:

```
{  
  "sdk": {  
    "version": "3.1.100",  
    "rollForward": "disable"  
  }  
}
```

The following example shows how to use the latest feature band and patch version installed of a specific major and minor version. The JSON shown disallows any SDK version earlier than 3.1.102 and allows 3.1.102 or any later 3.1.xxx version, such as 3.1.103 or 3.1.200.

```
{  
  "sdk": {  
    "version": "3.1.102",  
    "rollForward": "latestFeature"  
  }  
}
```

The following example shows how to use the highest patch version installed of a specific version. The JSON shown disallows any SDK version earlier than 3.1.102 and allows 3.1.102 or any later 3.1.1xx version, such as 3.1.103 or 3.1.199.

```
{  
  "sdk": {  
    "version": "3.1.102",  
    "rollForward": "latestPatch"  
  }  
}
```

global.json and the .NET CLI

To set an SDK version in the *global.json* file, it's helpful to know which SDK versions are installed on your machine. For information on how to do that, see [How to check that .NET is already installed](#).

To install additional .NET SDK versions on your machine, visit the [Download .NET](#) page.

You can create a new *global.json* file in the current directory by executing the `dotnet new` command, similar to the following example:

```
dotnet new globaljson --sdk-version 6.0.100
```

Matching rules

NOTE

The matching rules are governed by the `dotnet.exe` entry point, which is common across all installed .NET installed runtimes. The matching rules for the latest installed version of the .NET Runtime are used when you have multiple runtimes installed side-by-side or if you're using a `global.json` file.

The following rules apply when determining which version of the SDK to use:

- If no `global.json` file is found, or `global.json` doesn't specify an SDK version nor an `allowPrerelease` value, the highest installed SDK version is used (equivalent to setting `rollForward` to `latestMajor`). Whether prerelease SDK versions are considered depends on how `dotnet` is being invoked.
 - If you're **not** in Visual Studio, prerelease versions are considered.
 - If you are in Visual Studio, it uses the prerelease status requested. That is, if you're using a Preview version of Visual Studio or you set the **Use previews of the .NET SDK** option (under **Tools > Options > Environment > Preview Features**), prerelease versions are considered; otherwise, only release versions are considered.
- If a `global.json` file is found that doesn't specify an SDK version but it specifies an `allowPrerelease` value, the highest installed SDK version is used (equivalent to setting `rollForward` to `latestMajor`). Whether the latest SDK version can be release or prerelease depends on the value of `allowPrerelease`. `true` indicates prerelease versions are considered; `false` indicates that only release versions are considered.
- If a `global.json` file is found and it specifies an SDK version:
 - If no `rollForward` value is set, it uses `latestPatch` as the default `rollForward` policy. Otherwise, check each value and their behavior in the `rollForward` section.
 - Whether prerelease versions are considered and what's the default behavior when `allowPrerelease` isn't set is described in the `allowPrerelease` section.

Troubleshoot build warnings

- The following warnings indicate that your project was compiled using a prerelease version of the .NET SDK:

You are working with a preview version of the .NET Core SDK. You can define the SDK version via a `global.json` file in the current project. More at <https://go.microsoft.com/fwlink/?linkid=869452>.

You are using a preview version of .NET. See: <https://aka.ms/dotnet-core-preview>

.NET SDK versions have a history and commitment of being high quality. However, if you don't want to use a prerelease version, check the different strategies you can use in the `allowPrerelease` section. For machines that have never had a .NET Core 3.0 or higher runtime or SDK installed, you need to create a `global.json` file and specify the exact version you want to use.

- The following warning indicates that your project targets EF Core 1.0 or 1.1, which isn't compatible with .NET Core 2.1 SDK and later versions:

Startup project '{startupProject}' targets framework '.NETCoreApp' version '{targetFrameworkVersion}'. This version of the Entity Framework Core .NET Command-line Tools only supports version 2.0 or higher. For information on using older versions of the tools, see <https://go.microsoft.com/fwlink/?linkid=871254>.

Starting with .NET Core 2.1 SDK (version 2.1.300), the `dotnet ef` command comes included in the SDK.

To compile your project, install .NET Core 2.0 SDK (version 2.1.201) or earlier on your machine and define the desired SDK version using the *global.json* file. For more information about the `dotnet ef` command, see [EF Core .NET Command-line Tools](#).

See also

- [How project SDKs are resolved](#)

.NET SDK and .NET CLI telemetry

9/20/2022 • 7 minutes to read • [Edit Online](#)

The [.NET SDK](#) includes a telemetry feature that collects usage data and sends it to Microsoft when you use [.NET CLI](#) commands. The usage data includes exception information when the .NET CLI crashes. The .NET CLI comes with the .NET SDK and is the set of verbs that enable you to build, test, and publish your .NET apps. Telemetry data helps the .NET team understand how the tools are used so they can be improved. Information on failures helps the team resolve problems and fix bugs.

The collected data is published in aggregate under the [Creative Commons Attribution License](#). Some of the collected data is published at [.NET CLI Telemetry Data](#).

Scope

`dotnet` has two functions: to run apps and to execute CLI commands. Telemetry *isn't collected* when using `dotnet` to start an application in the following format:

- `dotnet [path-to-app].dll`

Telemetry *is collected* when using any of the [.NET CLI commands](#), such as:

- `dotnet build`
- `dotnet pack`
- `dotnet run`

How to opt out

The .NET SDK telemetry feature is enabled by default. To opt out of the telemetry feature, set the `DOTNET_CLI_TELEMETRY_OPTOUT` environment variable to `1` or `true`.

A single telemetry entry is also sent by the .NET SDK installer when a successful installation happens. To opt out, set the `DOTNET_CLI_TELEMETRY_OPTOUT` environment variable before you install the .NET SDK.

IMPORTANT

To opt out after you started the installer: close the installer, set the environment variable, and then run the installer again with that value set.

Disclosure

The .NET SDK displays text similar to the following when you first run one of the [.NET CLI commands](#) (for example, `dotnet build`). Text may vary slightly depending on the version of the SDK you're running. This "first run" experience is how Microsoft notifies you about data collection.

```
Telemetry
-----
The .NET tools collect usage data in order to help us improve your experience. The data is collected by
Microsoft and shared with the community. You can opt-out of telemetry by setting the
DOTNET_CLI_TELEMETRY_OPTOUT environment variable to '1' or 'true' using your favorite shell.

Read more about .NET CLI Tools telemetry: https://aka.ms/dotnet-cli-telemetry
```

To disable this message and the .NET welcome message, set the `DOTNET_NOLOGO` environment variable to `true`. Note that this variable has no effect on telemetry opt out.

Data points

The telemetry feature doesn't collect personal data, such as usernames or email addresses. It doesn't scan your code and doesn't extract project-level data, such as name, repository, or author. The data is sent securely to Microsoft servers using [Azure Monitor](#) technology, held under restricted access, and published under strict security controls from secure [Azure Storage](#) systems.

Protecting your privacy is important to us. If you suspect the telemetry is collecting sensitive data or the data is being insecurely or inappropriately handled, file an issue in the [dotnet/sdk](#) repository or send an email to dotnet@microsoft.com for investigation.

The telemetry feature collects the following data:

SDK VERSIONS	DATA
All	Timestamp of invocation.
All	Command invoked (for example, "build"), hashed starting in 2.1.
All	Three octet IP address used to determine the geographical location.
All	Operating system and version.
All	Runtime ID (RID) the SDK is running on.
All	.NET SDK version.
All	Telemetry profile: an optional value only used with explicit user opt-in and used internally at Microsoft.
>=2.0	Command arguments and options: several arguments and options are collected (not arbitrary strings). See collected options . Hashed after 2.1.300.
>=2.0	Whether the SDK is running in a container.
>=2.0	Target frameworks (from the <code>TargetFramework</code> event), hashed starting in 2.1.
>=2.0	Hashed Media Access Control (MAC) address (SHA256).
>=2.0	Hashed current working directory.
>=2.0	Install success report, with hashed installer exe filename.
>=2.1.300	Kernel version.
>=2.1.300	Libc release/version.

SDK VERSIONS	DATA
>=3.0.100	Whether the output was redirected (true or false).
>=3.0.100	On a CLI/SDK crash, the exception type and its stack trace (only CLI/SDK code is included in the stack trace sent). For more information, see Crash exception telemetry .
>=5.0.100	Hashed TargetFrameworkVersion used for build (MSBuild property)
>=5.0.100	Hashed RuntimeIdentifier used for build (MSBuild property)
>=5.0.100	Hashed SelfContained used for build (MSBuild property)
>=5.0.100	Hashed UseApphost used for build (MSBuild property)
>=5.0.100	Hashed OutputType used for build (MSBuild property)
>=5.0.201	Hashed PublishReadyToRun used for build (MSBuild property)
>=5.0.201	Hashed PublishTrimmed used for build (MSBuild property)
>=5.0.201	Hashed PublishSingleFile used for build (MSBuild property)
>=5.0.202	Elapsed time from process start until entering the CLI program's main method, measuring host and runtime startup.
>=5.0.202	Elapsed time for the step that adds .NET Tools to the path on first run.
>=5.0.202	Elapsed time to display first time use notice on first run.
>=5.0.202	Elapsed time for generating ASP.NET Certificate on first run.
>=5.0.202	Elapsed time to parse the CLI input.
>=6.0.100	OS architecture
>=6.0.104	Hashed PublishReadyToRunUseCrossgen2 used for build (MSBuild property)
>=6.0.104	Hashed Crossgen2PackVersion used for build (MSBuild property)
>=6.0.104	Hashed CompileListCount used for build (MSBuild property)
>=6.0.104	Hashed _ReadyToRunCompilationFailures used for build (MSBuild property)

SDK VERSIONS	DATA
>=6.0.300	If the CLI was invoked from a Continuous Integration environment. For more information, see Continuous Integration Detection .
>=7.0.100	Hashed PublishAot used for build (MSBuild property)
>=7.0.100	Hashed PublishProtocol used for build (MSBuild property)

Collected options

Certain commands send additional data. A subset of commands sends the first argument:

COMMAND	FIRST ARGUMENT DATA SENT
<code>dotnet help <arg></code>	The command help is being queried for.
<code>dotnet new <arg></code>	The template name (hashed).
<code>dotnet add <arg></code>	The word <code>package</code> or <code>reference</code> .
<code>dotnet remove <arg></code>	The word <code>package</code> or <code>reference</code> .
<code>dotnet list <arg></code>	The word <code>package</code> or <code>reference</code> .
<code>dotnet sln <arg></code>	The word <code>add</code> , <code>list</code> , or <code>remove</code> .
<code>dotnet nuget <arg></code>	The word <code>delete</code> , <code>locals</code> , or <code>push</code> .
<code>dotnet workload <subcommand> <arg></code>	The word <code>install</code> , <code>update</code> , <code>list</code> , <code>search</code> , <code>uninstall</code> , <code>repair</code> , <code>restore</code> and the workload name (hashed).
<code>dotnet tool <subcommand> <arg></code>	The word <code>install</code> , <code>update</code> , <code>list</code> , <code>search</code> , <code>uninstall</code> , <code>run</code> and the dotnet tool name (hashed).

A subset of commands sends selected options if they're used, along with their values:

OPTION	COMMANDS
<code>--verbosity</code>	All commands
<code>--language</code>	<code>dotnet new</code>
<code>--configuration</code>	<code>dotnet build</code> , <code>dotnet clean</code> , <code>dotnet publish</code> , <code>dotnet run</code> , <code>dotnet test</code>
<code>--framework</code>	<code>dotnet build</code> , <code>dotnet clean</code> , <code>dotnet publish</code> , <code>dotnet run</code> , <code>dotnet test</code> , <code>dotnet vstest</code>
<code>--runtime</code>	<code>dotnet build</code> , <code>dotnet publish</code>

OPTION	COMMANDS
--platform	dotnet vstest
--logger	dotnet vstest
--sdk-package-version	dotnet migrate

Except for `--verbosity` and `--sdk-package-version`, all the other values are hashed starting with .NET Core 2.1.100 SDK.

Template engine telemetry

The `dotnet new` template instantiation command collects additional data for Microsoft-authored templates, starting with .NET Core 2.1.100 SDK:

- `--framework`
- `--auth`

Crash exception telemetry

If the .NET CLI/SDK crashes, it collects the name of the exception and stack trace of the CLI/SDK code. This information is collected to assess problems and improve the quality of the .NET SDK and CLI. This article provides information about the data we collect. It also provides tips on how users building their own version of the .NET SDK can avoid inadvertent disclosure of personal or sensitive information.

The .NET CLI collects information for CLI/SDK exceptions only, not exceptions in your application. The collected data contains the name of the exception and the stack trace. This stack trace is of CLI/SDK code.

The following example shows the kind of data that is collected:

```
System.IO.IOException
at System.ConsolePal.WindowsConsoleStream.Write(Byte[] buffer, Int32 offset, Int32 count)
at System.IO.StreamWriter.Flush(Boolean flushStream, Boolean flushEncoder)
at System.IO.StreamWriter.Write(Char[] buffer)
at System.IO.TextWriter.WriteLine()
at System.IO.TextWriter.SyncTextWriter.WriteLine()
at Microsoft.DotNet.Cli.Utils.Reporter.WriteLine()
at Microsoft.DotNet.Tools.Run.RunCommand.EnsureProjectIsBuilt()
at Microsoft.DotNet.Tools.Run.RunCommand.Execute()
at Microsoft.DotNet.Tools.Run.RunCommand.Run(String[] args)
at Microsoft.DotNet.Cli.Program.ProcessArgs(String[] args, ITelemetry telemetryClient)
at Microsoft.DotNet.Cli.Program.Main(String[] args)
```

Continuous Integration Detection

In order to detect if the .NET CLI is running in a Continuous Integration environment, the .NET CLI probes for the presence and values of several well-known environment variables that common CI providers set.

The full list of environment variables, and what is done with their values, is shown below. Note that in every case, the value of the environment variable is never collected, only used to set a boolean flag.

VARIABLE(S)	PROVIDER	ACTION
TF_BUILD	Azure Pipelines	Parse boolean value

VARIABLE(S)	PROVIDER	ACTION
GITHUB_ACTIONS	GitHub Actions	Parse boolean value
APPVEYOR	Appveyor	Parse boolean value
CI	Many/Most	Parse boolean value
TRAVIS	Travis CI	Parse boolean value
CIRCLECI	Circle CI	Parse boolean value
CODEBUILD_BUILD_ID, AWS_REGION	Amazon Web Services CodeBuild	Check if all are present and non-null
BUILD_ID, BUILD_URL	Jenkins	Check if all are present and non-null
BUILD_ID, PROJECT_ID	Google Cloud Build	Check if all are present and non-null
TEAMCITY_VERSION	TeamCity	Check if present and non-null
JB_SPACE_API_URL	JetBrains Space	Check if present and non-null

Avoid inadvertent disclosure of information

.NET contributors and anyone else running a version of the .NET SDK that they built themselves should consider the path to their SDK source code. If a crash occurs while using a .NET SDK that is a custom debug build or configured with custom build symbol files, the SDK source file path from the build machine is collected as part of the stack trace and isn't hashed.

Because of this, custom builds of the .NET SDK shouldn't be located in directories whose path names expose personal or sensitive information.

See also

- [.NET CLI telemetry data](#)
- [Telemetry reference source \(dotnet/sdk repository\)](#)

.NET SDK error list

9/20/2022 • 15 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET 6 SDK and later versions

This is a complete list of the errors that you might get from the .NET SDK while developing .NET apps. If more info is available for a particular error, the error number is a link.

SDK MESSAGE NUMBER	MESSAGE
NETSDK1001	At least one possible target framework must be specified.
NETSDK1002	Project '{0}' targets '{2}'. It cannot be referenced by a project that targets '{1}'.
NETSDK1003	Invalid framework name: '{0}'.
NETSDK1004	Assets file '{0}' not found. Run a NuGet package restore to generate this file.
NETSDK1005	Assets file '{0}' doesn't have a target for '{1}'. Ensure that restore has run and that you have included '{2}' in the TargetFrameworks for your project.
NETSDK1006	Assets file path '{0}' is not rooted. Only full paths are supported.
NETSDK1007	Cannot find project info for '{0}'. This can indicate a missing project reference.
NETSDK1008	Missing '{0}' metadata on '{1}' item '{2}'.
NETSDK1009	Unrecognized preprocessor token '{0}' in '{1}'.
NETSDK1010	The '{0}' task must be given a value for parameter '{1}' in order to consume preprocessed content.
NETSDK1011	Assets are consumed from project '{0}', but no corresponding MSBuild project path was found in '{1}'.
NETSDK1012	Unexpected file type for '{0}'. Type is both '{1}' and '{2}'.
NETSDK1013	The TargetFramework value '{0}' was not recognized. It may be misspelled. If not, then the TargetFrameworkIdentifier and/or TargetFrameworkVersion properties must be specified explicitly.
NETSDK1014	Content item for '{0}' sets '{1}', but does not provide '{2}' or '{3}'.
NETSDK1015	The preprocessor token '{0}' has been given more than one value. Choosing '{1}' as the value.

SDK MESSAGE NUMBER	MESSAGE
NETSDK1016	Unable to find resolved path for '{0}'.
NETSDK1017	Asset preprocessor must be configured before assets are processed.
NETSDK1018	Invalid NuGet version string: '{0}'.
NETSDK1019	{0} is an unsupported framework.
NETSDK1020	Package Root {0} was incorrectly given for Resolved library {1}.
NETSDK1021	More than one file found for {0}
NETSDK1022	Duplicate '{0}' items were included. The .NET SDK includes '{0}' items from your project directory by default. You can either remove these items from your project file, or set the '{1}' property to '{2}' if you want to explicitly include them in your project file. For more information, see {4}. The duplicate items were: {3}.
NETSDK1023	A PackageReference for '{0}' was included in your project. This package is implicitly referenced by the .NET SDK and you do not typically need to reference it from your project. For more information, see {1}.
NETSDK1024	Folder '{0}' already exists - either delete it or provide a different ComposeWorkingDir.
NETSDK1025	The target manifest {0} provided is of not the correct format.
NETSDK1028	Specify a Runtimeldentifier.
NETSDK1029	Unable to use '{0}' as application host executable as it does not contain the expected placeholder byte sequence '{1}' that would mark where the application name would be written.
NETSDK1030	Given file name '{0}' is longer than 1024 bytes.
NETSDK1031	It is not supported to build or publish a self-contained application without specifying a Runtimeldentifier. You must either specify a Runtimeldentifier or set SelfContained to false.
NETSDK1032	The Runtimeldentifier platform '{0}' and the PlatformTarget '{1}' must be compatible.
NETSDK1042	Could not load PlatformManifest from '{0}' because it did not exist.
NETSDK1043	Error parsing PlatformManifest from '{0}' line {1}. Lines must have the format {2}.

SDK MESSAGE NUMBER	MESSAGE
NETSDK1044	Error parsing PlatformManifest from '{0}' line {1}. {2} '{3}' was invalid.
NETSDK1045	The current .NET SDK does not support targeting {0} {1}. Either target {0} {2} or lower, or use a version of the .NET SDK that supports {0} {1}.
NETSDK1046	The TargetFramework value '{0}' is not valid. To multi-target, use the 'TargetFrameworks' property instead.
NETSDK1047	Assets file '{0}' doesn't have a target for '{1}'. Ensure that restore has run and that you have included '{2}' in the TargetFrameworks for your project. You may also need to include '{3}' in your project's RuntimeIdentifiers.
NETSDK1048	'AdditionalProbingPaths' were specified for GenerateRuntimeConfigurationFiles, but are being skipped because 'RuntimeConfigDevPath' is empty.
NETSDK1049	Resolved file has a bad image, no metadata, or is otherwise inaccessible. {0} {1}.
NETSDK1050	The version of Microsoft.NET.Sdk used by this project is insufficient to support references to libraries targeting .NET Standard 1.5 or higher. Please install version 2.0 or higher of the .NET Core SDK.
NETSDK1051	Error parsing FrameworkList from '{0}'. {1} '{2}' was invalid.
NETSDK1052	Framework list file path '{0}' is not rooted. Only full paths are supported.
NETSDK1053	Pack as tool does not support self contained.
NETSDK1054	Only supports .NET Core.
NETSDK1055	DotnetTool does not support target framework lower than netcoreapp2.1.
NETSDK1056	Project is targeting runtime '{0}' but did not resolve any runtime-specific packages. This runtime may not be supported by the target framework.
NETSDK1057	You are using a preview version of .NET. See https://aka.ms/dotnet-support-policy .
NETSDK1058	Invalid value for ItemSpecToUse parameter: '{0}'. This property must be blank or set to 'Left' or 'Right'
NETSDK1059	The tool '{0}' is now included in the .NET SDK. Information on resolving this warning is available at https://aka.ms/dotnetclitools-in-box .
NETSDK1060	Error reading assets file: {0}

SDK MESSAGE NUMBER	MESSAGE
NETSDK1061	The project was restored using {0} version {1}, but with current settings, version {2} would be used instead. To resolve this issue, make sure the same settings are used for restore and for subsequent operations such as build or publish. Typically this issue can occur if the RuntimeIdentifier property is set during build or publish but not during restore. For more information, see https://aka.ms/dotnet-runtime-patch-selection .
NETSDK1063	The path to the project assets file was not set. Run a NuGet package restore to generate this file.
NETSDK1064	Package {0}, version {1} was not found. It might have been deleted since NuGet restore. Otherwise, NuGet restore might have only partially completed, which might have been due to maximum path length restrictions.
NETSDK1065	Cannot find app host for {0}. {0} could be an invalid runtime identifier (RID). For more information about RID, see https://aka.ms/rid-catalog .
NETSDK1067	Self-contained applications are required to use the application host. Either set SelfContained to false or set UseAppHost to true.
NETSDK1068	The framework-dependent application host requires a target framework of at least 'netcoreapp2.1'.
NETSDK1069	This project uses a library that targets .NET Standard 1.5 or higher, and the project targets a version of .NET Framework that doesn't have built-in support for that version of .NET Standard. Visit https://aka.ms/net-standard-known-issues for a set of known issues. Consider retargeting to .NET Framework 4.7.2.
NETSDK1070	The application configuration file must have root configuration element.
NETSDK1071	A PackageReference to '{0}' specified a Version of {1}. Specifying the version of this package is not recommended. For more information, see https://aka.ms/sdkimplicitrefs .
NETSDK1072	Unable to use '{0}' as application host executable because it's not a Windows executable for the CUI (Console) subsystem.
NETSDK1073	The FrameworkReference '{0}' was not recognized.
NETSDK1074	The application host executable will not be customized because adding resources requires that the build be performed on Windows (excluding Nano Server).
NETSDK1075	Update handle is invalid. This instance may not be used for further updates.
NETSDK1076	AddResource can only be used with integer resource types.

SDK MESSAGE NUMBER	MESSAGE
NETSDK1077	Failed to lock resource.
NETSDK1078	Unable to use '{0}' as application host executable because it's not a Windows PE file.
NETSDK1079	The Microsoft.AspNetCore.All package is not supported when targeting .NET Core 3.0 or higher. A FrameworkReference to Microsoft.AspNetCore.App should be used instead, and will be implicitly included by Microsoft.NET.Sdk.Web.
NETSDK1080	A PackageReference to Microsoft.AspNetCore.App is not necessary when targeting .NET Core 3.0 or higher. If Microsoft.NET.Sdk.Web is used, the shared framework will be referenced automatically. Otherwise, the PackageReference should be replaced with a FrameworkReference.
NETSDK1081	The targeting pack for {0} was not found. You may be able to resolve this by running a NuGet restore on the project.
NETSDK1082	There was no runtime pack for {0} available for the specified RuntimeIdentifier '{1}'.
NETSDK1083	The specified RuntimeIdentifier '{0}' is not recognized.
NETSDK1084	There is no application host available for the specified RuntimeIdentifier '{0}'.
NETSDK1085	The 'NoBuild' property was set to true but the 'Build' target was invoked.
NETSDK1086	A FrameworkReference for '{0}' was included in the project. This is implicitly referenced by the .NET SDK and you do not typically need to reference it from your project. For more information, see {1}.
NETSDK1087	Multiple FrameworkReference items for '{0}' were included in the project.
NETSDK1088	The COMVisible class '{0}' must have a GuidAttribute with the CLSID of the class to be made visible to COM in .NET Core.
NETSDK1089	The '{0}' and '{1}' types have the same CLSID '{2}' set in their GuidAttribute. Each COMVisible class needs to have a distinct guid for their CLSID.
NETSDK1090	The supplied assembly '{0}' is not valid. Cannot generate a CLSIDMap from it.
NETSDK1091	Unable to find a .NET Core COM host. The .NET Core COM host is only available on .NET Core 3.0 or higher when targeting Windows.

SDK MESSAGE NUMBER	MESSAGE
NETSDK1092	The CLSIDMap cannot be embedded on the COM host because adding resources requires that the build be performed on Windows (excluding Nano Server).
NETSDK1093	Project tools (DotnetCliTool) only support targeting .NET Core 2.2 and lower.
NETSDK1094	Unable to optimize assemblies for performance: a valid runtime package was not found. Either set the PublishReadyToRun property to false, or use a supported runtime identifier when publishing. When targeting .NET 6 or higher, make sure to restore packages with the PublishReadyToRun property set to true.
NETSDK1095	Optimizing assemblies for performance is not supported for the selected target platform or architecture. Please verify you are using a supported runtime identifier, or set the PublishReadyToRun property to false.
NETSDK1096	Optimizing assemblies for performance failed. You can either exclude the failing assemblies from being optimized, or set the PublishReadyToRun property to false.
NETSDK1097	It is not supported to publish an application to a single-file without specifying a Runtimeldentifier. You must either specify a Runtimeldentifier or set PublishSingleFile to false.
NETSDK1098	Applications published to a single-file are required to use the application host. You must either set PublishSingleFile to false or set UseAppHost to true.
NETSDK1099	Publishing to a single-file is only supported for executable applications.
NETSDK1100	To build a project targeting Windows on this operating system, set the EnableWindowsTargeting property to true.
NETSDK1102	Optimizing assemblies for size is not supported for the selected publish configuration. Please ensure that you are publishing a self-contained app.
NETSDK1103	RollForward setting is only supported on .NET Core 3.0 or higher.
NETSDK1104	RollForward value '{0}' is invalid. Allowed values are {1}.
NETSDK1105	Windows desktop applications are only supported on .NET Core 3.0 or higher.
NETSDK1106	Microsoft.NET.Sdk.WindowsDesktop requires 'UseWpf' or 'UseWindowsForms' to be set to 'true'.
NETSDK1107	Microsoft.NET.Sdk.WindowsDesktop is required to build Windows desktop applications. 'UseWpf' and 'UseWindowsForms' are not supported by the current SDK.

SDK MESSAGE NUMBER	MESSAGE
NETSDK1109	Runtime list file '{0}' was not found. Report this error to the .NET team here: https://aka.ms/dotnet-sdk-issue .
NETSDK1110	More than one asset in the runtime pack has the same destination sub-path of '{0}'. Report this error to the .NET team here: https://aka.ms/dotnet-sdk-issue .
NETSDK1111	Failed to delete output apphost: {0}.
NETSDK1112	The runtime pack for {0} was not downloaded. Try running a NuGet restore with the RuntimeIdentifier '{1}'.
NETSDK1113	Failed to create apphost (attempt {0} out of {1}): {2}.
NETSDK1114	Unable to find a .NET Core IJW host. The .NET Core IJW host is only available on .NET Core 3.1 or higher when targeting Windows.
NETSDK1115	The current .NET SDK does not support .NET Framework without using .NET SDK Defaults. It is likely due to a mismatch between C++/CLI project CLRSupport property and TargetFramework.
NETSDK1116	C++/CLI projects targeting .NET Core must be dynamic libraries.
NETSDK1117	Does not support publish of C++/CLI project targeting dotnet core.
NETSDK1118	C++/CLI projects targeting .NET Core cannot be packed.
NETSDK1119	C++/CLI projects targeting .NET Core cannot use EnableComHosting=true.
NETSDK1120	C++/CLI projects targeting .NET Core require a target framework of at least 'netcoreapp3.1'.
NETSDK1121	C++/CLI projects targeting .NET Core cannot use SelfContained=true.
NETSDK1122	ReadyToRun compilation will be skipped because it is only supported for .NET Core 3.0 or higher.
NETSDK1123	Publishing an application to a single-file requires .NET Core 3.0 or higher.
NETSDK1124	Trimming assemblies requires .NET Core 3.0 or higher.
NETSDK1125	Publishing to a single-file is only supported for netcoreapp target.
NETSDK1126	Publishing ReadyToRun using Crossgen2 is only supported for self-contained applications.

SDK MESSAGE NUMBER	MESSAGE
NETSDK1127	The targeting pack {0} is not installed. Please restore and try again.
NETSDK1128	COM hosting does not support self-contained deployments.
NETSDK1129	The 'Publish' target is not supported without specifying a target framework. The current project targets multiple frameworks, you must specify the framework for the published application.
NETSDK1130	{1} cannot be referenced. Referencing a Windows Metadata component directly when targeting .NET 5 or higher is not supported. For more information, see https://aka.ms/netsdk1130 .
NETSDK1131	Producing a managed Windows Metadata component with WinMDExp is not supported when targeting {0}.
NETSDK1132	No runtime pack information was available for {0}.
NETSDK1133	There was conflicting information about runtime packs available for {0}.
NETSDK1134	Building a solution with a specific RuntimeIdentifier is not supported. If you would like to publish for a single RID, specify the RID at the individual project level instead.
NETSDK1135	SupportedOSPlatformVersion {0} cannot be higher than TargetPlatformVersion {1}.
NETSDK1136	The target platform must be set to Windows (usually by including '-windows' in the TargetFramework property) when using Windows Forms or WPF, or referencing projects or packages that do so.
NETSDK1137	It is no longer necessary to use the Microsoft.NET.Sdk.WindowsDesktop SDK. Consider changing the Sdk attribute of the root Project element to 'Microsoft.NET.Sdk'.
NETSDK1138	The target framework '{0}' is out of support and will not receive security updates in the future. Please refer to https://aka.ms/dotnet-core-support for more information about the support policy.
NETSDK1139	The target platform identifier {0} was not recognized.
NETSDK1140	{0} is not a valid TargetPlatformVersion for {1}. Valid versions include:
NETSDK1141	Unable to resolve the .NET SDK version as specified in the global.json located at {0}.
NETSDK1142	Including symbols in a single file bundle is not supported when publishing for .NET5 or higher.

SDK MESSAGE NUMBER	MESSAGE
NETSDK1143	Including all content in a single file bundle also includes native libraries. If <code>IncludeAllContentForSelfExtract</code> is true, <code>IncludeNativeLibrariesForSelfExtract</code> must not be false.
NETSDK1144	Optimizing assemblies for size failed. Optimization can be disabled by setting the <code>PublishTrimmed</code> property to false.
NETSDK1145	The {0} pack is not installed and NuGet package restore is not supported. Upgrade Visual Studio, remove <code>global.json</code> if it specifies a certain SDK version, and uninstall the newer SDK. For more options visit https://aka.ms/targeting-apphost-pack-missing . Pack Type:{0}, Pack directory: {1}, targetframework: {2}, Pack PackageId: {3}, Pack Package Version: {4}.
NETSDK1146	PackAsTool does not support <code>TargetPlatformIdentifier</code> being set. For example, <code>TargetFramework</code> cannot be <code>net5.0-windows</code> , only <code>net5.0</code> . PackAsTool also does not support <code>UseWPF</code> or <code>UseWindowsForms</code> when targeting .NET 5 and higher.
NETSDK1147	To build this project, the following workloads must be installed: {0}.
NETSDK1148	A referenced assembly was compiled using a newer version of <code>Microsoft.Windows.SDK.NET.dll</code> . Please update to a newer .NET SDK in order to reference this assembly.
NETSDK1149	{0} cannot be referenced because it uses built-in support for WinRT, which is no longer supported in .NET 5 and higher. An updated version of the component supporting .NET 5 is needed. For more information, see https://aka.ms/netsdk1149 .
NETSDK1150	The referenced project '{0}' is a non self-contained executable. A non self-contained executable cannot be referenced by a self-contained executable. For more information, see https://aka.ms/netsdk1150 .
NETSDK1151	The referenced project '{0}' is a self-contained executable. A self-contained executable cannot be referenced by a non self-contained executable. For more information, see https://aka.ms/netsdk1151 .
NETSDK1152	Found multiple publish output files with the same relative path: {0}.
NETSDK1153	CrossgenTool not specified in PDB compilation mode.
NETSDK1154	Crossgen2Tool must be specified when <code>UseCrossgen2</code> is set to true.

SDK MESSAGE NUMBER	MESSAGE
NETSDK1155	Crossgen2Tool executable '{0}' not found.
NETSDK1156	.NET host executable '{0}' not found.
NETSDK1157	JIT library '{0}' not found.
NETSDK1158	Required '{0}' metadata missing on Crossgen2Tool item.
NETSDK1159	CrossgenTool must be specified when UseCrossgen2 is set to false.
NETSDK1160	CrossgenTool executable '{0}' not found.
NETSDK1161	DiaSymReader library '{0}' not found.
NETSDK1162	PDB generation: R2R executable '{0}' not found.
NETSDK1163	Input assembly '{0}' not found.
NETSDK1164	Missing output PDB path in PDB generation mode (OutputPDBImage metadata).
NETSDK1165	Missing output R2R image path (OutputR2RImage metadata).
NETSDK1166	Cannot emit symbols when publishing for .NET 5 with Crossgen2 using composite mode.
NETSDK1167	Compression in a single file bundle is only supported when publishing for .NET 6 or higher.
NETSDK1168	WPF is not supported or recommended with trimming enabled. Please go to https://aka.ms/dotnet-illink/wpf for more details.
NETSDK1169	The same resource ID {0} was specified for two type libraries '{1}' and '{2}'. Duplicate type library IDs are not allowed.
NETSDK1170	The provided type library ID '{0}' for type libary '{1}' is invalid. The ID must be a positive integer less than 65536.
NETSDK1171	An integer ID less than 65536 must be provided for type library '{0}' because more than one type library is specified.
NETSDK1172	The provided type library '{0}' does not exist.
NETSDK1173	The provided type library '{0}' is in an invalid format.
NETSDK1174	The abbreviation of -p for --project is deprecated. Please use --project.

SDK MESSAGE NUMBER	MESSAGE
NETSDK1175	Windows Forms is not supported or recommended with trimming enabled. Please go to https://aka.ms/dotnet-ilink/windows-forms for more details.
NETSDK1176	Compression in a single file bundle is only supported when publishing a self-contained application.
NETSDK1177	Failed to sign apphost with error code {1}: {0}.
NETSDK1178	The project depends on the following workload packs that do not exist in any of the workloads available in this installation: {0}.
NETSDK1179	One of '--self-contained' or '--no-self-contained' options are required when '--runtime' is used.
NETSDK1181	Error getting pack version: Pack '{0}' was not present in workload manifests.
NETSDK1182	Targeting .NET 6.0 or higher in Visual Studio 2019 is not supported.
NETSDK1183	Unable to optimize assemblies for Ahead of time compilation: a valid runtime package was not found. Either set the PublishAot property to false, or use a supported runtime identifier when publishing. When targeting .NET 7 or higher, make sure to restore packages with the PublishAot property set to true.
NETSDK1184	The Targeting Pack for FrameworkReference '{0}' was not available. This may be because DisableTransitiveFrameworkReferenceDownloads was set to true.
NETSDK1185	The Runtime Pack for FrameworkReference '{0}' was not available. This may be because DisableTransitiveFrameworkReferenceDownloads was set to true.
NETSDK1186	This project depends on Maui Essentials through a project or NuGet package reference, but doesn't declare that dependency explicitly. To build this project, you must set the UseMauiEssentials property to true (and install the Maui workload if necessary).

NETSDK1004: Assets file not found

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 2.1.100 SDK and later versions

NuGet writes a file named *project.assets.json* in the *obj* folder, and the .NET SDK uses it to get information about packages to pass into the compiler. This error occurs when the assets file *project.assets.json* is not found during build. The full error message is similar to the following example:

NETSDK1004: Assets file 'C:\path\to\project.assets.json' not found. Run a NuGet package restore to generate this file.

Here are some possible causes of the error:

- You are running the `dotnet build` command from a directory path that contains a `%` character. To resolve the error, remove the `%` from the folder name, and rerun `dotnet build`.
- A change to the project file wasn't automatically detected and restored by the project system. To resolve the error, open a command prompt and run `dotnet restore` on the project.
- A project was restored separately by an older version of Nuget.exe. To resolve the error, open a command prompt and run `dotnet restore` on the project.
- An earlier error, such as NETSDK1045 (the version of the SDK you're using doesn't support the project's target framework), prevented NuGet from creating the project assets file. To resolve the NETSDK1004 error, resolve the earlier error, and then run `dotnet restore` on the project.
- App Center CI is building a project that has an external assembly that is not in NuGet. To resolve the error, use a NuGet package for the assembly.
- You added a solution folder in Visual Studio with a name that starts with a period. To resolve the error, remove the leading period from the folder name.
- You have a source in the `<packageSources>` section in the *NuGet.Config* file with a path that doesn't exist. To resolve the error, edit the *NuGet.Config* file to correct the package source path.

NETSDK1005 and NETSDK1047: Asset file is missing target

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 2.1.100 SDK and later versions

When the .NET SDK issues error NETSDK1005 or NETSDK1047, the project's assets file is missing information on one of your target frameworks. NuGet writes a file named `project.assets.json` in the `obj` folder, and the .NET SDK uses it to get information about packages to pass into the compiler. In .NET 5, NuGet added a new field named `TargetFrameworkAlias`, so earlier versions of MSBuild or NuGet generate an assets file without the new field. For more information, see [error NETSDK1005](#).

Here are some actions you can take that may resolve the error:

- Make sure that you're using MSBuild version 16.8 or later and NuGet version 5.8 or later, and restore the project after updating your tools. When you're using NuGet version 5.8 or later, you should be using Visual Studio 2019 version 16.8 or later, MSBuild version 16.8 or later, and .NET 5 SDK or later.
- If you get the error while building a project in Visual Studio 2019 for the first time after installing version 16.8 or after changing the project's target framework, build the project a second time.
- Delete the `obj` folder before building the project.
- Make sure that the missing target value is included in the `TargetFrameworks` property of your project.
- If you're building a Docker image, make sure the `.dockerignore` file ignores the `bin` and `obj` directories. For more information, see GitHub pull request [dotnet/docs #29530](#).

NETSDK1013: The TargetFramework value was not recognized

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 3.1.100 SDK and later versions

The SDK tries to parse the values provided in the project file for `<TargetFramework>` or `<TargetFrameworks>` into a well known value. If the value is not recognized, the `TargetFrameworkIdentifier` or `TargetFrameworkVersion` value may be set to an empty string or `Unsupported`.

To resolve this, check the spelling of your `TargetFramework` value from the list of [supported frameworks](#). You can also set the `TargetFrameworkIdentifier` and `TargetFrameworkVersion` properties directly in your project file.

```
<PropertyGroup Condition="$(TargetFrameworkIdentifier)' == ''">
  <TargetFrameworkIdentifier>.NETCOREAPP</TargetFrameworkIdentifier>
  <TargetFrameworkVersion>3.1</TargetFrameworkVersion>
</PropertyGroup>
```

NETSDK1022: Duplicate items were included.

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 2.1.100 SDK and later versions

Starting in Visual Studio 2017 / MSBuild version 15.3, the .NET SDK automatically includes items from the project directory by default. This includes `Compile` and `Content` targets. This should greatly clean up your project file and reduce the complexity you have there.

You can revert to the earlier behavior by setting the correct property to false.

Example for `Compile` items:

```
<PropertyGroup>
    <EnableDefaultCompileItems>false</EnableDefaultCompileItems>
</PropertyGroup>
```

NETSDK1045: The current .NET SDK does not support 'newer version' as a target.

9/20/2022 • 3 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 2.1.100 SDK and later versions

This error occurs when the build tools can't find the version of the .NET SDK that's needed to build a project. This is typically due to a .NET SDK installation or configuration issue. The full error message is similar to the following example:

```
NETSDK1045: The current .NET SDK does not support 'newer version' as a target. Either target 'older version' or lower, or use a .NET SDK version that supports 'newer version'.
```

The following sections describe some of the possible reasons for this error. Check each one and see which one applies to you. Keep in mind that when making changes to the environment or the configuration files, you might have to restart command windows, restart Visual Studio, or reboot your machine, for your changes to take effect.

.NET SDK version

Open the project file (.csproj, .vbproj, or .fsproj) and check the target framework. This is the version of the framework that your app is trying to use.

```
<TargetFramework>netcoreapp3.0</TargetFramework>
```

Make sure that the version of .NET listed is installed on the machine. You can list the installed versions by using the following command (open a Developer Command Prompt and run this command):

```
dotnet --list-sdks
```

x86 or x64 architecture

Each version of the .NET SDK is available in both x86 and x64 architecture. The project might be trying to find the .NET SDK for the wrong architecture, or the .NET SDK for the architecture your project needs might not be installed. Check the installation folders for the architecture you need. For example, on Windows, the x86 version of the .NET SDK is installed in *C:\Program Files (x86)\dotnet* and the x64 version is installed in *C:\Program Files\dotnet*. See [How to check that .NET is already installed](#) and choose your operating system to find out how to detect what's installed on your machine.

If the version you need isn't installed, find the one you need at the [.NET Downloads](#) page.

Preview not enabled

If you have a preview installed of the requested .NET SDK version, you also need to set the option to enable previews in Visual Studio. Go to **Tools > Options > Environment > Preview Features**, and make sure that **Use previews of the .NET Core SDK** is checked.

Visual Studio version

For example, .NET Core 3.0 and later require Visual Studio 2019. Upgrade to [Visual Studio 2019 version 16.3](#) or later to build your project.

PATH environment variable

The build tools use the PATH environment variable to find the right version of the .NET build tools. If the PATH environment variable contains direct paths to older build tools, this error message could appear. Make sure the only path to the .NET tools in the PATH environment variable is to the top-level *dotnet* folder, for example, *C:\Program Files\dotnet*. An example of an incorrect PATH would be something like *C:\Program Files\dotnet\2.1.0\sdk*.

MSBuildSDKPath environment variable

Check the MSBuildSDKPath environment variable. This optional environment variable is recognized by MSBuild and if set, overrides the default value. It might be set to a specific older version of the .NET SDK. If it's set, try deleting it and rebuilding your project.

global.json file

Check for a *global.json* file in the root folder in your project and up the directory chain to the root of the volume, since it can be anywhere in the folder structure. If it contains an SDK version, delete the `sdk` node and all its children, or update it to the desired newer .NET Core version.

```
{  
  "sdk": {  
    "version": "2.1.0"  
  }  
}
```

The *global.json* file is not required, so if it doesn't contain anything other than the `sdk` node, you can delete the whole file.

Directory.build.props file

The *Directory.build.props* file is an optional MSBuild file that can set global properties. Check for these files in the solution folder and up the directory chain to the root of the volume, since they can be anywhere in the folder structure. Look for `TargetFramework` elements, or settings of `MSBuildSDKPath` that could override your desired settings.

See also

- [.NET Downloads](#)
- [The Current .NET SDK does not support targeting .NET Core 3.0 – Fix](#)

NETSDK1059: Project contains obsolete .NET CLI tool

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 2.1.100 SDK and later versions

When the .NET SDK issues warning NETSDK1059, your project contains an obsolete .NET CLI tool. As of .NET Core 2.1, these tools are included in the .NET SDK and do not need to be explicitly referenced by the project. For more information, see [Project tools now included in SDK](#).

NETSDK1064: Package not found

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 2.1.100 SDK and later versions

This error occurs when the build tools can't find a NuGet package that's needed to build a project. This is typically due to a package restore issue. The full error message is similar to the following example:

NETSDK1064: Package 'PackageName', version x.x.x was not found. It might have been deleted since NuGet restore. Otherwise, NuGet restore might have only partially completed, which might have been due to maximum path length restrictions.

Here are some actions you can take to resolve this error:

- Add the `/restore` option to your *MSBuild.exe* command. Don't use `/t:Restore;Build`, as that can result in subtle bugs. An alternative is to use the `dotnet build` command, since it automatically does a package restore.
- If you're running package restore by using Visual Studio 2019 or *MSBuild.exe*, the error may be caused by maximum path length restrictions. For more information, see [Long Path Support \(NuGet CLI\)](#) and [NuGet/Home issue #3324](#).
- If you're restoring with x86 *nuget.exe* and building with x64 *MSBuild.exe*, the mismatched bitness could cause this error. The build can't find the packages that the restore claims it acquired because the path in *project.assets.json* doesn't work in a process of different bitness. To resolve the error, use tools of the same bitness for restore and build, or configure NuGet to restore packages to a folder that doesn't virtualize between x86 and x64. For more information, see [dotnet/core issue #4332](#).
- If you're building a Docker image, make sure the *.dockerignore* file ignores the *bin* and *obj* directories. For more information, see [NETSDK1064: Package DnsClient, 1.2.0 was not found](#).

NETSDK1071: Explicitly versioned PackageReference to a metapackage that would be included with the framework.

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET 5.0.100 SDK and later versions

When the .NET SDK issues warning NETSDK1071, it suggests there may be a version conflict in the future between the version of a metapackage specified in a PackageReference and the version of that metapackage as implicitly referenced via a TargetFramework property:

```
<PropertyGroup>
  <TargetFramework>netcoreapp3.1</TargetFramework>
</PropertyGroup>
```

Since the TargetFramework automatically brings in a version of the metapackage, the versions will conflict should they ever differ.

To resolve this:

1. Consider, when targeting .NET Core or .NET Standard, avoiding explicit references to `Microsoft.NETCore.App` or `NETStandard.Library` in your project file.
2. If you need a specific version of the runtime when targeting .NET Core, use the `<RuntimeFrameworkVersion>` property instead of referencing the metapackage directly. As an example, this could happen if you're using [self-contained deployments](#) and need a specific patch of the 1.0.0 LTS runtime.
3. If you need a specific version of `NetStandard.Library` when targeting .NET Standard, you can use the `<NetStandardImplicitPackageVersion>` property and set it to the version you need.
4. Don't explicitly add or update references to either `Microsoft.NETCore.App` or `NETStandard.Library` in .NET Framework projects. NuGet automatically installs any version of `NETStandard.Library` you need when using a .NET Standard-based NuGet package.
5. Do not specify a version for `Microsoft.AspNetCore.App` or `Microsoft.AspNetCore.All` when using .NET Core 2.1+, as the .NET SDK automatically selects the appropriate version. (Note: This only works when targeting .NET Core 2.1 if the project also uses `Microsoft.NET.Sdk.Web`. This problem was resolved in the .NET Core 2.2 SDK.)
6. If you want the warning to go away, you can also disable it:

```
<PackageReference Include="Microsoft.NetCore.App" Version="2.2.8" >
  <AllowExplicitVersion>true</AllowExplicitVersion>
</PackageReference>
```

NETSDK1073: The FrameworkReference was not recognized

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 2.1.100 SDK and later versions

This error typically means there is a version of a particular FrameworkReference that the SDK cannot find. Try deleting your *obj* and *bin* folders and running `dotnet restore` to redownload the latest targeting packs.

Alternatively, there could be an issue with your install, so ensure you're on the latest versions of .NET and Visual Studio

NETSDK1079: The Microsoft.AspNetCore.All package is not supported when targeting .NET Core 3.0 or higher.

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core SDK 3.1.100 and later versions

You may receive this error message when:

- You retarget an ASP.NET Core project from .NET Core 2.2 or earlier to .NET Core 3.0 or later.
- The project uses the Microsoft.AspNetCore.All package.

Remove the `PackageReference` for Microsoft.AspNetCore.All. You may also need to add package references for packages that were referenced from Microsoft.AspNetCore.All but are not included in the ASP.NET Core shared framework. These packages are listed here: [Migrating from Microsoft.AspNetCore.All to Microsoft.AspNetCore.App](#).

See also [Migrate from ASP.NET Core 2.2 to 3.0](#)

NETSDK1080: PackageReference to Microsoft.AspNetCore.App is not necessary

9/20/2022 • 2 minutes to read • [Edit Online](#)

NETSDK1080 warns you that the `PackageReference` element for `Microsoft.AspNetCore.App` in your project file is not necessary. The full error message is similar to the following example:

```
warning NETSDK1080: A PackageReference to Microsoft.AspNetCore.App is not necessary when targeting .NET Core 3.0 or higher. If Microsoft.NET.Sdk.Web is used, the shared framework will be referenced automatically. Otherwise, the PackageReference should be replaced with a FrameworkReference.
```

This error typically occurs after you've upgraded a project to .NET Core 3.0 or later, from an earlier version that required `PackageReference` entries in the project file.

ASP.NET Core project files

For example, your original project file might look like this example:

```
<Project Sdk="Microsoft.NET.Sdk.Web">

<PropertyGroup>
  <TargetFramework>netcoreapp2.2</TargetFramework>
  <AspNetCoreHostingModel>InProcess</AspNetCoreHostingModel>
</PropertyGroup>

<ItemGroup>
  <PackageReference Include="Microsoft.AspNetCore.App"/>
  <PackageReference Include="Microsoft.AspNetCore.Razor.Design" Version="2.2.0" PrivateAssets="All" />
</ItemGroup>

</Project>
```

After updating to .NET Core 3.1 the project file for the same project should look like the following example:

```
<Project Sdk="Microsoft.NET.Sdk.Web">

<PropertyGroup>
  <TargetFramework>netcoreapp3.1</TargetFramework>
</PropertyGroup>

</Project>
```

Make these changes, in particular delete the `PackageReference` element, to eliminate the warning. For more information, see [Remove obsolete package references](#).

Class library project

In a class library project that uses ASP.NET Core APIs, replace the `PackageReference` with a `FrameworkReference`, as shown in the following example:

```
<Project Sdk="Microsoft.NET.Sdk">

<PropertyGroup>
  <TargetFramework>netcoreapp3.1</TargetFramework>
</PropertyGroup>

<ItemGroup>
  <FrameworkReference Include="Microsoft.AspNetCore.App" />
</ItemGroup>

</Project>
```

For more information, see [Use ASP.NET Core APIs in a class library](#).

NETSDK1082: PackageReference to Microsoft.AspNetCore.App is not necessary

9/20/2022 • 2 minutes to read • [Edit Online](#)

NETSDK1082 warns you that the runtime pack for your [runtime identifier](#) (RID) could not be found in your NuGet feed. The full error message is similar to the following example:

```
There was no runtime pack for <RuntimePack> available for the specified RuntimeIdentifier '<RID>'.
```

.NET automatically downloads known runtime packs for self-contained applications but there could be a pointer to one that's not available to you. Investigate your NuGet configuration and feeds to find out why the required runtime pack is missing. In some scenarios, you might have to override the `LatestRuntimeFrameworkVersion` value to one that's available on your NuGet feeds by adding markup like the following example to the project file:

```
<ItemGroup>
  <KnownRuntimePack Update="@{KnownRuntimePack}">
    <LatestRuntimeFrameworkVersion Condition="'%(TargetFramework)' == 'TARGETFRAMEWORK'">EXISTINGVERSION</LatestRuntimeFrameworkVersion>
  </KnownRuntimePack>
</ItemGroup>
```

In this example, `TARGETFRAMEWORK` represents values like `net6.0`, `net5.0`, or `netcoreapp3.1` – basically anything that's in the [NET 5+ \(and .NET Core\)](#) list in [Supported target frameworks](#). `EXISTINGVERSION` would need to be a valid version that has been released. For example, `6.0.7` for `net6.0`, `5.0.17` for `net5.0`, and so forth. For information about released versions, see [Download .NET 6.0](#) and [Download .NET 5.0](#).

NETSDK1100: Set the `EnableWindowsTargeting` property to true

9/20/2022 • 2 minutes to read • [Edit Online](#)

NETSDK1100 indicates that you're building a project that targets Windows on Linux or macOS. The full error message is similar to the following example:

To build a project targeting Windows on this operating system, set the `EnableWindowsTargeting` property to true.

To resolve this error, set the `EnableWindowsTargeting` property to true. You can set it in the project file or by passing `/p:EnableWindowsTargeting=true` to a .NET CLI command, such as `dotnet build`. Here's an example project file:

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>net6.0</TargetFramework>
    <EnableWindowsTargeting>true</EnableWindowsTargeting>
  </PropertyGroup>
</Project>
```

If you want to apply this setting to your whole solution or repository, you can set it in a `Directory.Build.props` file.

By default, .NET downloads all targeting packs (and runtime packs for self-contained builds) for the current target framework whether they're needed or not, because they might be brought in by a transitive framework reference. We didn't want to ship the Windows targeting packs with the non-Windows SDK builds, but we also didn't want a vanilla Console or ASP.NET Core app to automatically download these targeting and runtime packs the first time you build. The `EnableWindowsTargeting` property enables them to only be downloaded if you opt in.

NETSDK1112: The runtime pack was not downloaded

9/20/2022 • 2 minutes to read • [Edit Online](#)

A runtime pack that your project requires was not downloaded. The full error message is similar to the following example:

```
The runtime pack for <Runtime> was not downloaded. Try running a NuGet restore with the  
RuntimeIdentifier '<RID>'.
```

This can happen in the following scenarios:

- The publish parameters are different than what's in the project file. Review publish parameters and change any instances where publish parameters differ from the project file.
- You used Debug configuration to build an app that is set up to require Release configuration. Try building with Release configuration.

NETSDK1130: Can't reference a Windows Metadata component directly

9/20/2022 • 2 minutes to read • [Edit Online](#)

NETSDK1130 indicates that you're trying to reference a Windows Metadata component directly from an app that targets .NET 5 or later. The full error message is similar to the following example:

```
<Component name> cannot be referenced. Referencing a Windows Metadata component directly when targeting .NET 5 or higher is not supported.
```

To resolve this error:

- Remove references to the [Microsoft.Windows.SDK.Contracts package](#). Instead, specify the version of the Windows APIs that you want to access via the `TargetFramework` property of the project. For example:

```
<TargetFramework>net5.0-windows10.0.19041.0</TargetFramework>
```

- If you're consuming a third-party runtime component that's defined in a `.winmd` file, add a reference to the [Microsoft.Windows.CsWinRT NuGet package](#). For information on how to generate the C# projection, see the [C#/WinRT](#) documentation.

For more information, see [Built-in support for WinRT is removed from .NET](#) and [Call Windows Runtime APIs in desktop apps](#).

NETSDK1135: SupportedOSPlatformVersion can't be higher than TargetPlatformVersion

9/20/2022 • 2 minutes to read • [Edit Online](#)

The value that your project file specifies for `SupportedOSPlatformVersion` is too high. The full error message is similar to the following example:

`SupportedOSPlatformVersion <version> cannot be higher than TargetPlatformVersion <version>.`

Check your project files for `SupportedOSPlatformVersion`, `TargetPlatformVersion` (TPV) is inferred from the `TargetFramework` value. For example, `net6.0-windows10.0.19041` will set the TPV to be `10.0.19041`. .NET uses `TargetPlatformVersion` to determine which APIs are available at compile time.

NETSDK1136: The target framework must be Windows

9/20/2022 • 2 minutes to read • [Edit Online](#)

If `UseWindowsForms` or `UseWPF` is `true`, .NET assumes that your project is a Windows app, and so the platform has to be set to Windows. This error can happen if you have a project-to-project reference where one is set to Windows and the other is not. The full error message is similar to the following example:

The target platform must be set to Windows (usually by including `-windows` in the `TargetFramework` property) when using Windows Forms or WPF, or referencing projects or packages that do so.

For example, set `TargetFramework` to `net6.0-windows`, as shown in this project file:

```
<Project Sdk="Microsoft.NET.Sdk.Web">
  <PropertyGroup>
    <TargetFramework>net6.0-windows</TargetFramework>
  </PropertyGroup>
</Project>
```

NETSDK1137: Don't use the Microsoft.NET.Sdk.WindowsDesktop SDK

9/20/2022 • 2 minutes to read • [Edit Online](#)

NETSDK1137 indicates that your project specifies an outdated project SDK. The full error message is similar to the following example:

It is no longer necessary to use the Microsoft.NET.Sdk.WindowsDesktop SDK. Consider changing the Sdk attribute of the root Project element to 'Microsoft.NET.Sdk'.

To resolve this error, modify your project file to have `<Project Sdk="Microsoft.NET.Sdk">` instead of the `WindowsDesktop` SDK.

NETSDK1138: The target framework is out of support

9/20/2022 • 2 minutes to read • [Edit Online](#)

NETSDK1138 indicates that your project targets a version of the framework that is out of support. The full error message is similar to the following example:

```
The target framework '<framework>' is out of support and will not receive security updates in the future.  
Please refer to https://aka.ms/dotnet-core-support for more information about the support policy.
```

Out-of-support versions include 1.0, 1.1, 2.0, 2.1, 2.2, and 3.0. In November 2022, .NET 5 will be added to this list.

To resolve this error, change your project to target a supported version of .NET Core or .NET 5+.

If you want to suppress the message without targeting a later framework, set the MSBuild property `CheckEolTargetFramework` to false. You can set it in the project file or by passing `/p:CheckEolTargetFramework=false` to a .NET CLI command, such as `dotnet build`. Here's an example project file:

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    <OutputType>Exe</OutputType>  
    <TargetFramework>netcoreapp3.0</TargetFramework>  
    <CheckEolTargetFramework>false</CheckEolTargetFramework>  
  </PropertyGroup>  
</Project>
```

Here's an example .NET CLI command:

```
dotnet build /p:CheckEolTargetFramework=false
```

See also

- <https://aka.ms/dotnet-core-support>

NETSDK1141: Unable to resolve the .NET SDK version as specified in the `global.json`

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 5.0 SDK and later versions

There was a problem with the version of the SDK specified in the `global.json` file.

NETSDK1141: Unable to resolve the .NET SDK version as specified in the `global.json` located at C:\path\global.json.

Cause

- The SDK version in the `global.json` file is incorrectly specified.
- The SDK version specified in the `global.json` file was not installed.
- The SDK version specified in `global.json` could not be found, due to an incorrect path.

How to fix the error

- Install the SDK version requested in `global.json`.
- Specify a different SDK version in `global.json`.
- Check for typos or other problems in `global.json`. See [global.json](#) for the correct structure of that file.
- Delete `global.json`. In this case, the latest installed version of the SDK is used.

When you're working on a shared project, developers need to agree on the SDK version that will be used for the project. Without `global.json`, if developers on different dev machines don't have the same SDK versions, the build environment might be inconsistent across the dev team. To solve this, the SDK version can be specified in `global.json` and checked into source control as a common file which would be the same for all developers and ensure that the same SDK version is being used in all development environments. Therefore, to resolve this issue in a shared project, you might need to agree as a team on a particular SDK version and update all the code to use this version.

See also

[global.json How to check that the .NET SDK is installed](#)

NETSDK1145: Targeting or apphost pack missing

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET 5.0.100 SDK and later versions

When the .NET SDK issues error NETSDK1145, the targeting or apphost pack is not installed and NuGet package restore is not supported. This is typically caused by having a newer SDK than the one included in Visual Studio for C++/CLI projects. Upgrade Visual Studio, remove *global.json* if it specifies a certain SDK version, and uninstall the newer SDK. Alternatively, you could override the targeting or apphost version. Find the version that exists under the pack directory from the error message and matches the target framework of the project. Add the following to the project:

For apphost pack

```
<ItemGroup>
  <KnownAppHostPack Update="@{KnownAppHostPack}">
    <AppHostPackVersion Condition="'%(TargetFramework)' ==
'TARGETFRAMEWORK'">EXISTINGVERSION</AppHostPackVersion>
  </KnownAppHostPack>
</ItemGroup>
```

For targeting pack

```
<ItemGroup>
  <KnownFrameworkReference Update="@{KnownFrameworkReference}">
    <TargetingPackVersion Condition="'%(TargetFramework)' ==
'TARGETFRAMEWORK'">EXISTINGVERSION</TargetingPackVersion>
  </KnownFrameworkReference>
</ItemGroup>
```

NETSDK1147: Missing workload for specified target framework

9/20/2022 • 2 minutes to read • [Edit Online](#)

This error is caused by trying to compile a project that requires an optional workload, but you don't have the workload installed. The full error message is similar to the following example:

NETSDK1147: To build this project, the following workloads must be installed: <workload ID>

To install these workloads, run the following command: dotnet workload install <workload ID>

For example, if your project targets `net6.0-android`, you might have to run the `dotnet workload install` command and specify workload ID `android`:

```
dotnet workload install android
```

For more information, see [dotnet workload install](#).

NETSDK1149: Built-in WinRT support not provided in .NET 5 and later

9/20/2022 • 2 minutes to read • [Edit Online](#)

NETSDK1149 indicates that you're trying to reference a component that requires WinRT in an application that targets .NET 5 or a later version. These .NET versions don't have built-in support for WinRT. The full error message is similar to the following example:

<Component name> cannot be referenced because it uses built-in support for WinRT, which is no longer supported in .NET 5 and higher. An updated version of the component supporting .NET 5 is needed.

If your application calls Windows Runtime APIs, resolve this error by changing the application's Target Framework Moniker (TFM) to a value that targets Windows 10. For more information, see [Call Windows Runtime APIs in desktop apps](#).

If your application calls a 3rd party WinRT component, get an updated version of the component that supports .NET 5. You can generate an updated version by using [C#/WinRT](#).

For more information, see [Built-in support for WinRT is removed from .NET](#).

NETSDK1174: -p abbreviation for --project in dotnet run is deprecated

9/20/2022 • 2 minutes to read • [Edit Online](#)

The full error message is similar to the following example:

The abbreviation of -p for --project is deprecated. Please use --project.

The use of `-p` in `dotnet run` is deprecated because of the close relationship `dotnet run` has with `dotnet build` and `dotnet publish`. In `dotnet build` and `dotnet publish`, `p` is used to set MSBuild properties. This deprecation is the first step in aligning abbreviations for these three commands.

For more information, see [-p option for dotnet run is deprecated](#).

NETSDK1182: Targeting .NET 6.0 or higher in Visual Studio 2019 is not supported

9/20/2022 • 2 minutes to read • [Edit Online](#)

NETSDK1182 indicates that you're trying to open a .NET 6+ project in Visual Studio 2019. Or you might be trying to build from the command line using MSBuild from Visual Studio 2019. The full error message is similar to the following example:

Targeting .NET 6.0 or higher in Visual Studio 2019 is not supported.

To resolve this error switch to Visual Studio 2022, or change your project to target .NET 5 or earlier.

See also

- [Overview of .NET, MSBuild, and Visual Studio versioning](#)

.NET CLI overview

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 2.1 SDK and later versions

The .NET command-line interface (CLI) is a cross-platform toolchain for developing, building, running, and publishing .NET applications.

The .NET CLI is included with the [.NET SDK](#). To learn how to install the .NET SDK, see [Install .NET Core](#).

CLI commands

The following commands are installed by default:

Basic commands

- [new](#)
- [restore](#)
- [build](#)
- [publish](#)
- [run](#)
- [test](#)
- [vstest](#)
- [pack](#)
- [migrate](#)
- [clean](#)
- [sln](#)
- [help](#)
- [store](#)

Project modification commands

- [add package](#)
- [add reference](#)
- [remove package](#)
- [remove reference](#)
- [list reference](#)

Advanced commands

- [nuget delete](#)
- [nuget locals](#)
- [nuget push](#)
- [msbuild](#)
- [dotnet install script](#)

Tool management commands

- [tool install](#)
- [tool list](#)
- [tool update](#)

- `tool restore` Available since .NET Core SDK 3.0.
- `tool run` Available since .NET Core SDK 3.0.
- `tool uninstall`

Tools are console applications that are installed from NuGet packages and are invoked from the command prompt. You can write tools yourself or install tools written by third parties. Tools are also known as global tools, tool-path tools, and local tools. For more information, see [.NET tools overview](#).

Command structure

CLI command structure consists of **the driver ("dotnet")**, **the command**, and possibly command **arguments** and **options**. You see this pattern in most CLI operations, such as creating a new console app and running it from the command line as the following commands show when executed from a directory named *my_app*:

```
dotnet new console  
dotnet build --output ./build_output  
dotnet ./build_output/my_app.dll
```

Driver

The driver is named **dotnet** and has two responsibilities, either running a [framework-dependent app](#) or executing a command.

To run a framework-dependent app, specify the app after the driver, for example, `dotnet /path/to/my_app.dll`. When executing the command from the folder where the app's DLL resides, simply execute `dotnet my_app.dll`. If you want to use a specific version of the .NET Runtime, use the `--fx-version <VERSION>` option (see the [dotnet command reference](#)).

When you supply a command to the driver, `dotnet.exe` starts the CLI command execution process. For example:

```
dotnet build
```

First, the driver determines the version of the SDK to use. If there is no [global.json](#) file, the latest version of the SDK available is used. This might be either a preview or stable version, depending on what is latest on the machine. Once the SDK version is determined, it executes the command.

Command

The command performs an action. For example, `dotnet build` builds code. `dotnet publish` publishes code. The commands are implemented as a console application using a `dotnet {command}` convention.

Arguments

The arguments you pass on the command line are the arguments to the command invoked. For example, when you execute `dotnet publish my_app.csproj`, the `my_app.csproj` argument indicates the project to publish and is passed to the `publish` command.

Options

The options you pass on the command line are the options to the command invoked. For example, when you execute `dotnet publish --output /build_output`, the `--output` option and its value are passed to the `publish` command.

See also

- [dotnet/sdk GitHub repository](#)
- [.NET installation guide](#)

dotnet command

9/20/2022 • 7 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 3.1 SDK and later versions

Name

`dotnet` - The generic driver for the .NET CLI.

Synopsis

To get information about the available commands and the environment:

```
dotnet [--version] [--info] [--list-runtimes] [--list-sdks]  
dotnet -h|--help
```

To run a command (requires SDK installation):

```
dotnet <COMMAND> [-d|--diagnostics] [-h|--help] [--verbosity <LEVEL>]  
[command-options] [arguments]
```

To run an application:

```
dotnet [--additionalprobingpath <PATH>] [--additional-deps <PATH>]  
[--fx-version <VERSION>] [--roll-forward <SETTING>]  
<PATH_TO_APPLICATION> [arguments]  
  
dotnet exec [--additionalprobingpath] [--additional-deps <PATH>]  
[--depsfile <PATH>]  
[--fx-version <VERSION>] [--roll-forward <SETTING>]  
[--runtimeconfig <PATH>]  
<PATH_TO_APPLICATION> [arguments]
```

Description

The `dotnet` command has two functions:

- It provides commands for working with .NET projects.

For example, `dotnet build` builds a project. Each command defines its own options and arguments. All commands support the `--help` option for printing out brief documentation about how to use the command.

- It runs .NET applications.

You specify the path to an application `.dll` file to run the application. To run the application means to find and execute the entry point, which in the case of console apps is the `Main` method. For example, `dotnet myapp.dll` runs the `myapp` application. See [.NET application deployment](#) to learn about deployment options.

Options

Different options are available for:

- Displaying information about the environment.
- Running a command.
- Running an application.

Options for displaying environment information and available commands

The following options are available when `dotnet` is used by itself, without specifying a command or an application to run. For example, `dotnet --info` or `dotnet --version`. They print out information about the environment.

- `--info`

Prints out detailed information about a .NET installation and the machine environment, such as the current operating system, and commit SHA of the .NET version.

- `--version`

Prints out the version of the .NET SDK used by `dotnet` commands, which may be affected by a *global.json* file. Available only when the SDK is installed.

- `--list-runtimes`

Prints out a list of the installed .NET runtimes. An x86 version of the SDK lists only x86 runtimes, and an x64 version of the SDK lists only x64 runtimes.

- `--list-sdks`

Prints out a list of the installed .NET SDKs.

- `-?|-h|--help`

Prints out a list of available commands.

Options for running a command

The following options are for `dotnet` with a command. For example, `dotnet build --help` or `dotnet build --verbosity diagnostic`.

- `-d|--diagnostics`

Enables diagnostic output.

- `-v|--verbosity <LEVEL>`

Sets the verbosity level of the command. Allowed values are `q[uiet]`, `m[inimal]`, `n[ormal]`, `d[etailed]`, and `diag[nostic]`. Not supported in every command. See specific command page to determine if this option is available.

- `-?|-h|--help`

Prints out documentation for a given command. For example, `dotnet build --help` displays help for the `build` command.

- `command options`

Each command defines options specific to that command. See specific command page for a list of available options.

Options for running an application

The following options are available when `dotnet` runs an application. For example,

```
dotnet --roll-forward Major myapp.dll .
```

- `--additionalprobingpath <PATH>`

Path containing probing policy and assemblies to probe. Repeat the option to specify multiple paths.

- `--additional-deps <PATH>`

Path to an additional `.deps.json` file. A `.deps.json` file contains a list of dependencies, compilation dependencies, and version information used to address assembly conflicts. For more information, see [Runtime Configuration Files](#) on GitHub.

- `--roll-forward <SETTING> **`

Controls how roll forward is applied to the app. The `<SETTING>` can be one of the following values. If not specified, `Minor` is the default.

- `LatestPatch` - Roll forward to the highest patch version. This disables minor version roll forward.
- `Minor` - Roll forward to the lowest higher minor version, if requested minor version is missing. If the requested minor version is present, then the `LatestPatch` policy is used.
- `Major` - Roll forward to lowest higher major version, and lowest minor version, if requested major version is missing. If the requested major version is present, then the `Minor` policy is used.
- `LatestMinor` - Roll forward to highest minor version, even if requested minor version is present. Intended for component hosting scenarios.
- `LatestMajor` - Roll forward to highest major and highest minor version, even if requested major is present. Intended for component hosting scenarios.
- `Disable` - Don't roll forward. Only bind to specified version. This policy isn't recommended for general use because it disables the ability to roll forward to the latest patches. This value is only recommended for testing.

With the exception of `Disable`, all settings will use the highest available patch version.

Roll forward behavior can also be configured in a project file property, a runtime configuration file property, and an environment variable. For more information, see [Major-version runtime roll forward](#).

- `--fx-version <VERSION>`

Version of the .NET runtime to use to run the application.

This option overrides the version of the first framework reference in the application's `.runtimeconfig.json` file. This means it only works as expected if there's just one framework reference. If the application has more than one framework reference, using this option may cause errors.

Options for running an application with the `exec` command

The following options are available only when `dotnet` runs an application by using the `exec` command. For example, `dotnet exec --runtimeconfig myapp.runtimeconfig.json myapp.dll .`

- `--depsfile <PATH>`

Path to a `deps.json` file. A `deps.json` file is a configuration file that contains information about dependencies necessary to run the application. This file is generated by the .NET SDK.

- `--runtimeconfig <PATH>`

Path to a `runtimeconfig.json` file. A `runtimeconfig.json` file contains run-time settings and is typically

named `<applicationname>.runtimeconfig.json`. For more information, see [.NET runtime configuration settings](#).

dotnet commands

General

COMMAND	FUNCTION
<code>dotnet build</code>	Builds a .NET application.
<code>dotnet build-server</code>	Interacts with servers started by a build.
<code>dotnet clean</code>	Clean build outputs.
<code>dotnet exec</code>	Runs a .NET application.
<code>dotnet help</code>	Shows more detailed documentation online for the command.
<code>dotnet migrate</code>	Migrates a valid Preview 2 project to a .NET Core SDK 1.0 project.
<code>dotnet msbuild</code>	Provides access to the MSBuild command line.
<code>dotnet new</code>	Initializes a C# or F# project for a given template.
<code>dotnet pack</code>	Creates a NuGet package of your code.
<code>dotnet publish</code>	Publishes a .NET framework-dependent or self-contained application.
<code>dotnet restore</code>	Restores the dependencies for a given application.
<code>dotnet run</code>	Runs the application from source.
<code>dotnet sdk check</code>	Shows up-to-date status of installed SDK and Runtime versions.
<code>dotnet sln</code>	Options to add, remove, and list projects in a solution file.
<code>dotnet store</code>	Stores assemblies in the runtime package store.
<code>dotnet test</code>	Runs tests using a test runner.

Project references

COMMAND	FUNCTION
<code>dotnet add reference</code>	Adds a project reference.
<code>dotnet list reference</code>	Lists project references.
<code>dotnet remove reference</code>	Removes a project reference.

NuGet packages

COMMAND	FUNCTION
dotnet add package	Adds a NuGet package.
dotnet remove package	Removes a NuGet package.

NuGet commands

COMMAND	FUNCTION
dotnet nuget delete	Deletes or unlists a package from the server.
dotnet nuget push	Pushes a package to the server and publishes it.
dotnet nuget locals	Clears or lists local NuGet resources such as http-request cache, temporary cache, or machine-wide global packages folder.
dotnet nuget add source	Adds a NuGet source.
dotnet nuget disable source	Disables a NuGet source.
dotnet nuget enable source	Enables a NuGet source.
dotnet nuget list source	Lists all configured NuGet sources.
dotnet nuget remove source	Removes a NuGet source.
dotnet nuget update source	Updates a NuGet source.

Workload commands

COMMAND	FUNCTION
dotnet workload install	Installs an optional workload.
dotnet workload list	Lists all installed workloads.
dotnet workload repair	Repairs all installed workloads.
dotnet workload search	List selected workloads or all available workloads.
dotnet workload uninstall	Uninstalls a workload.
dotnet workload update	Reinstalls all installed workloads.

Global, tool-path, and local tools commands

Tools are console applications that are installed from NuGet packages and are invoked from the command prompt. You can write tools yourself or install tools written by third parties. Tools are also known as global tools, tool-path tools, and local tools. For more information, see [.NET tools overview](#).

COMMAND	FUNCTION
<code>dotnet tool install</code>	Installs a tool on your machine.
<code>dotnet tool list</code>	Lists all global, tool-path, or local tools currently installed on your machine.
<code>dotnet tool search</code>	Searches NuGet.org for tools that have the specified search term in their name or metadata.
<code>dotnet tool uninstall</code>	Uninstalls a tool from your machine.
<code>dotnet tool update</code>	Updates a tool that is installed on your machine.

Additional tools

The following additional tools are available as part of the .NET SDK:

TOOL	FUNCTION
<code>dev-certs</code>	Creates and manages development certificates.
<code>ef</code>	Entity Framework Core command-line tools.
<code>user-secrets</code>	Manages development user secrets.
<code>watch</code>	A file watcher that restarts or hot reloads an application when it detects changes in the source code.

For more information about each tool, type `dotnet <tool-name> --help`.

Examples

Create a new .NET console application:

```
dotnet new console
```

Build a project and its dependencies in a given directory:

```
dotnet build
```

Run an application:

```
dotnet exec myapp.dll
```

```
dotnet myapp.dll
```

See also

- [Environment variables used by .NET SDK, .NET CLI, and .NET runtime](#)
- [Runtime Configuration Files](#)

- .NET runtime configuration settings

dotnet add package

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 3.1 SDK and later versions

Name

`dotnet add package` - Adds or updates a package reference in a project file.

Synopsis

```
dotnet add [<PROJECT>] package <PACKAGE_NAME>
  [-f|--framework <FRAMEWORK>] [--interactive]
  [-n|--no-restore] [--package-directory <PACKAGE_DIRECTORY>]
  [--prerelease] [-s|--source <SOURCE>] [-v|--version <VERSION>]

dotnet add package -h|--help
```

Description

The `dotnet add package` command provides a convenient option to add or update a package reference in a project file. When you run the command, there's a compatibility check to ensure the package is compatible with the frameworks in the project. If the check passes and the package isn't referenced in the project file, a `<PackageReference>` element is added to the project file. If the check passes and the package is already referenced in the project file, the `<PackageReference>` element is updated to the latest compatible version. After the project file is updated, `dotnet restore` is run.

For example, adding `Microsoft.EntityFrameworkCore` to `ToDo.csproj` produces output similar to the following example:

```
Determining projects to restore...
Writing C:\Users\username\AppData\Local\Temp\tmp24A8.tmp
info : Adding PackageReference for package 'Microsoft.EntityFrameworkCore' into project
'C:\ToDo\ToDo.csproj'.
info : CACHE https://api.nuget.org/v3/registration5-gz-semver2/microsoft.entityframeworkcore/index.json
info : GET https://pkgs.dev.azure.com/dnceng/9ee6d478-d288-47f7-aacc-f6e6d082ae6d/_packaging/516521bf-
6417-457e-9a9c-0a4bdfde03e7/nuget/v3/registrations2-semver2/microsoft.entityframeworkcore/index.json
info : CACHE https://api.nuget.org/v3/registration5-gz-semver2/microsoft.entityframeworkcore/page/0.0.1-
alpha/3.1.3.json
info : CACHE https://api.nuget.org/v3/registration5-gz-
semver2/microsoft.entityframeworkcore/page/3.1.4/7.0.0-preview.2.22153.1.json
info : CACHE https://api.nuget.org/v3/registration5-gz-semver2/microsoft.entityframeworkcore/page/7.0.0-
preview.3.22175.1/7.0.0-preview.3.22175.1.json
info : NotFound https://pkgs.dev.azure.com/dnceng/9ee6d478-d288-47f7-aacc-
f6e6d082ae6d/_packaging/516521bf-6417-457e-9a9c-0a4bdfde03e7/nuget/v3/registrations2-
semver2/microsoft.entityframeworkcore/index.json 257ms
info : Restoring packages for C:\ToDo\ToDo.csproj...
info : Package 'Microsoft.EntityFrameworkCore' is compatible with all the specified frameworks in project
'C:\ToDo\ToDo.csproj'.
info : PackageReference for package 'Microsoft.EntityFrameworkCore' version '6.0.4' added to file
'C:\ToDo\ToDo.csproj'.
info : Writing assets file to disk. Path: C:\ToDo\obj\project.assets.json
log : Restored C:\ToDo\ToDo.csproj (in 171 ms).
```

The `ToDo.csproj` file now contains a `<PackageReference>` element for the referenced package.

```
<PackageReference Include="Microsoft.EntityFrameworkCore" Version="6.0.4" />
```

Implicit restore

You don't have to run `dotnet restore` because it's run implicitly by all commands that require a restore to occur, such as `dotnet new`, `dotnet build`, `dotnet run`, `dotnet test`, `dotnet publish`, and `dotnet pack`. To disable implicit restore, use the `--no-restore` option.

The `dotnet restore` command is still useful in certain scenarios where explicitly restoring makes sense, such as [continuous integration builds in Azure DevOps Services](#) or in build systems that need to explicitly control when the restore occurs.

For information about how to manage NuGet feeds, see the [dotnet restore](#) documentation.

Arguments

- `PROJECT`

Specifies the project file. If not specified, the command searches the current directory for one.

- `PACKAGE_NAME`

The package reference to add.

Options

- `-f|--framework <FRAMEWORK>`

Adds a package reference only when targeting a specific [framework](#).

- `-?|-h|--help`

Prints out a description of how to use the command.

- `--interactive`

Allows the command to stop and wait for user input or action. For example, to complete authentication.

- `-n|--no-restore`

Adds a package reference without performing a restore preview and compatibility check.

- `--package-directory <PACKAGE_DIRECTORY>`

The directory where to restore the packages. The default package restore location is `%userprofile%\.nuget\packages` on Windows and `~/.nuget/packages` on macOS and Linux. For more information, see [Managing the global packages, cache, and temp folders in NuGet](#).

- `--prerelease`

Allows prerelease packages to be installed. Available since .NET Core 5 SDK

- `-s|--source <SOURCE>`

The URI of the NuGet package source to use during the restore operation.

- `-v|--version <VERSION>`

Version of the package. See [NuGet package versioning](#).

Examples

- Add `Microsoft.EntityFrameworkCore` NuGet package to a project:

```
dotnet add package Microsoft.EntityFrameworkCore
```

- Add a specific version of a package to a project:

```
dotnet add ToDo.csproj package Microsoft.Azure.DocumentDB.Core -v 1.0.0
```

- Add a package using a specific NuGet source:

```
dotnet add package Microsoft.AspNetCore.StaticFiles -s https://dotnet.myget.org/F/dotnet-core/api/v3/index.json
```

See also

- [Managing the global packages, cache, and temp folders in NuGet](#)
- [NuGet package versioning](#)

dotnet list package

9/20/2022 • 3 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 3.1 SDK and later versions

Name

`dotnet list package` - Lists the package references for a project or solution.

Synopsis

```
dotnet list [<PROJECT>|<SOLUTION>] package [--config <SOURCE>]
  [--deprecated]
  [--framework <FRAMEWORK>] [--highest-minor] [--highest-patch]
  [--include-prerelease] [--include-transitive] [--interactive]
  [--outdated] [--source <SOURCE>] [-v|--verbosity <LEVEL>]
  [--vulnerable]

dotnet list package -h|--help
```

Description

The `dotnet list package` command provides a convenient option to list all NuGet package references for a specific project or a solution. You first need to build the project in order to have the assets needed for this command to process. The following example shows the output of the `dotnet list package` command for the [SentimentAnalysis](#) project:

```
Project 'SentimentAnalysis' has the following package references
[netcoreapp2.1]:
  Top-level Package      Requested   Resolved
  > Microsoft.ML          1.4.0       1.4.0
  > Microsoft.NETCore.App (A) [2.1.0, )  2.1.0

(A) : Auto-referenced package.
```

The **Requested** column refers to the package version specified in the project file and can be a range. The **Resolved** column lists the version that the project is currently using and is always a single value. The packages displaying an `(A)` right next to their names represent implicit package references that are inferred from your project settings (`Sdk` type, or `<TargetFramework>` or `<TargetFrameworks>` property).

Use the `--outdated` option to find out if there are newer versions available of the packages you're using in your projects. By default, `--outdated` lists the latest stable packages unless the resolved version is also a prerelease version. To include prerelease versions when listing newer versions, also specify the `--include-prerelease` option. To update a package to the latest version, use [dotnet add package](#).

The following example shows the output of the `dotnet list package --outdated --include-prerelease` command for the same project as the previous example:

```
The following sources were used:  
https://api.nuget.org/v3/index.json  
C:\Program Files (x86)\Microsoft SDKs\NuGetPackages\
```

```
Project `SentimentAnalysis` has the following updates to its packages  
[netcoreapp2.1]:  
Top-level Package Requested Resolved Latest  
> Microsoft.ML 1.4.0 1.4.0 1.5.0-preview
```

If you need to find out whether your project has transitive dependencies, use the `--include-transitive` option. Transitive dependencies occur when you add a package to your project that in turn relies on another package. The following example shows the output from running the `dotnet list package --include-transitive` command for the [HelloPlugin](#) project, which displays top-level packages and the packages they depend on:

```
Project 'HelloPlugin' has the following package references  
[netcoreapp3.0]:  
Transitive Package Resolved  
> PluginBase 1.0.0
```

Arguments

[PROJECT](#) | [SOLUTION](#)

The project or solution file to operate on. If not specified, the command searches the current directory for one. If more than one solution or project is found, an error is thrown.

Options

- `--config <SOURCE>`
The NuGet sources to use when searching for newer packages. Requires the `--outdated` option.
- `--deprecated`
Displays packages that have been deprecated.
- `--framework <FRAMEWORK>`
Displays only the packages applicable for the specified [target framework](#). To specify multiple frameworks, repeat the option multiple times. For example: `--framework net6.0 --framework netstandard2.0`.
- `-?|-h|--help`
Prints out a description of how to use the command.
- `--highest-minor`
Considers only the packages with a matching major version number when searching for newer packages. Requires the `--outdated` or `--deprecated` option.
- `--highest-patch`
Considers only the packages with a matching major and minor version numbers when searching for newer packages. Requires the `--outdated` or `--deprecated` option.
- `--include-prerelease`
Considers packages with prerelease versions when searching for newer packages. Requires the

--outdated or --deprecated option.

- --include-transitive

Lists transitive packages, in addition to the top-level packages. When specifying this option, you get a list of packages that the top-level packages depend on.

- --interactive

Allows the command to stop and wait for user input or action. For example, to complete authentication.

Available since .NET Core 3.0 SDK.

- --outdated

Lists packages that have newer versions available.

- -s|--source <SOURCE>

The NuGet sources to use when searching for newer packages. Requires the --outdated or --deprecated option.

- -v|--verbosity <LEVEL>

Sets the verbosity level of the command. Allowed values are q[uiet], m[inimal], n[ormal], d[etailed], and diag[nostic]. The default is minimal. For more information, see [LoggerVerbosity](#).

- --vulnerable

Lists packages that have known vulnerabilities. Cannot be combined with --deprecated or --outdated options. Nuget.org is the source of information about vulnerabilities. For more information, see [Vulnerabilities](#) and [How to Scan NuGet Packages for Security Vulnerabilities](#).

Examples

- List package references of a specific project:

```
dotnet list SentimentAnalysis.csproj package
```

- List package references that have newer versions available, including prerelease versions:

```
dotnet list package --outdated --include-prerelease
```

- List package references for a specific target framework:

```
dotnet list package --framework netcoreapp3.0
```

dotnet remove package

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 3.1 SDK and later versions

Name

`dotnet remove package` - Removes package reference from a project file.

Synopsis

```
dotnet remove [<PROJECT>] package <PACKAGE_NAME>
```

```
dotnet remove package -h|--help
```

Description

The `dotnet remove package` command provides a convenient option to remove a NuGet package reference from a project.

Arguments

`PROJECT`

Specifies the project file. If not specified, the command searches the current directory for one.

`PACKAGE_NAME`

The package reference to remove.

Options

• `-?|-h|--help`

Prints out a description of how to use the command.

Examples

- Remove `Newtonsoft.Json` NuGet package from a project in the current directory:

```
dotnet remove package Newtonsoft.Json
```

dotnet add reference

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 3.1 SDK and later versions

Name

`dotnet add reference` - Adds project-to-project (P2P) references.

Synopsis

```
dotnet add [<PROJECT>] reference [-f|--framework <FRAMEWORK>]
```

```
[--interactive] <PROJECT_REFERENCES>
```

```
dotnet add reference -h|--help
```

Description

The `dotnet add reference` command provides a convenient option to add project references to a project. After running the command, the `<ProjectReference>` elements are added to the project file.

```
<ItemGroup>
  <ProjectReference Include="app.csproj" />
  <ProjectReference Include=".\\lib2\\lib2.csproj" />
  <ProjectReference Include=".\\lib1\\lib1.csproj" />
</ItemGroup>
```

Arguments

- `PROJECT`

Specifies the project file. If not specified, the command searches the current directory for one.

- `PROJECT_REFERENCES`

Project-to-project (P2P) references to add. Specify one or more projects. [Glob patterns](#) are supported on Unix/Linux-based systems.

Options

- `-f|--framework <FRAMEWORK>`

Adds project references only when targeting a specific [framework](#) using the TFM format.

- `-?|-h|--help`

Prints out a description of how to use the command.

- `--interactive`

Allows the command to stop and wait for user input or action. For example, to complete authentication. Available since .NET Core 3.0 SDK.

Examples

- Add a project reference:

```
dotnet add app/app.csproj reference lib/lib.csproj
```

- Add multiple project references to the project in the current directory:

```
dotnet add reference lib1/lib1.csproj lib2/lib2.csproj
```

- Add multiple project references using a globbing pattern on Linux/Unix:

```
dotnet add app/app.csproj reference **/*.csproj
```

dotnet list reference

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 3.1 SDK and later versions

Name

`dotnet list reference` - Lists project-to-project references.

Synopsis

```
dotnet list [<PROJECT>] reference
```

```
dotnet list -h|--help
```

Description

The `dotnet list reference` command provides a convenient option to list project references for a given project.

Arguments

- `PROJECT`

The project file to operate on. If a file is not specified, the command will search the current directory for one.

Options

- `-?|-h|--help`

Prints out a description of how to use the command.

Examples

- List the project references for the specified project:

```
dotnet list app/app.csproj reference
```

- List the project references for the project in the current directory:

```
dotnet list reference
```

dotnet remove reference

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 3.1 SDK and later versions

Name

`dotnet remove reference` - Removes project-to-project (P2P) references.

Synopsis

```
dotnet remove [<PROJECT>] reference [-f|--framework <FRAMEWORK>]  
<PROJECT_REFERENCES>  
  
dotnet remove reference -h|--help
```

Description

The `dotnet remove reference` command provides a convenient option to remove project references from a project.

Arguments

`PROJECT`

Target project file. If not specified, the command searches the current directory for one.

`PROJECT_REFERENCES`

Project-to-project (P2P) references to remove. You can specify one or multiple projects. [Glob patterns](#) are supported on Unix/Linux based terminals.

Options

• `-?|-h|--help`

Prints out a description of how to use the command.

• `-f|--framework <FRAMEWORK>`

Removes the reference only when targeting a specific [framework](#) using the TFM format.

Examples

- Remove a project reference from the specified project:

```
dotnet remove app/app.csproj reference lib/lib.csproj
```

- Remove multiple project references from the project in the current directory:

```
dotnet remove reference lib1/lib1.csproj lib2/lib2.csproj
```

- Remove multiple project references using a glob pattern on Unix/Linux:

```
dotnet remove app/app.csproj reference **/*.csproj`
```

dotnet build

9/20/2022 • 6 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 3.1 SDK and later versions

Name

`dotnet build` - Builds a project and all of its dependencies.

Synopsis

```
dotnet build [<PROJECT>|<SOLUTION>] [-a|--arch <ARCHITECTURE>]
  [-c|--configuration <CONFIGURATION>] [-f|--framework <FRAMEWORK>]
  [--force] [--interactive] [--no-dependencies] [--no-incremental]
  [--no-restore] [--nologo] [--no-self-contained] [--os <OS>]
  [-o|--output <OUTPUT_DIRECTORY>] [-r|--runtime <RUNTIME_IDENTIFIER>]
  [--self-contained [true|false]] [--source <SOURCE>]
  [-v|--verbosity <LEVEL>] [--version-suffix <VERSION_SUFFIX>]

dotnet build -h|--help
```

Description

The `dotnet build` command builds the project and its dependencies into a set of binaries. The binaries include the project's code in Intermediate Language (IL) files with a `.dll` extension. Depending on the project type and settings, other files may be included, such as:

- An executable that can be used to run the application, if the project type is an executable targeting .NET Core 3.0 or later.
- Symbol files used for debugging with a `.pdb` extension.
- A `.deps.json` file, which lists the dependencies of the application or library.
- A `.runtimeconfig.json` file, which specifies the shared runtime and its version for an application.
- Other libraries that the project depends on (via project references or NuGet package references).

For executable projects targeting versions earlier than .NET Core 3.0, library dependencies from NuGet are typically NOT copied to the output folder. They're resolved from the NuGet global packages folder at run time. With that in mind, the product of `dotnet build` isn't ready to be transferred to another machine to run. To create a version of the application that can be deployed, you need to publish it (for example, with the `dotnet publish` command). For more information, see [.NET Application Deployment](#).

For executable projects targeting .NET Core 3.0 and later, library dependencies are copied to the output folder. This means that if there isn't any other publish-specific logic (such as Web projects have), the build output should be deployable.

Implicit restore

Building requires the `project.assets.json` file, which lists the dependencies of your application. The file is created when `dotnet restore` is executed. Without the assets file in place, the tooling can't resolve reference assemblies, which results in errors.

You don't have to run `dotnet restore` because it's run implicitly by all commands that require a restore to occur, such as `dotnet new`, `dotnet build`, `dotnet run`, `dotnet test`, `dotnet publish`, and `dotnet pack`. To disable

implicit restore, use the `--no-restore` option.

The `dotnet restore` command is still useful in certain scenarios where explicitly restoring makes sense, such as [continuous integration builds in Azure DevOps Services](#) or in build systems that need to explicitly control when the restore occurs.

For information about how to manage NuGet feeds, see the [dotnet restore documentation](#).

This command supports the `dotnet restore` options when passed in the long form (for example, `--source`). Short form options, such as `-s`, are not supported.

Executable or library output

Whether the project is executable or not is determined by the `<OutputType>` property in the project file. The following example shows a project that produces executable code:

```
<PropertyGroup>
  <OutputType>Exe</OutputType>
</PropertyGroup>
```

To produce a library, omit the `<OutputType>` property or change its value to `Library`. The IL DLL for a library doesn't contain entry points and can't be executed.

MSBuild

`dotnet build` uses MSBuild to build the project, so it supports both parallel and incremental builds. For more information, see [Incremental Builds](#).

In addition to its options, the `dotnet build` command accepts MSBuild options, such as `-p` for setting properties or `-l` to define a logger. For more information about these options, see the [MSBuild Command-Line Reference](#). Or you can also use the `dotnet msbuild` command.

NOTE

When `dotnet build` is run automatically by `dotnet run`, arguments like `-property:property=value` aren't respected.

Running `dotnet build` is equivalent to running `dotnet msbuild -restore`; however, the default verbosity of the output is different.

Workload manifest downloads

When you run this command, it initiates an asynchronous background download of advertising manifests for workloads. If the download is still running when this command finishes, the download is stopped. For more information, see [Advertising manifests](#).

Arguments

PROJECT | SOLUTION

The project or solution file to build. If a project or solution file isn't specified, MSBuild searches the current working directory for a file that has a file extension that ends in either `proj` or `sln` and uses that file.

Options

- `-a|--arch <ARCHITECTURE>`

Specifies the target architecture. This is a shorthand syntax for setting the [Runtime Identifier \(RID\)](#), where

the provided value is combined with the default RID. For example, on a `win-x64` machine, specifying `--arch x86` sets the RID to `win-x86`. If you use this option, don't use the `-r|--runtime` option. Available since .NET 6 Preview 7.

- `-c|--configuration <CONFIGURATION>`

Defines the build configuration. The default for most projects is `Debug`, but you can override the build configuration settings in your project.

- `-f|--framework <FRAMEWORK>`

Compiles for a specific [framework](#). The framework must be defined in the [project file](#).

- `--force`

Forces all dependencies to be resolved even if the last restore was successful. Specifying this flag is the same as deleting the `project.assets.json` file.

- `-?|-h|--help`

Prints out a description of how to use the command.

- `--interactive`

Allows the command to stop and wait for user input or action. For example, to complete authentication. Available since .NET Core 3.0 SDK.

- `--no-dependencies`

Ignores project-to-project (P2P) references and only builds the specified root project.

- `--no-incremental`

Marks the build as unsafe for incremental build. This flag turns off incremental compilation and forces a clean rebuild of the project's dependency graph.

- `--no-restore`

Doesn't execute an implicit restore during build.

- `--nologo`

Doesn't display the startup banner or the copyright message.

- `--no-self-contained`

Publishes the application as a framework dependent application. A compatible .NET runtime must be installed on the target machine to run the application. Available since .NET 6 SDK.

- `-o|--output <OUTPUT_DIRECTORY>`

Directory in which to place the built binaries. If not specified, the default path is `./bin/<configuration>/<framework>/`. For projects with multiple target frameworks (via the `TargetFrameworks` property), you also need to define `--framework` when you specify this option.

- `--os <OS>`

Specifies the target operating system (OS). This is a shorthand syntax for setting the [Runtime Identifier \(RID\)](#), where the provided value is combined with the default RID. For example, on a `win-x64` machine, specifying `--os linux` sets the RID to `linux-x64`. If you use this option, don't use the `-r|--runtime` option. Available since .NET 6.

- `-r|--runtime <RUNTIME_IDENTIFIER>`

Specifies the target runtime. For a list of Runtime Identifiers (RIDs), see the [RID catalog](#). If you use this option with .NET 6 SDK, use `--self-contained` or `--no-self-contained` also. If not specified, the default is to build for the current OS and architecture.

- `--self-contained [true|false]`

Publishes the .NET runtime with the application so the runtime doesn't need to be installed on the target machine. The default is `true` if a runtime identifier is specified. Available since .NET 6 SDK.

- `--source <SOURCE>`

The URI of the NuGet package source to use during the restore operation.

- `-v|--verbosity <LEVEL>`

Sets the verbosity level of the command. Allowed values are `q[uiet]`, `m[inimal]`, `n[ormal]`, `d[etailed]`, and `diag[nostic]`. The default is `minimal`. For more information, see [LoggerVerbosity](#).

- `--version-suffix <VERSION_SUFFIX>`

Sets the value of the `$(VersionSuffix)` property to use when building the project. This only works if the `$(Version)` property isn't set. Then, `$(Version)` is set to the `$(VersionPrefix)` combined with the `$(VersionSuffix)`, separated by a dash.

Examples

- Build a project and its dependencies:

```
dotnet build
```

- Build a project and its dependencies using Release configuration:

```
dotnet build --configuration Release
```

- Build a project and its dependencies for a specific runtime (in this example, Ubuntu 18.04):

```
dotnet build --runtime ubuntu.18.04-x64
```

- Build the project and use the specified NuGet package source during the restore operation:

```
dotnet build --source c:\packages\mypackages
```

- Build the project and set version 1.2.3.4 as a build parameter using the `-p` [MSBuild option](#):

```
dotnet build -p:Version=1.2.3.4
```

dotnet build-server

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 3.1 SDK and later versions

Name

`dotnet build-server` - Interacts with servers started by a build.

Synopsis

```
dotnet build-server shutdown [--msbuild] [--razor] [--vbcscompiler]  
dotnet build-server shutdown -h|--help  
dotnet build-server -h|--help
```

Commands

- `shutdown`

Shuts down build servers that are started from dotnet. By default, all servers are shut down.

Options

- `-?|-h|--help`

Prints out a description of how to use the command.

- `--msbuild`

Shuts down the MSBuild build server.

- `--razor`

Shuts down the Razor build server.

- `--vbcscompiler`

Shuts down the VB/C# compiler build server.

dotnet clean

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 3.1 SDK and later versions

Name

`dotnet clean` - Cleans the output of a project.

Synopsis

```
dotnet clean [<PROJECT>|<SOLUTION>] [-c|--configuration <CONFIGURATION>]
    [-f|--framework <FRAMEWORK>] [--interactive]
    [--nologo] [-o|--output <OUTPUT_DIRECTORY>]
    [-r|--runtime <RUNTIME_IDENTIFIER>] [-v|--verbosity <LEVEL>]

dotnet clean -h|--help
```

Description

The `dotnet clean` command cleans the output of the previous build. It's implemented as an [MSBuild target](#), so the project is evaluated when the command is run. Only the outputs created during the build are cleaned. Both intermediate (*obj*) and final output (*bin*) folders are cleaned.

Arguments

`PROJECT | SOLUTION`

The MSBuild project or solution to clean. If a project or solution file is not specified, MSBuild searches the current working directory for a file that has a file extension that ends in *proj* or *sln*, and uses that file.

Options

- `-c|--configuration <CONFIGURATION>`

Defines the build configuration. The default for most projects is `Debug`, but you can override the build configuration settings in your project. This option is only required when cleaning if you specified it during build time.

- `-f|--framework <FRAMEWORK>`

The [framework](#) that was specified at build time. The framework must be defined in the [project file](#). If you specified the framework at build time, you must specify the framework when cleaning.

- `-?|-h|--help`

Prints out a description of how to use the command.

- `--interactive`

Allows the command to stop and wait for user input or action. For example, to complete authentication. Available since .NET Core 3.0 SDK.

- `--nologo`
Doesn't display the startup banner or the copyright message.
- `-o|--output <OUTPUT_DIRECTORY>`
The directory that contains the build artifacts to clean. Specify the `-f|--framework <FRAMEWORK>` switch with the output directory switch if you specified the framework when the project was built.
- `-r|--runtime <RUNTIME_IDENTIFIER>`
Cleans the output folder of the specified runtime. This is used when a [self-contained deployment](#) was created.
- `-v|--verbosity <LEVEL>`
Sets the verbosity level of the command. Allowed values are `q[uiet]`, `m[inimal]`, `n[ormal]`, `d[etailed]`, and `diag[nostic]`. The default is `normal`. For more information, see [LoggerVerbosity](#).

Examples

- Clean a default build of the project:

```
dotnet clean
```

- Clean a project built using the Release configuration:

```
dotnet clean --configuration Release
```

dotnet dev-certs

9/20/2022 • 5 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 3.1 SDK and later versions

Name

`dotnet dev-certs` - Generates a self-signed certificate to enable HTTPS use in development.

Synopsis

```
dotnet dev-certs https  
[-c|--check] [--clean] [-ep|--export-path <PATH>]  
[--format] [-i|--import] [-np|--no-password]  
[-p|--password] [-q|--quiet] [-t|--trust]  
[-v|--verbose] [--version]  
  
dotnet dev-certs https -h|--help
```

Description

The `dotnet dev-certs` command manages a self-signed certificate to enable HTTPS use in local web app development. Its main functions are:

- Generating a certificate for use with HTTPS endpoints during development.
- Trusting the generated certificate on the local machine.
- Removing the generated certificate from the local machine.
- Exporting a certificate in various formats so that it can be used by other tools.
- Importing an existing certificate generated by the tool into the local machine.

Commands

- `https`

`dotnet dev-certs` has only one command: `https`. The `dotnet dev-certs https` command with no options checks if a development certificate is present in the current user's certificate store on the machine. If the command finds a development certificate, it displays a message like the following example:

```
A valid HTTPS certificate is already present.
```

If the command doesn't find a development certificate, it creates one in the current user's certificate store, the store named `My` in the location `CurrentUser`. The physical location of the certificate is an implementation detail of the .NET runtime that could change at any time. On macOS in .NET 7.0, the certificate is stored in the user key chain and as a PFX file: `~/.aspnet/https-aspnetcore-localhost-<Thumbprint[0..5]>.pfx`.

After creating a certificate, the command displays a message like the following example:

```
The HTTPS developer certificate was generated successfully.
```

By default, the newly created certificate is not trusted. To trust the certificate, use the `--trust` option.

To create a file that you can use with other tools, use the `--export-path` option.

Options

- `-c|--check`

Checks for the existence of the development certificate but doesn't perform any action. Use this option with the `--trust` option to check if the certificate is not only valid but also trusted.

- `--clean`

Removes all HTTPS development certificates from the certificate store by using the .NET certificate store API. Doesn't remove any physical files that were created by using the `--export-path` option. On macOS in .NET 7.0, the `dotnet dev-certs` command creates the certificate on a path on disk, and the clean operation removes that certificate file.

If there's at least one certificate in the certificate store, the command displays a message like the following example:

```
Cleaning HTTPS development certificates
from the machine.
A prompt might get displayed to confirm
the removal of some of the certificates.

HTTPS development certificates
successfully removed from the machine.
```

- `-ep|--export-path <PATH>`

Exports the certificate to a file so that it can be used by other tools. Specify the full path to the exported certificate file, including the file name. The type of certificate files that are created depends on which options are used with `--export-path`:

OPTIONS	WHAT IS EXPORTED
<code>--export-path</code>	The public part of the certificate as a PFX file.
<code>--export-path --format PEM</code>	The public part of the certificate in PEM format. No separate <code>.key</code> file is created.
<code>--export-path --password</code>	The public and private parts of the certificate as a PFX file.
<code>--export-path --password --format PEM</code>	The public and private parts of the certificate as a pair of files in PEM format. The key file has the <code>.key</code> extension and is protected by the given password.
<code>--export-path --no-password --format PEM</code>	The public and private parts of the certificate as a pair of files in PEM format. The key file has the <code>.key</code> extension and is exported in plain text. The <code>--no-password</code> option is intended for internal testing use only.

- `--format`

When used with `--export-path`, specifies the format of the exported certificate file. Valid values are `PFX` and `PEM`, case-insensitive. `PFX` is the default.

The file format is independent of the file name extension. For example, if you specify `--format pfx` and `--export-path ./cert.pem`, you'll get a file named `cert.pem` in `PFX` format.

For information about the effect of this option when used with `--password`, `--no-password`, or without either of those options, see [--export-path](#) earlier in this article.

- `-i|--import <PATH>`

Imports the provided HTTPS development certificate into the local machine. Requires that you also specify the `--clean` option, which clears out any existing HTTPS developer certificates.

`PATH` specifies a path to a PFX certificate file. Provide the password with the `--password` option.

- `-np| --no-password`

Doesn't use a password for the key when exporting a certificate to PEM format files. The key file is exported in plain text. This option is not applicable to PFX files and is intended for internal testing use only.

- `-p| --password`

Specifies the password to use:

- When exporting the development certificate to a PFX or PEM file.
- When importing a PFX file.

When exporting with `--format PEM`, the public and private parts of the certificate are exported as a pair of files in PEM format. The key file has the `.key` extension and is protected by the given password. In addition to the file name specified for the `--export-path` option, the command creates another file in the same directory with the same name but a `.key` extension. For example, the following command will generate a file named `/localhost.pem` and a file named `/localhost.key` in the `/home/user` directory:

```
dotnet dev-certs https --format pem -ep /home/user/localhost.pem -p $CREDENTIAL_PLACEHOLDER$
```

In the example, `$CREDENTIAL_PLACEHOLDER$` represents a password.

- `-q| --quiet`

Display warnings and errors only.

- `-t| --trust`

Trusters the certificate on the local machine.

If this option isn't specified, the certificate is added to the certificate store but not to a trusted list.

When combined with the `--check` option, validates that the certificate is trusted.

- `-v| --verbose`

Display debug information.

Examples

- Check for the presence of a development certificate, and create one in the default certificate store if one doesn't exist yet. But don't trust the certificate.

```
dotnet dev-certs https
```

- Remove any development certificates that already exist on the local machine.

```
dotnet dev-certs https --clean
```

- Import a PFX file.

```
dotnet dev-certs https --clean --import ./certificate.pfx -p $CREDENTIAL_PLACEHOLDER$
```

In the preceding example, `$CREDENTIAL_PLACEHOLDER$` represents a password.

- Check if a trusted development certificate is present on the local machine.

```
dotnet dev-certs https --check --trust
```

- Create a certificate, trust it, and export it to a PFX file.

```
dotnet dev-certs https -ep ./certificate.pfx -p $CREDENTIAL_PLACEHOLDER$ --trust
```

- Create a certificate, trust it, and export it to a PEM file.

```
dotnet dev-certs https -ep ./certificate.crt --trust --format PEM
```

- Create a certificate, trust it, and export it to a PEM file including the private key:

```
dotnet dev-certs https -ep ./certificate.crt -p $CREDENTIAL_PLACEHOLDER$ --trust --format PEM
```

See also

- [Generate self-signed certificates with the .NET CLI](#)
- [Enforce HTTPS in ASP.NET Core](#)
- [Troubleshoot certificate problems such as certificate not trusted](#)
- [Hosting ASP.NET Core images with Docker over HTTPS](#)
- [Hosting ASP.NET Core images with Docker Compose over HTTPS](#)

dotnet format

9/20/2022 • 3 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET 6.x SDK and later versions

Name

`dotnet format` - Formats code to match `editorconfig` settings.

Synopsis

```
dotnet format [options] [<PROJECT | SOLUTION>]
```

```
dotnet format -h|--help
```

Description

`dotnet format` is a code formatter that applies style preferences to a project or solution. Preferences will be read from an `.editorconfig` file, if present, otherwise a default set of preferences will be used. For more information, see the [EditorConfig documentation](#).

Arguments

`PROJECT | SOLUTION`

The MSBuild project or solution to run code formatting on. If a project or solution file is not specified, MSBuild searches the current working directory for a file that has a file extension that ends in `proj` or `sln`, and uses that file.

Options

None of the options below are required for the `dotnet format` command to succeed, but you can use them to further customize what is formatted and by which rules.

- `--diagnostics <DIAGNOSTICS>`

A space-separated list of diagnostic IDs to use as a filter when fixing code style or third-party issues.

Default value is whichever IDs are listed in the `.editorconfig` file. For a list of built-in analyzer rule IDs that you can specify, see the [list of IDs for code-analysis style rules](#).

- `--severity`

The minimum severity of diagnostics to fix. Allowed values are `info`, `warn`, and `error`. The default value is `warn`.

- `--no-restore`

Doesn't execute an implicit restore before formatting. Default is to do implicit restore.

- `--verify-no-changes`

Verifies that no formatting changes would be performed. Terminates with a non zero exit code if any files

would have been formatted.

- `--include <INCLUDE>`

A space-separated list of relative file or folder paths to include in formatting. The default is all files in the solution or project.

- `--exclude <EXCLUDE>`

A space-separated list of relative file or folder paths to exclude from formatting. The default is none.

- `--include-generated`

Formats files generated by the SDK.

- `-v|--verbosity <LEVEL>`

Sets the verbosity level. Allowed values are `q[uiet]`, `m[inimal]`, `n[ormal]`, `d[etailed]`, and `diag[nostic]`. Default value is `m[inimal]`.

- `--binarylog <BINARY-LOG-PATH>`

Logs all project or solution load information to a binary log file.

- `--report <REPORT-PATH>`

Produces a JSON report in the directory specified by `<REPORT_PATH>`.

- `-h|--help`

Shows help and usage information

Subcommands

Whitespace

`dotnet format whitespace` - Formats code to match `editorconfig` settings for whitespace.

Description

The `dotnet format whitespace` subcommand will only run formatting rules associated with whitespace formatting. For a complete list of possible formatting options that you can specify in your `.editorconfig` file, see the [C# formatting options](#).

Options

- `--folder`

Treat the `<PROJECT | SOLUTION>` argument as a path to a simple folder of code files.

Style

`dotnet format style` - Formats code to match EditorConfig settings for code style.

Description

The `dotnet format style` subcommand will only run formatting rule associated with code style formatting. For a complete list of formatting options that you can specify in your `editorconfig` file, see [Code style rules](#).

Options

- `--diagnostics <DIAGNOSTICS>`

A space-separated list of diagnostic IDs to use as a filter when fixing code style or third-party issues.

Default value is whichever IDs are listed in the `.editorconfig` file. For a list of built-in analyzer rule IDs that you can specify, see the [list of IDs for code-analysis style rules](#).

- `--severity`

The minimum severity of diagnostics to fix. Allowed values are `info`, `warn`, and `error`. The default value is `warn`.

Analyzers

```
dotnet format analyzers
```

 - Formats code to match `editorconfig` settings for analyzers.

Description

The `dotnet format analyzers` subcommand will only run formatting rule associated with analyzers. For a list of analyzer rules that you can specify in your `editorconfig` file, see [Code style rules](#).

Options

- `--diagnostics <DIAGNOSTICS>`

A space-separated list of diagnostic IDs to use as a filter when fixing code style or third-party issues.

Default value is whichever IDs are listed in the `.editorconfig` file. For a list of built-in analyzer rule IDs that you can specify, see the [list of IDs for code-analysis style rules](#).

- `--severity`

The minimum severity of diagnostics to fix. Allowed values are `info`, `warn`, and `error`. The default value is `warn`.

Examples

- Format all code in the solution:

```
dotnet format ./solution.sln
```

- Clean up all code in the application project:

```
dotnet format ./src/application.csproj
```

- Verify that all code is correctly formatted:

```
dotnet format --verify-no-changes
```

- Clean up all code in the `src` and `tests` directory but not in `src/submodule-a`:

```
dotnet format --include ./src/ ./tests/ --exclude ./src/submodule-a/
```

dotnet help reference

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 3.1 SDK and later versions

Name

`dotnet help` - Shows more detailed documentation online for the specified command.

Synopsis

```
dotnet help <COMMAND_NAME> [-h|--help]
```

Description

The `dotnet help` command opens up the reference page for more detailed information about the specified command.

Arguments

- `COMMAND_NAME`

Name of the .NET CLI command. For a list of the valid CLI commands, see [CLI commands](#).

Options

- `-?|-h|--help`

Prints out a description of how to use the command.

Examples

- Opens the documentation page for the `dotnet new` command:

```
dotnet help new
```

dotnet migrate

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 2.x SDK

Name

`dotnet migrate` - Migrates a Preview 2 .NET Core project to a .NET Core SDK-style project.

Synopsis

```
dotnet migrate [<SOLUTION_FILE|PROJECT_DIR>] [--format-report-file-json <REPORT_FILE>]
  [-r|--report-file <REPORT_FILE>] [-s|--skip-project-references [Debug|Release]]
  [--skip-backup] [-t|--template-file <TEMPLATE_FILE>] [-v|--sdk-package-version]
  [-x|--xproj-file]

dotnet migrate -h|--help
```

Description

This command is deprecated. The `dotnet migrate` command is no longer available starting with .NET Core 3.0 SDK. It can only migrate a Preview 2 .NET Core project to a 1.x .NET Core project, which is out of support.

By default, the command migrates the root project and any project references that the root project contains. This behavior is disabled using the `--skip-project-references` option at run time.

Migration can be performed on the following assets:

- A single project by specifying the *project.json* file to migrate.
- All of the directories specified in the *global.json* file by passing in a path to the *global.json* file.
- A *solution.sln* file, where it migrates the projects referenced in the solution.
- On all subdirectories of the given directory recursively.

The `dotnet migrate` command keeps the migrated *project.json* file inside a `backup` directory, which it creates if the directory doesn't exist. This behavior is overridden using the `--skip-backup` option.

By default, the migration operation outputs the state of the migration process to standard output (STDOUT). If you use the `--report-file <REPORT_FILE>` option, the output is saved to the file specify.

The `dotnet migrate` command only supports valid Preview 2 *project.json*-based projects. This means that you cannot use it to migrate DNX or Preview 1 *project.json*-based projects directly to MSBuild/csproj projects. You first need to manually migrate the project to a Preview 2 *project.json*-based project and then use the `dotnet migrate` command to migrate the project.

Arguments

`PROJECT_JSON/GLOBAL_JSON/SOLUTION_FILE/PROJECT_DIR`

The path to one of the following:

- a *project.json* file to migrate.
- a *global.json* file: the folders specified in *global.json* are migrated.

- a *solution.sln* file: the projects referenced in the solution are migrated.
- a directory to migrate: recursively searches for *project.json* files to migrate inside the specified directory.

Defaults to current directory if nothing is specified.

Options

```
--format-report-file-json <REPORT_FILE>
```

Output migration report file as JSON rather than user messages.

```
-h|--help
```

Prints out a short help for the command.

```
-r|--report-file <REPORT_FILE>
```

Output migration report to a file in addition to the console.

```
-s|--skip-project-references [Debug|Release]
```

Skip migrating project references. By default, project references are migrated recursively.

```
--skip-backup
```

Skip moving *project.json*, *global.json*, and **.xproj* to a `backup` directory after successful migration.

```
-t|--template-file <TEMPLATE_FILE>
```

Template csproj file to use for migration. By default, the same template as the one dropped by `dotnet new console` is used.

```
-v|--sdk-package-version <VERSION>
```

The version of the sdk package that's referenced in the migrated app. The default is the version of the SDK in `dotnet new .`

```
-x|--xproj-file <FILE>
```

The path to the xproj file to use. Required when there is more than one xproj in a project directory.

Examples

Migrate a project in the current directory and all of its project-to-project dependencies:

```
dotnet migrate
```

Migrate all projects that *global.json* file includes:

```
dotnet migrate path/to/global.json
```

Migrate only the current project and no project-to-project (P2P) dependencies. Also, use a specific SDK version:

```
dotnet migrate -s -v 1.0.0-preview4
```

dotnet msbuild

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 3.1 SDK and later versions

Name

`dotnet msbuild` - Builds a project and all of its dependencies. Note: A solution or project file may need to be specified if there are multiple.

Synopsis

```
dotnet msbuild <MSBUILD_ARGUMENTS>  
dotnet msbuild -h
```

Description

The `dotnet msbuild` command allows access to a fully functional MSBuild.

The command has the exact same capabilities as the existing MSBuild command-line client for SDK-style projects only. The options are all the same. For more information about the available options, see the [MSBuild command-line reference](#).

The `dotnet build` command is equivalent to `dotnet msbuild -restore`. When you don't want to build the project and you have a specific target you want to run, use `dotnet build` or `dotnet msbuild` and specify the target.

Examples

- Build a project and its dependencies:

```
dotnet msbuild
```

- Build a project and its dependencies using Release configuration:

```
dotnet msbuild -property:Configuration=Release
```

- Run the publish target and publish for the `osx.10.11-x64` RID:

```
dotnet msbuild -target:Publish -property:RuntimeIdentifiers=osx.10.11-x64
```

- See the whole project with all targets included by the SDK:

```
dotnet msbuild -preprocess  
dotnet msbuild -preprocess:<fileName>.xml
```

dotnet new <TEMPLATE>

9/20/2022 • 5 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 3.1 SDK and later versions

Name

`dotnet new` - Creates a new project, configuration file, or solution based on the specified template.

Synopsis

```
dotnet new <TEMPLATE> [--dry-run] [--force] [-lang|--language {"C#"|"F#"|VB"}]
[-n|--name <OUTPUT_NAME>] [--no-update-check] [-o|--output <OUTPUT_DIRECTORY>] [Template options]

dotnet new -h|--help
```

Description

The `dotnet new` command creates a .NET project or other artifacts based on a template.

The command calls the [template engine](#) to create the artifacts on disk based on the specified template and options.

NOTE

Starting with .NET SDK 7.0.100 Preview 2, the `dotnet new` syntax has changed:

- The `--list`, `--search`, `--install`, and `--uninstall` options became `list`, `search`, `install` and `uninstall` subcommands.
- The `--update-apply` option became the `update` subcommand.
- to use `--update-check`, use the `update` subcommand with the `--check` option.

Other options that were available before are still available to use with their respective subcommands. Separate help for each subcommand is available via the `-h` or `--help` option: `dotnet new <subcommand> --help` lists all supported options for the subcommand.

Additionally, tab completion is now available for `dotnet new`. It supports completion for installed template names, as well as completion for the options a selected template provides. To activate tab completion for the .NET SDK, see [Enable tab completion](#).

Implicit restore

You don't have to run `dotnet restore` because it's run implicitly by all commands that require a restore to occur, such as `dotnet new`, `dotnet build`, `dotnet run`, `dotnet test`, `dotnet publish`, and `dotnet pack`. To disable implicit restore, use the `--no-restore` option.

The `dotnet restore` command is still useful in certain scenarios where explicitly restoring makes sense, such as [continuous integration builds in Azure DevOps Services](#) or in build systems that need to explicitly control when the restore occurs.

For information about how to manage NuGet feeds, see the [dotnet restore](#) documentation.

Arguments

- **TEMPLATE**

The template to instantiate when the command is invoked. Each template might have specific options you can pass. For more information, see [Template options](#).

You can run `dotnet new --list` to see a list of all installed templates.

Starting with .NET Core 3.0 SDK and ending with .NET Core 5.0.300 SDK, the CLI searches for templates in NuGet.org when you invoke the `dotnet new` command in the following conditions:

- If the CLI can't find a template match when invoking `dotnet new`, not even partial.
- If there's a newer version of the template available. In this case, the project or artifact is created but the CLI warns you about an updated version of the template.

Starting with .NET Core 5.0.300 SDK, the `--search` option should be used to search for templates in NuGet.org..

The following table shows the templates that come pre-installed with the .NET SDK. The default language for the template is shown inside the brackets. Click on the short name link to see the specific template options.

TEMPLATES	SHORT NAME	LANGUAGE	TAGS	INTRODUCED
Console Application	console	[C#], F#, VB	Common/Console	1.0
Class library	classlib	[C#], F#, VB	Common/Library	1.0
WPF Application	wpf	[C#], VB	Common/WPF	3.0 (5.0 for VB)
WPF Class library	wpfclib	[C#], VB	Common/WPF	3.0 (5.0 for VB)
WPF Custom Control Library	wpfcustomcontrollib	[C#], VB	Common/WPF	3.0 (5.0 for VB)
WPF User Control Library	wpfusercontrollib	[C#], VB	Common/WPF	3.0 (5.0 for VB)
Windows Forms (WinForms) Application	winforms	[C#], VB	Common/WinForms	3.0 (5.0 for VB)
Windows Forms (WinForms) Class library	winformslib	[C#], VB	Common/WinForms	3.0 (5.0 for VB)
Worker Service	worker	[C#]	Common/Worker/Web	3.0
Unit Test Project	mstest	[C#], F#, VB	Test/MSTest	1.0
NUnit 3 Test Project	nunit	[C#], F#, VB	Test/NUnit	2.1.400
NUnit 3 Test Item	nunit-test	[C#], F#, VB	Test/NUnit	2.2

TEMPLATES	SHORT NAME	LANGUAGE	TAGS	INTRODUCED
xUnit Test Project	xunit	[C#], F#, VB	Test/xUnit	1.0
Razor Component	razorcomponent	[C#]	Web/ASP.NET	3.0
Razor Page	page	[C#]	Web/ASP.NET	2.0
MVC ViewImports	viewimports	[C#]	Web/ASP.NET	2.0
MVC ViewStart	viewstart	[C#]	Web/ASP.NET	2.0
Blazor Server App	blazorserver	[C#]	Web/Blazor	3.0
Blazor WebAssembly App	blazorwasm	[C#]	Web/Blazor/WebAssembly	3.1.300
ASP.NET Core Empty	web	[C#], F#	Web/Empty	1.0
ASP.NET Core Web App (Model-View-Controller)	mvc	[C#], F#	Web/MVC	1.0
ASP.NET Core Web App	webapp, razor	[C#]	Web/MVC/Razor Pages	2.2, 2.0
ASP.NET Core with Angular	angular	[C#]	Web/MVC/SPA	2.0
ASP.NET Core with React.js	react	[C#]	Web/MVC/SPA	2.0
ASP.NET Core with React.js and Redux	reactredux	[C#]	Web/MVC/SPA	2.0
Razor Class Library	razorclasslib	[C#]	Web/Razor/Library/Razor Class Library	2.1
ASP.NET Core Web API	webapi	[C#], F#	Web/WebAPI	1.0
ASP.NET Core gRPC Service	grpc	[C#]	Web/gRPC	3.0
dotnet gitignore file	gitignore		Config	3.0
global.json file	globaljson		Config	2.0
NuGet Config	nugetconfig		Config	1.0
Dotnet local tool manifest file	tool-manifest		Config	3.0

TEMPLATES	SHORT NAME	LANGUAGE	TAGS	INTRODUCED
Web Config	webconfig		Config	1.0
Solution File	sln		Solution	1.0
Protocol Buffer File	proto		Web/gRPC	3.0
EditorConfig file	editorconfig		Config	6.0

Options

- `--dry-run`

Displays a summary of what would happen if the given command were run if it would result in a template creation. Available since .NET Core 2.2 SDK.

- `--force`

Forces content to be generated even if it would change existing files. This is required when the template chosen would override existing files in the output directory.

- `-?|-h|--help`

Prints out help for the command. It can be invoked for the `dotnet new` command itself or for any template. For example, `dotnet new mvc --help`.

- `-lang|--language {C#|F#|VB}`

The language of the template to create. The language accepted varies by the template (see defaults in the [arguments](#) section). Not valid for some templates.

NOTE

Some shells interpret `#` as a special character. In those cases, enclose the language parameter value in quotes. For example, `dotnet new console -lang "F#"`.

- `-n|--name <OUTPUT_NAME>`

The name for the created output. If no name is specified, the name of the current directory is used.

- `-no-update-check`

Disables checking for template package updates when instantiating a template. Available since .NET 6.0.100 SDK. When instantiating the template from a template package that was installed by using `dotnet new --install`, `dotnet new` checks if there is an update for the template. Starting with .NET 6, no update checks are done for .NET default templates. To update .NET default templates, install the patch version of the .NET SDK.

- `-o|--output <OUTPUT_DIRECTORY>`

Location to place the generated output. The default is the current directory.

Template options

Each template may have additional options defined. For more information, see [.NET default templates for](#)

```
dotnet new .
```

Examples

- Create a C# console application project:

```
dotnet new console
```

- Create an F# console application project in the current directory:

```
dotnet new console --language "F#"
```

- Create a .NET Standard 2.0 class library project in the specified directory:

```
dotnet new classlib --framework "netstandard2.0" -o MyLibrary
```

- Create a new ASP.NET Core C# MVC project in the current directory with no authentication:

```
dotnet new mvc -au None
```

- Create a new xUnit project:

```
dotnet new xunit
```

- Create a *global.json* in the current directory setting the SDK version to 3.1.101:

```
dotnet new globaljson --sdk-version 3.1.101
```

- Show help for the C# console application template:

```
dotnet new console -h
```

- Show help for the F# console application template:

```
dotnet new console --language "F#" -h
```

See also

- [dotnet new --list option](#)
- [dotnet new --search option](#)
- [dotnet new --install option](#)
- [.NET default templates for dotnet new](#)
- [Custom templates for dotnet new](#)
- [Create a custom template for dotnet new](#)

dotnet new list

9/20/2022 • 3 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 3.1 SDK and later versions

Name

`dotnet new --list` - Lists available templates to be run using `dotnet new`.

Synopsis

```
dotnet new [<TEMPLATE_NAME>] -l|--list [--author <AUTHOR>] [-lang|--language {"C#"|"F#"|VB"}]
[--tag <TAG>] [--type <TYPE>] [--columns <COLUMNS>] [--columns-all]
```

Description

The `dotnet new --list` option lists available templates to use with `dotnet new`. If the `<TEMPLATE_NAME>` is specified, lists templates containing the specified name. This option lists only default and installed templates. To find templates in NuGet that you can install locally, use the `--search` option.

NOTE

Starting with .NET SDK 7.0.100 Preview 2, the `dotnet new` syntax has changed:

- The `--list`, `--search`, `--install`, and `--uninstall` options became `list`, `search`, `install` and `uninstall` subcommands.
- The `--update-apply` option became the `update` subcommand.
- to use `--update-check`, use the `update` subcommand with the `--check` option.

Other options that were available before are still available to use with their respective subcommands. Separate help for each subcommand is available via the `-h` or `--help` option: `dotnet new <subcommand> --help` lists all supported options for the subcommand.

Additionally, tab completion is now available for `dotnet new`. It supports completion for installed template names, as well as completion for the options a selected template provides. To activate tab completion for the .NET SDK, see [Enable tab completion](#).

Examples of the new syntax:

- Show help for `list` subcommand

```
dotnet new list --help
```

- List all templates matching the `we` substring that support the F# language.

```
dotnet new list we --language "F#"
```

Arguments

- `TEMPLATE_NAME`

If the argument is specified, only the templates containing `<TEMPLATE_NAME>` in template name or short name will be shown.

NOTE

Starting with .NET SDK 6.0.100, you can put the `<TEMPLATE_NAME>` argument after the `--list` option. For example, `dotnet new --list web` provides the same result as `dotnet new web --list`. Using more than one argument is not allowed.

Options

- `--author <AUTHOR>`

Filters templates based on template author. Partial match is supported. Available since .NET Core 5.0.300 SDK.

- `--columns <COLUMNS>`

Comma-separated list of columns to display in the output. The supported columns are:

- `language` - A comma-separated list of languages supported by the template.
- `tags` - The list of template tags.
- `author` - The template author.
- `type` - The template type: project or item.

The template name and short name are always shown. The default list of columns is template name, short name, language, and tags. This list is equivalent to specifying `--columns=language,tags`. Available since .NET Core 5.0.300 SDK.

- `--columns-all`

Displays all columns in the output. Available since .NET Core 5.0.300 SDK.

- `-lang|--language {C#|F#|VB}`

Filters templates based on language supported by the template. The language accepted varies by the template. Not valid for some templates.

NOTE

Some shells interpret `#` as a special character. In those cases, enclose the language parameter value in quotes.

For example, `dotnet new --list --language "F#"`.

- `--tag <TAG>`

Filters templates based on template tags. To be selected, a template must have at least one tag that exactly matches the criteria. Available since .NET Core 5.0.300 SDK.

- `--type <TYPE>`

Filters templates based on template type. Predefined values are `project`, `item`, and `solution`.

Examples

- List all templates

```
dotnet new --list
```

- List all Single Page Application (SPA) templates:

- since .NET SDK 6.0.100

```
dotnet new --list spa
```

- before .NET SDK 6.0.100

```
dotnet new spa --list
```

- List all templates matching the *we* substring.

- since .NET SDK 6.0.100

```
dotnet new --list we
```

- before .NET SDK 6.0.100

```
dotnet new we --list
```

- List all templates matching the *we* substring that support the F# language.

```
dotnet new --list we --language "F#"
```

- List all item templates.

```
dotnet new --list --type item
```

- List all C# templates, showing the author and the type in the output.

```
dotnet new --list --language "C#" --columns "author,type"
```

See also

- [dotnet new command](#)
- [dotnet new --search option](#)
- [Custom templates for dotnet new](#)

dotnet new search

9/20/2022 • 3 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 5.0.300 SDK and later versions

Name

`dotnet new --search` - searches for the templates supported by `dotnet new` on NuGet.org.

Synopsis

```
dotnet new <TEMPLATE_NAME> --search

dotnet new [<TEMPLATE_NAME>] --search [--author <AUTHOR>] [-lang|--language {"C#"|"F#"|VB}]
  [--package <PACKAGE>] [--tag <TAG>] [--type <TYPE>]
  [--columns <COLUMNS>] [--columns-all]
```

Description

The `dotnet new --search` option searches for templates supported by `dotnet new` on NuGet.org. When the `<TEMPLATE_NAME>` is specified, searches for templates containing the specified name.

NOTE

Starting with .NET SDK 7.0.100 Preview 2, the `dotnet new` syntax has changed:

- The `--list`, `--search`, `--install`, and `--uninstall` options became `list`, `search`, `install` and `uninstall` subcommands.
- The `--update-apply` option became the `update` subcommand.
- to use `--update-check`, use the `update` subcommand with the `--check` option.

Other options that were available before are still available to use with their respective subcommands. Separate help for each subcommand is available via the `-h` or `--help` option: `dotnet new <subcommand> --help` lists all supported options for the subcommand.

Additionally, tab completion is now available for `dotnet new`. It supports completion for installed template names, as well as completion for the options a selected template provides. To activate tab completion for the .NET SDK, see [Enable tab completion](#).

Examples of the new syntax:

- Show help for the `search` subcommand.

```
dotnet new search --help
```

- Search for all templates available on NuGet.org matching the "we" substring and supporting the F# language

```
dotnet new search we --language "F#"
```

Arguments

- `<TEMPLATE_NAME>`

If the argument is specified, only templates containing `<TEMPLATE_NAME>` in the template name or short name will be shown. The argument is mandatory when `--author`, `--language`, `--package`, `--tag` or `--type` options are not specified.

NOTE

Starting with .NET SDK 6.0.100, you can put the `<TEMPLATE_NAME>` argument after the `--search` option. For example, `dotnet new --search web` provides the same result as `dotnet new web --search`. Using more than one argument is not allowed.

Options

- `--author <AUTHOR>`

Filters templates based on template author. Partial match is supported.

- `--columns <COLUMNS>`

Comma-separated list of columns to display in the output. The supported columns are:

- `language` - A comma-separated list of languages supported by the template.
- `tags` - The list of template tags.
- `author` - The template author.
- `type` - The template type: project or item.

The template name, short name, package name and total downloads count are always shown. The default list of columns is template name, short name, author, language, package, and total downloads. This list is equivalent to specifying `--columns=author,language`.

- `--columns-all`

Displays all columns in the output.

- `-lang|--language {C#|F#|VB}`

Filters templates based on language supported by the template. The language accepted varies by the template. Not valid for some templates.

NOTE

Some shells interpret `#` as a special character. In those cases, enclose the language parameter value in quotes.

For example, `dotnet new --search --language "F#"`.

- `--package <PACKAGE>`

Filters templates based on NuGet package ID. Partial match is supported.

- `--tag <TAG>`

Filters templates based on template tags. To be selected, a template must have at least one tag that exactly matches the criteria.

- `--type <TYPE>`

Filters templates based on template type. Predefined values are `project`, `item`, and `solution`.

NOTE

To ensure that the template package appears in `dotnet new --search` result, set the NuGet package type to `Template`.

Examples

- Search for all templates available on NuGet.org matching the `spa` substring.

- since .NET SDK 6.0.100

```
dotnet new --search spa
```

- before .NET SDK 6.0.100

```
dotnet new spa --search
```

- Search for all templates available on NuGet.org matching the `we` substring and supporting the F# language.

- since .NET SDK 6.0.100

```
dotnet new --search we --language "F#"
```

- before .NET SDK 6.0.100

```
dotnet new we --search --language "F#"
```

- Search for item templates.

```
dotnet new --search --type item
```

- Search for all C# templates, showing the type and tags in the output.

```
dotnet new --search --language "C#" --columns "type,tags"
```

See also

- [dotnet new command](#)
- [dotnet new --list option](#)
- [Custom templates for dotnet new](#)

dotnet new install

9/20/2022 • 3 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 3.1 SDK and later versions

Name

`dotnet new --install` - installs a template package.

Synopsis

```
dotnet new --install <PATH|NUGET_ID> [--interactive] [--nuget-source <SOURCE>]
```

Description

The `dotnet new --install` command installs a template package from the `PATH` or `NUGET_ID` provided. If you want to install a specific version or prerelease version of a template package, specify the version in the format `<package-name>:<package-version>`. By default, `dotnet new` passes * for the version, which represents the latest stable package version. For more information, see the [Examples](#) section.

If a version of the template package was already installed when you run this command, the template package will be updated to the specified version. If no version is specified, the package is updated to the latest stable version. Starting with .NET SDK 6.0.100, if the argument specifies the version, and that version of the NuGet package is already installed, it won't be reinstalled. If the argument is a `PATH` and it's already installed, it won't be reinstalled.

Prior to .NET SDK 6.0.100, template packages were managed individually for each .NET SDK version, including [patch versions](#). For example, if you install the template package using `dotnet new --install` in .NET SDK 5.0.100, it will be installed only for .NET SDK 5.0.100. Templates from the package won't be available in other .NET SDK versions installed on your machine.

Starting with .NET SDK 6.0.100, installed template packages are available in later .NET SDK versions installed on your machine. A template package installed in .NET SDK 6.0.100 will also be available in .NET SDK 6.0.101, .NET SDK 6.0.200, and so on. However, these template packages won't be available in .NET SDK versions prior to .NET SDK 6.0.100. To use a template package installed in .NET SDK 6.0.100 or later in earlier .NET SDK versions, you need to install it using `dotnet new --install` in that .NET SDK version.

NOTE

Starting with .NET SDK 7.0.100 Preview 2, the `dotnet new` syntax has changed:

- The `--list`, `--search`, `--install`, and `--uninstall` options became `list`, `search`, `install` and `uninstall` subcommands.
- The `--update-apply` option became the `update` subcommand.
- to use `--update-check`, use the `update` subcommand with the `--check` option.

Other options that were available before are still available to use with their respective subcommands. Separate help for each subcommand is available via the `-h` or `--help` option: `dotnet new <subcommand> --help` lists all supported options for the subcommand.

Additionally, tab completion is now available for `dotnet new`. It supports completion for installed template names, as well as completion for the options a selected template provides. To activate tab completion for the .NET SDK, see [Enable tab completion](#).

Examples of new syntax:

- Show help for `install` subcommand

```
dotnet new install --help
```

- Install the latest version of Azure web jobs project template package:

```
dotnet new install Microsoft.Azure.WebJobs.ProjectTemplates
```

Options

- `--interactive`

Allows the command to stop and wait for user input or action. For example, to complete authentication.
Available since .NET 5.0 SDK.

- `--nuget-source <SOURCE>`

By default, `dotnet new --install` uses the hierarchy of NuGet configuration files from the current directory to determine the NuGet source the package can be installed from. If `--nuget-source` is specified, the source will be added to the list of sources to be checked.

To check the configured sources for the current directory use `dotnet nuget list source`. For more information, see [Common NuGet Configurations](#)

Examples

- Install the latest version of SPA templates for ASP.NET Core:

```
dotnet new --install Microsoft.DotNet.Web.Spa.ProjectTemplates
```

- Install version 2.0 of the SPA templates for ASP.NET Core:

```
dotnet new --install Microsoft.DotNet.Web.Spa.ProjectTemplates::2.0.0
```

- Install version 2.0 of the SPA templates for ASP.NET Core from a custom NuGet source using interactive mode:

```
dotnet new --install Microsoft.DotNet.Web.Spa.ProjectTemplates::2.0.0 --nuget-source "https://api.my-
custom-nuget.com/v3/index.json" --interactive
```

See also

- [dotnet new](#) command
- [dotnet new --search](#) option
- Custom templates for [dotnet new](#)

dotnet new uninstall

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 3.1 SDK and later versions

Name

`dotnet new --uninstall` - uninstalls a template package.

Synopsis

```
dotnet new --uninstall <PATH|NUGET_ID>
```

Description

The `dotnet new --uninstall` command uninstalls a template package at the `<PATH|NUGET_ID>` provided. When the `<PATH|NUGET_ID>` value isn't specified, all currently installed template packages and their associated templates are displayed. When specifying `NUGET_ID`, don't include the version number.

NOTE

Starting with .NET SDK 7.0.100 Preview 2, the `dotnet new` syntax has changed:

- The `--list`, `--search`, `--install`, and `--uninstall` options became `list`, `search`, `install` and `uninstall` subcommands.
- The `--update-apply` option became the `update` subcommand.
- to use `--update-check`, use the `update` subcommand with the `--check` option.

Other options that were available before are still available to use with their respective subcommands. Separate help for each subcommand is available via the `-h` or `--help` option: `dotnet new <subcommand> --help` lists all supported options for the subcommand.

Additionally, tab completion is now available for `dotnet new`. It supports completion for installed template names, as well as completion for the options a selected template provides. To activate tab completion for the .NET SDK, see [Enable tab completion](#).

Examples of the new syntax:

- Show help for the `uninstall` subcommand.

```
dotnet new uninstall --help
```

- List the installed templates and details about them, including how to uninstall them:

```
dotnet new uninstall
```

- Uninstall the Azure web jobs project template package:

```
dotnet new uninstall Microsoft.Azure.WebJobs.ProjectTemplates
```

Examples

- List the installed templates and details about them, including how to uninstall them:

```
dotnet new --uninstall
```

- Uninstall the SPA templates for ASP.NET Core:

```
dotnet new --uninstall Microsoft.DotNet.Web.Spa.ProjectTemplates
```

See also

- [dotnet new command](#)
- [dotnet new --list option](#)
- [dotnet new --search option](#)
- [Custom templates for dotnet new](#)

dotnet new update

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 3.1 SDK and later versions

Name

`dotnet new --update-check` - Checks for available updates for installed template packages.

`dotnet new --update-apply` - Applies updates to installed template packages.

Synopsis

```
dotnet new --update-check
```

```
dotnet new --update-apply
```

Description

The `dotnet new --update-check` option checks if there are updates available for the template packages that are currently installed. The `dotnet new --update-apply` option checks if there are updates available for the template packages that are currently installed and installs them.

NOTE

Starting with .NET SDK 7.0.100 Preview 2, the `dotnet new` syntax has changed:

- The `--list`, `--search`, `--install`, and `--uninstall` options became `list`, `search`, `install` and `uninstall` subcommands.
- The `--update-apply` option became the `update` subcommand.
- to use `--update-check`, use the `update` subcommand with the `--check` option.

Other options that were available before are still available to use with their respective subcommands. Separate help for each subcommand is available via the `-h` or `--help` option: `dotnet new <subcommand> --help` lists all supported options for the subcommand.

Additionally, tab completion is now available for `dotnet new`. It supports completion for installed template names, as well as completion for the options a selected template provides. To activate tab completion for the .NET SDK, see [Enable tab completion](#).

Examples of the new syntax:

- Show help for the `update` subcommand.

```
dotnet new update --help
```

- Check for updates for installed template packages:

```
dotnet new update --check
```

- Update installed template packages:

```
dotnet new update
```

See also

- [dotnet new command](#)
- [dotnet new --search option](#)
- [dotnet new --install option](#)
- [Custom templates for dotnet new](#)

.NET default templates for dotnet new

9/20/2022 • 17 minutes to read • [Edit Online](#)

When you install the [.NET SDK](#), you receive over a dozen built-in templates for creating projects and files, including console apps, class libraries, unit test projects, ASP.NET Core apps (including [Angular](#) and [React](#) projects), and configuration files. To list the built-in templates, run the `dotnet new` command with the `-l|--list` option:

```
dotnet new --list
```

The following table shows the templates that come pre-installed with the .NET SDK. The default language for the template is shown inside the brackets. Click on the short name link to see the specific template options.

TEMPLATES	SHORT NAME	LANGUAGE	TAGS	INTRODUCED
Console Application	console	[C#], F#, VB	Common/Console	1.0
Class library	classlib	[C#], F#, VB	Common/Library	1.0
WPF Application	wpf	[C#], VB	Common/WPF	3.0 (5.0 for VB)
WPF Class library	wpflib	[C#], VB	Common/WPF	3.0 (5.0 for VB)
WPF Custom Control Library	wpfcustomcontrollib	[C#], VB	Common/WPF	3.0 (5.0 for VB)
WPF User Control Library	wpfusercontrollib	[C#], VB	Common/WPF	3.0 (5.0 for VB)
Windows Forms (WinForms) Application	winforms	[C#], VB	Common/WinForms	3.0 (5.0 for VB)
Windows Forms (WinForms) Class library	winformslib	[C#], VB	Common/WinForms	3.0 (5.0 for VB)
Worker Service	worker	[C#]	Common/Worker/Web	3.0
Unit Test Project	mstest	[C#], F#, VB	Test/MSTest	1.0
NUnit 3 Test Project	nunit	[C#], F#, VB	Test/NUnit	2.1.400
NUnit 3 Test Item	nunit-test	[C#], F#, VB	Test/NUnit	2.2
xUnit Test Project	xunit	[C#], F#, VB	Test/xUnit	1.0
Razor Component	razorcomponent	[C#]	Web/ASP.NET	3.0

TEMPLATES	SHORT NAME	LANGUAGE	TAGS	INTRODUCED
Razor Page	<code>page</code>	[C#]	Web/ASP.NET	2.0
MVC ViewImports	<code>viewimports</code>	[C#]	Web/ASP.NET	2.0
MVC ViewStart	<code>viewstart</code>	[C#]	Web/ASP.NET	2.0
Blazor Server App	<code>blazorserver</code>	[C#]	Web/Blazor	3.0
Blazor Server App Empty	<code>blazorserver-empty</code>	[C#]	Web/Blazor	7.0
Blazor WebAssembly App	<code>blazorwasm</code>	[C#]	Web/Blazor/WebAssembly	3.1.300
Blazor WebAssembly App Empty	<code>blazorwasm-empty</code>	[C#]	Web/Blazor/WebAssembly	7.0
ASP.NET Core Empty	<code>web</code>	[C#], F#	Web/Empty	1.0
ASP.NET Core Web App (Model-View-Controller)	<code>mvc</code>	[C#], F#	Web/MVC	1.0
ASP.NET Core Web App	<code>webapp, razor</code>	[C#]	Web/MVC/Razor Pages	2.2, 2.0
ASP.NET Core with Angular	<code>angular</code>	[C#]	Web/MVC/SPA	2.0
ASP.NET Core with React.js	<code>react</code>	[C#]	Web/MVC/SPA	2.0
ASP.NET Core with React.js and Redux	<code>reactredux</code>	[C#]	Web/MVC/SPA	2.0
Razor Class Library	<code>razorclasslib</code>	[C#]	Web/Razor/Library/Razor Class Library	2.1
ASP.NET Core Web API	<code>webapi</code>	[C#], F#	Web/WebAPI	1.0
ASP.NET Core gRPC Service	<code>grpc</code>	[C#]	Web/gRPC	3.0
dotnet gitignore file	<code>gitignore</code>		Config	3.0
global.json file	<code>globaljson</code>		Config	2.0
NuGet Config	<code>nugetconfig</code>		Config	1.0
Dotnet local tool manifest file	<code>tool-manifest</code>		Config	3.0

TEMPLATES	SHORT NAME	LANGUAGE	TAGS	INTRODUCED
Web Config	webconfig		Config	1.0
Solution File	sln		Solution	1.0
Protocol Buffer File	proto		Web/gRPC	3.0
EditorConfig file	editorconfig (#editorconfig)		Config	6.0

Template options

Each template may have additional options available. The core templates have the following additional options:

console

- f|--framework <FRAMEWORK>

Specifies the [framework](#) to target. Available since .NET Core 3.0 SDK.

The following table lists the default values according to the SDK version number you're using:

SDK VERSION	DEFAULT VALUE
6.0	net6.0
5.0	net5.0
3.1	netcoreapp3.1

The ability to create a project for an earlier TFM depends on having that version of the SDK installed. For example, if you have only the .NET 6 SDK installed, then the only value available for `--framework` is `net6.0`. If you install the .NET 5 SDK, the value `net5.0` becomes available for `--framework`. If you install the .NET Core 3.1 SDK, `netcoreapp3.1` becomes available, and so on. So by specifying `--framework netcoreapp3.1` you can target .NET Core 3.1 even while running `dotnet new` in the .NET 6 SDK.

Alternatively, to create a project that targets a framework earlier than the SDK that you're using, you might be able to do it by installing the NuGet package for the template. [Common](#), [web](#), and [SPA](#) project types use different packages per target framework moniker (TFM). For example, to create a `console` project that targets `netcoreapp1.0`, run `dotnet new --install Microsoft.DotNet.Common.ProjectTemplates.1.x`.

- langVersion <VERSION_NUMBER>

Sets the `LangVersion` property in the created project file. For example, use `--langVersion 7.3` to use C# 7.3. Not supported for F#. Available since .NET Core 2.2 SDK.

For a list of default C# versions, see [Defaults](#).

- no-restore

If specified, doesn't execute an implicit restore during project creation. Available since .NET Core 2.2 SDK.

- `--use-program-main`

If specified, an explicit `Program` class and `Main` method will be used instead of top-level statements.

Available since .NET SDK 6.0.300. Default value: `false`.

`classlib`

- `-f|--framework <FRAMEWORK>`

Specifies the [framework](#) to target. Values: `net6.0`, `net5.0`, or `netcoreapp3.1` to create a .NET Class Library or `netstandard<version>` to create a .NET Standard Class Library. The default value for .NET 6 SDK is `net6.0`.

To create a project that targets a framework earlier than the SDK that you're using, see [--framework for console projects](#) earlier in this article.

- `--langVersion <VERSION_NUMBER>`

Sets the `LangVersion` property in the created project file. For example, use `--langVersion 7.3` to use C# 7.3. Not supported for F#. Available since .NET Core 2.2 SDK.

For a list of default C# versions, see [Defaults](#).

- `--no-restore`

Doesn't execute an implicit restore during project creation.

`wpf`, `wpflib`, `wpfcustomcontrollib`, `wpfusercontrollib`

- `-f|--framework <FRAMEWORK>`

Specifies the [framework](#) to target. For the .NET 6 SDK, the default value is `net6.0`. Available since .NET Core 3.1 SDK.

- `--langVersion <VERSION_NUMBER>`

Sets the `LangVersion` property in the created project file. For example, use `--langVersion 7.3` to use C# 7.3.

For a list of default C# versions, see [Defaults](#).

- `--no-restore`

Doesn't execute an implicit restore during project creation.

`winforms`, `winformslib`

- `--langVersion <VERSION_NUMBER>`

Sets the `LangVersion` property in the created project file. For example, use `--langVersion 7.3` to use C# 7.3.

For a list of default C# versions, see [Defaults](#).

- `--no-restore`

Doesn't execute an implicit restore during project creation.

worker , grpc

- `-f|--framework <FRAMEWORK>`

Specifies the [framework](#) to target. The default value is `netcoreapp3.1`. Available since .NET Core 3.1 SDK.

To create a project that targets a framework earlier than the SDK that you're using, see [--framework for console projects](#) earlier in this article.

- `--exclude-launch-settings`

Excludes `launchSettings.json` from the generated template.

- `--no-restore`

Doesn't execute an implicit restore during project creation.

- `--use-program-main`

If specified, an explicit `Program` class and `Main` method will be used instead of top-level statements.

Available since .NET SDK 6.0.300. Default value: `false`.

mstest , xunit

- `-f|--framework <FRAMEWORK>`

Specifies the [framework](#) to target. Option available since .NET Core 3.0 SDK.

The following table lists the default values according to the SDK version number you're using:

SDK VERSION	DEFAULT VALUE
6.0	<code>net6.0</code>
5.0	<code>net5.0</code>
3.1	<code>netcoreapp3.1</code>

The ability to create a project for an earlier TFM depends on having that version of the SDK installed. For example, if you have only the .NET 6 SDK installed, then the only value available for `--framework` is `net6.0`. If you install the .NET 5 SDK, the value `net5.0` becomes available for `--framework`. If you install the .NET Core 3.1 SDK, `netcoreapp3.1` becomes available, and so on. So by specifying `--framework netcoreapp3.1` you can target .NET Core 3.1 even while running `dotnet new` in the .NET 6 SDK.

- `-p|--enable-pack`

Enables packaging for the project using [dotnet pack](#).

- `--no-restore`

Doesn't execute an implicit restore during project creation.

nunit

- `-f|--framework <FRAMEWORK>`

Specifies the [framework](#) to target.

The following table lists the default values according to the SDK version number you're using:

SDK VERSION	DEFAULT VALUE
6.0	net6.0
5.0	net5.0
3.1	netcoreapp3.1

The ability to create a project for an earlier TFM depends on having that version of the SDK installed. For example, if you have only the .NET 6 SDK installed, then the only value available for `--framework` is `net6.0`. If you install the .NET 5 SDK, the value `net5.0` becomes available for `--framework`. If you install the .NET Core 3.1 SDK, `netcoreapp3.1` becomes available, and so on. So by specifying `--framework netcoreapp3.1` you can target .NET Core 3.1 even while running `dotnet new` in the .NET 6 SDK.

- `-p|--enable-pack`

Enables packaging for the project using [dotnet pack](#).

- `--no-restore`

Doesn't execute an implicit restore during project creation.

page

- `-na|--namespace <NAMESPACE_NAME>`

Namespace for the generated code. The default value is `MyApp.Namespace`.

- `-np|--no-pagemodel`

Creates the page without a PageModel.

viewimports , proto

- `-na|--namespace <NAMESPACE_NAME>`

Namespace for the generated code. The default value is `MyApp.Namespace`.

blazorserver

- `-au|--auth <AUTHENTICATION_TYPE>`

The type of authentication to use. The possible values are:

- `None` - No authentication (Default).
- `Individual` - Individual authentication.
- `IndividualB2C` - Individual authentication with Azure AD B2C.
- `SingleOrg` - Organizational authentication for a single tenant.
- `MultiOrg` - Organizational authentication for multiple tenants.

- `Windows` - Windows authentication.
- `--aad-b2c-instance <INSTANCE>`

The Azure Active Directory B2C instance to connect to. Use with `IndividualB2C` authentication. The default value is `https://login.microsoftonline.com/tfp/`.
- `-ssp|--susi-policy-id <ID>`

The sign-in and sign-up policy ID for this project. Use with `IndividualB2C` authentication.
- `-rp|--reset-password-policy-id <ID>`

The reset password policy ID for this project. Use with `IndividualB2C` authentication.
- `-ep|--edit-profile-policy-id <ID>`

The edit profile policy ID for this project. Use with `IndividualB2C` authentication.
- `--aad-instance <INSTANCE>`

The Azure Active Directory instance to connect to. Use with `SingleOrg` or `MultiOrg` authentication. The default value is `https://login.microsoftonline.com/`.
- `--client-id <ID>`

The Client ID for this project. Use with `IndividualB2C`, `SingleOrg`, or `MultiOrg` authentication. The default value is `11111111-1111-1111-1111-111111111111`.
- `--domain <DOMAIN>`

The domain for the directory tenant. Use with `SingleOrg` or `IndividualB2C` authentication. The default value is `qualified.domain.name`.
- `--tenant-id <ID>`

The TenantId ID of the directory to connect to. Use with `SingleOrg` authentication. The default value is `22222222-2222-2222-2222-222222222222`.
- `--callback-path <PATH>`

The request path within the application's base path of the redirect URI. Use with `SingleOrg` or `IndividualB2C` authentication. The default value is `/signin-oidc`.
- `-r|--org-read-access`

Allows this application read-access to the directory. Only applies to `SingleOrg` or `MultiOrg` authentication.
- `--exclude-launch-settings`

Excludes `launchSettings.json` from the generated template.
- `--no-https`

Turns off HTTPS. This option only applies if `Individual`, `IndividualB2C`, `SingleOrg`, or `MultiOrg` aren't being used for `--auth`.
- `-uld|--use-local-db`

Specifies LocalDB should be used instead of SQLite. Only applies to `Individual` or `IndividualB2C` authentication.

- `--no-restore`

Doesn't execute an implicit restore during project creation.

- `--kestrelHttpPort`

Port number to use for the HTTP endpoint in *launchSettings.json*.

- `--kestrelHttpsPort`

Port number to use for the HTTPS endpoint in *launchSettings.json*. This option is not applicable when the parameter `no-https` is used (but `no-https` is ignored when an individual or organizational authentication setting is chosen for `--auth`).

- `--use-program-main`

If specified, an explicit `Program` class and `Main` method will be used instead of top-level statements.

Available since .NET SDK 6.0.300. Default value: `false`.

blazorwasm

- `-f|--framework <FRAMEWORK>`

Specifies the [framework](#) to target.

The following table lists the default values according to the SDK version number you're using:

SDK VERSION	DEFAULT VALUE
7.0	<code>net7.0</code>
6.0	<code>net6.0</code>
5.0	<code>net5.0</code>
3.1	<code>netcoreapp3.1</code>

To create a project that targets a framework earlier than the SDK that you're using, see [--framework for console projects](#) earlier in this article.

- `--no-restore`

Doesn't execute an implicit restore during project creation.

- `-ho|--hosted`

Includes an ASP.NET Core host for the Blazor WebAssembly app.

- `-au|--auth <AUTHENTICATION_TYPE>`

The type of authentication to use. The possible values are:

- `None` - No authentication (Default).
- `Individual` - Individual authentication.
- `IndividualB2C` - Individual authentication with Azure AD B2C.
- `SingleOrg` - Organizational authentication for a single tenant.

- `--authority <AUTHORITY>`

The authority of the OIDC provider. Use with `Individual` authentication. The default value is `https://login.microsoftonline.com/`.

- `--aad-b2c-instance <INSTANCE>`

The Azure Active Directory B2C instance to connect to. Use with `IndividualB2C` authentication. The default value is `https://aadB2CInstance.b2clogin.com/`.

- `-ssp|--sus-i-policy-id <ID>`

The sign-in and sign-up policy ID for this project. Use with `IndividualB2C` authentication.

- `--aad-instance <INSTANCE>`

The Azure Active Directory instance to connect to. Use with `SingleOrg` authentication. The default value is `https://login.microsoftonline.com/`.

- `--client-id <ID>`

The Client ID for this project. Use with `IndividualB2C`, `SingleOrg`, or `Individual` authentication in standalone scenarios. The default value is `33333333-3333-3333-3333333333333333`.

- `--domain <DOMAIN>`

The domain for the directory tenant. Use with `SingleOrg` or `IndividualB2C` authentication. The default value is `qualified.domain.name`.

- `--app-id-uri <URI>`

The App ID Uri for the server API you want to call. Use with `SingleOrg` or `IndividualB2C` authentication. The default value is `api.id.uri`.

- `--api-client-id <ID>`

The Client ID for the API that the server hosts. Use with `SingleOrg` or `IndividualB2C` authentication. The default value is `11111111-1111-1111-1111111111111111`.

- `-s|--default-scope <SCOPE>`

The API scope the client needs to request to provision an access token. Use with `SingleOrg` or `IndividualB2C` authentication. The default value is `user_impersonation`.

- `--tenant-id <ID>`

The TenantId ID of the directory to connect to. Use with `SingleOrg` authentication. The default value is `22222222-2222-2222-2222-222222222222`.

- `-r|--org-read-access`

Allows this application read-access to the directory. Only applies to `SingleOrg` authentication.

- `--exclude-launch-settings`

Excludes `launchSettings.json` from the generated template.

- `-p|--pwa`

produces a Progressive Web Application (PWA) supporting installation and offline use.

- `--no-https`

Turns off HTTPS. This option only applies if `Individual`, `IndividualB2C`, or `SingleOrg` aren't being used

for `--auth`.

- `-uld|--use-local-db`

Specifies LocalDB should be used instead of SQLite. Only applies to `Individual` or `IndividualB2C` authentication.

- `--called-api-url <URL>`

URL of the API to call from the web app. Only applies to `SingleOrg` or `IndividualB2C` authentication without an ASP.NET Core host specified. The default value is `https://graph.microsoft.com/v1.0/me`.

- `--calls-graph`

Specifies if the web app calls Microsoft Graph. Only applies to `SingleOrg` authentication.

- `--called-api-scopes <SCOPES>`

Scopes to request to call the API from the web app. Only applies to `SingleOrg` or `IndividualB2C` authentication without an ASP.NET Core host specified. The default is `user.read`.

- `--kestrelHttpPort`

Port number to use for the HTTP endpoint in `launchSettings.json`.

- `--kestrelHttpsPort`

Port number to use for the HTTPS endpoint in `launchSettings.json`. This option is not applicable when the parameter `no-https` is used (but `no-https` is ignored when an individual or organizational authentication setting is chosen for `--auth`).

- `--use-program-main`

If specified, an explicit `Program` class and `Main` method will be used instead of top-level statements. Available since .NET SDK 6.0.300. Default value: `false`.

web

- `--exclude-launch-settings`

Excludes `launchSettings.json` from the generated template.

- `-f|--framework <FRAMEWORK>`

Specifies the `framework` to target. Option not available in .NET Core 2.2 SDK.

The following table lists the default values according to the SDK version number you're using:

SDK VERSION	DEFAULT VALUE
6.0	<code>net6.0</code>
5.0	<code>net5.0</code>
3.1	<code>netcoreapp3.1</code>
3.0	<code>netcoreapp3.0</code>

SDK VERSION	DEFAULT VALUE
2.1	netcoreapp2.1

To create a project that targets a framework earlier than the SDK that you're using, see [--framework](#) for [console](#) projects earlier in this article.

- `--no-restore`

Doesn't execute an implicit restore during project creation.

- `--no-https`

Turns off HTTPS.

- `--kestrelHttpPort`

Port number to use for the HTTP endpoint in *launchSettings.json*.

- `--kestrelHttpsPort`

Port number to use for the HTTPS endpoint in *launchSettings.json*. This option is not applicable when the parameter `no-https` is used (but `no-https` is ignored when an individual or organizational authentication setting is chosen for `--auth`).

- `--use-program-main`

If specified, an explicit `Program` class and `Main` method will be used instead of top-level statements.

Available since .NET SDK 6.0.300. Default value: `false`.

`mvc` , `webapp`

- `-au|--auth <AUTHENTICATION_TYPE>`

The type of authentication to use. The possible values are:

- `None` - No authentication (Default).
- `Individual` - Individual authentication.
- `IndividualB2C` - Individual authentication with Azure AD B2C.
- `SingleOrg` - Organizational authentication for a single tenant.
- `MultiOrg` - Organizational authentication for multiple tenants.
- `Windows` - Windows authentication.

- `--aad-b2c-instance <INSTANCE>`

The Azure Active Directory B2C instance to connect to. Use with `IndividualB2C` authentication. The default value is `https://login.microsoftonline.com/tfp/`.

- `-ssp|--sus-i-policy-id <ID>`

The sign-in and sign-up policy ID for this project. Use with `IndividualB2C` authentication.

- `-rp|--reset-password-policy-id <ID>`

The reset password policy ID for this project. Use with `IndividualB2C` authentication.

- `-ep|--edit-profile-policy-id <ID>`

The edit profile policy ID for this project. Use with `IndividualB2C` authentication.

- `--aad-instance <INSTANCE>`

The Azure Active Directory instance to connect to. Use with `SingleOrg` or `MultiOrg` authentication. The default value is `https://login.microsoftonline.com/`.

- `--client-id <ID>`

The Client ID for this project. Use with `IndividualB2C`, `SingleOrg`, or `MultiOrg` authentication. The default value is `11111111-1111-1111-1111-111111111111`.

- `--domain <DOMAIN>`

The domain for the directory tenant. Use with `SingleOrg` or `IndividualB2C` authentication. The default value is `qualified.domain.name`.

- `--tenant-id <ID>`

The TenantId ID of the directory to connect to. Use with `SingleOrg` authentication. The default value is `22222222-2222-2222-2222-222222222222`.

- `--callback-path <PATH>`

The request path within the application's base path of the redirect URI. Use with `SingleOrg` or `IndividualB2C` authentication. The default value is `/signin-oidc`.

- `-r|--org-read-access`

Allows this application read-access to the directory. Only applies to `SingleOrg` or `MultiOrg` authentication.

- `--exclude-launch-settings`

Excludes `launchSettings.json` from the generated template.

- `--no-https`

Turns off HTTPS. This option only applies if `Individual`, `IndividualB2C`, `SingleOrg`, or `MultiOrg` aren't being used.

- `-uld|--use-local-db`

Specifies LocalDB should be used instead of SQLite. Only applies to `Individual` or `IndividualB2C` authentication.

- `-f|--framework <FRAMEWORK>`

Specifies the [framework](#) to target. Option available since .NET Core 3.0 SDK.

The following table lists the default values according to the SDK version number you're using:

SDK VERSION	DEFAULT VALUE
6.0	<code>net6.0</code>
5.0	<code>net5.0</code>
3.1	<code>netcoreapp3.1</code>

SDK VERSION	DEFAULT VALUE
3.0	netcoreapp3.0

To create a project that targets a framework earlier than the SDK that you're using, see [--framework](#) for [console projects](#) earlier in this article.

- `--no-restore`

Doesn't execute an implicit restore during project creation.

- `--use-browserlink`

Includes BrowserLink in the project. Option not available in .NET Core 2.2 and 3.1 SDK.

- `-rrc|--razor-runtime-compilation`

Determines if the project is configured to use [Razor runtime compilation](#) in Debug builds. Option available since .NET Core 3.1.201 SDK.

- `--kestrelHttpPort`

Port number to use for the HTTP endpoint in *launchSettings.json*.

- `--kestrelHttpsPort`

Port number to use for the HTTPS endpoint in *launchSettings.json*. This option is not applicable when the parameter `no-https` is used (but `no-https` is ignored when an individual or organizational authentication setting is chosen for `--auth`).

- `--use-program-main`

If specified, an explicit `Program` class and `Main` method will be used instead of top-level statements.

Available since .NET SDK 6.0.300. Default value: `false`.

`angular` , `react`

- `-au|--auth <AUTHENTICATION_TYPE>`

The type of authentication to use. Available since .NET Core 3.0 SDK.

The possible values are:

- `None` - No authentication (Default).
- `Individual` - Individual authentication.

- `--exclude-launch-settings`

Excludes *launchSettings.json* from the generated template.

- `--no-restore`

Doesn't execute an implicit restore during project creation.

- `--no-https`

Turns off HTTPS. This option only applies if authentication is `None`.

- `-uld|--use-local-db`

Specifies LocalDB should be used instead of SQLite. Only applies to `Individual` or `IndividualB2C` authentication. Available since .NET Core 3.0 SDK.

- `-f|--framework <FRAMEWORK>`

Specifies the [framework](#) to target. Option not available in .NET Core 2.2 SDK.

The following table lists the default values according to the SDK version number you're using:

SDK VERSION	DEFAULT VALUE
6.0	<code>net6.0</code>
5.0	<code>net5.0</code>
3.1	<code>netcoreapp3.1</code>
3.0	<code>netcoreapp3.0</code>
2.1	<code>netcoreapp2.0</code>

To create a project that targets a framework earlier than the SDK that you're using, see [--framework](#) for [console projects](#) earlier in this article.

- `--kestrelHttpPort`

Port number to use for the HTTP endpoint in *launchSettings.json*.

- `--kestrelHttpsPort`

Port number to use for the HTTPS endpoint in *launchSettings.json*. This option is not applicable when the parameter `no-https` is used (but `no-https` is ignored when an individual or organizational authentication setting is chosen for `--auth`).

- `--use-program-main`

If specified, an explicit `Program` class and `Main` method will be used instead of top-level statements.

Available since .NET SDK 6.0.300. Default value: `false`.

reactredux

- `--exclude-launch-settings`

Excludes *launchSettings.json* from the generated template.

- `-f|--framework <FRAMEWORK>`

Specifies the [framework](#) to target. Option not available in .NET Core 2.2 SDK.

The following table lists the default values according to the SDK version number you're using:

SDK VERSION	DEFAULT VALUE
6.0	<code>net6.0</code>
5.0	<code>net5.0</code>

SDK VERSION	DEFAULT VALUE
3.1	netcoreapp3.1
3.0	netcoreapp3.0
2.1	netcoreapp2.0

To create a project that targets a framework earlier than the SDK that you're using, see [--framework](#) for [console projects](#) earlier in this article.

- `--no-restore`

Doesn't execute an implicit restore during project creation.

- `--no-https`

Turns off HTTPS.

- `--kestrelHttpPort`

Port number to use for the HTTP endpoint in *launchSettings.json*.

- `--kestrelHttpsPort`

Port number to use for the HTTPS endpoint in *launchSettings.json*. This option is not applicable when the parameter `no-https` is used (but `no-https` is ignored when an individual or organizational authentication setting is chosen for `--auth`).

razorclasslib

- `--no-restore`

Doesn't execute an implicit restore during project creation.

- `-s|--support-pages-and-views`

Supports adding traditional Razor pages and Views in addition to components to this library. Available since .NET Core 3.0 SDK.

webapi

- `-au|--auth <AUTHENTICATION_TYPE>`

The type of authentication to use. The possible values are:

- `None` - No authentication (Default).
- `IndividualB2C` - Individual authentication with Azure AD B2C.
- `SingleOrg` - Organizational authentication for a single tenant.
- `Windows` - Windows authentication.

- `--aad-b2c-instance <INSTANCE>`

The Azure Active Directory B2C instance to connect to. Use with `IndividualB2C` authentication. The default value is `https://login.microsoftonline.com/tfp/`.

- `-minimal`

Create a project that uses the [ASP.NET Core minimal API](#).

- `-ssp|--susi-policy-id <ID>`

The sign-in and sign-up policy ID for this project. Use with `IndividualB2C` authentication.

- `--aad-instance <INSTANCE>`

The Azure Active Directory instance to connect to. Use with `SingleOrg` authentication. The default value is `https://login.microsoftonline.com/`.

- `--client-id <ID>`

The Client ID for this project. Use with `IndividualB2C` or `SingleOrg` authentication. The default value is `11111111-1111-1111-1111-111111111111`.

- `--domain <DOMAIN>`

The domain for the directory tenant. Use with `IndividualB2C` or `SingleOrg` authentication. The default value is `qualified.domain.name`.

- `--tenant-id <ID>`

The TenantId ID of the directory to connect to. Use with `SingleOrg` authentication. The default value is `22222222-2222-2222-2222-222222222222`.

- `-r|--org-read-access`

Allows this application read-access to the directory. Only applies to `SingleOrg` authentication.

- `--exclude-launch-settings`

Excludes `launchSettings.json` from the generated template.

- `--no-https`

Turns off HTTPS. `app.UseHsts` and `app.UseHttpsRedirection` aren't added to `Startup.Configure`. This option only applies if `IndividualB2C` or `SingleOrg` aren't being used for authentication.

- `-uld|--use-local-db`

Specifies LocalDB should be used instead of SQLite. Only applies to `IndividualB2C` authentication.

- `-f|--framework <FRAMEWORK>`

Specifies the [framework](#) to target. Option not available in .NET Core 2.2 SDK.

The following table lists the default values according to the SDK version number you're using:

SDK VERSION	DEFAULT VALUE
6.0	<code>net6.0</code>
5.0	<code>net5.0</code>
3.1	<code>netcoreapp3.1</code>
3.0	<code>netcoreapp3.0</code>

SDK VERSION	DEFAULT VALUE
2.1	netcoreapp2.1

To create a project that targets a framework earlier than the SDK that you're using, see [--framework](#) for [console projects](#) earlier in this article.

- `--no-restore`

Doesn't execute an implicit restore during project creation.

- `--use-program-main`

If specified, an explicit `Program` class and `Main` method will be used instead of top-level statements.

Available since .NET SDK 6.0.300. Default value: `false`.

globaljson

- `--sdk-version <VERSION_NUMBER>`

Specifies the version of the .NET SDK to use in the *global.json* file.

editorconfig

Creates an *.editorconfig* file for configuring code style preferences.

- `--empty`

Creates an empty *.editorconfig* instead of the defaults for .NET.

See also

- [dotnet new command](#)
- [dotnet new --list option](#)
- [Custom templates for dotnet new](#)
- [Create a custom template for dotnet new](#)
- [Implicit using directives](#)

Custom templates for dotnet new

9/20/2022 • 10 minutes to read • [Edit Online](#)

The [.NET SDK](#) comes with many templates already installed and ready for you to use. The `dotnet new` command isn't only the way to use a template, but also how to install and uninstall templates. You can create your own custom templates for any type of project, such as an app, service, tool, or class library. You can even create a template that outputs one or more independent files, such as a configuration file.

You can install custom templates from a NuGet package on any NuGet feed, by referencing a NuGet `.nupkg` file directly, or by specifying a file system directory that contains the template. The template engine offers features that allow you to replace values, include and exclude files, and execute custom processing operations when your template is used.

The template engine is open source, and the online code repository is at [dotnettemplating](#) on GitHub. More templates, including templates from third parties, can be found using `dotnet new --search`. For more information about creating and using custom templates, see [How to create your own templates for dotnet new](#) and the [dotnettemplating GitHub repo Wiki](#).

NOTE

Template examples are available at the [dotnet/dotnet-template-samples](#) GitHub repository. However, while these examples are good resource for learning how the templates work, the repository is archived and no longer maintained. The examples may be out of date and no longer working.

To follow a walkthrough and create a template, see the [Create a custom template for dotnet new](#) tutorial.

.NET default templates

When you install the [.NET SDK](#), you receive over a dozen built-in templates for creating projects and files, including console apps, class libraries, unit test projects, ASP.NET Core apps (including [Angular](#) and [React](#) projects), and configuration files. To list the built-in templates, run the `dotnet new` command with the `-l|--list` option:

```
dotnet new --list
```

Configuration

A template is composed of the following parts:

- Source files and folders.
- A configuration file (`template.json`).

Source files and folders

The source files and folders include whatever files and folders you want the template engine to use when the `dotnet new <TEMPLATE>` command is run. The template engine is designed to use *runnable projects* as source code to produce projects. This has several benefits:

- The template engine doesn't require you to inject special tokens into your project's source code.
- The code files aren't special files or modified in any way to work with the template engine. So, the tools you normally use when working with projects also work with template content.

- You build, run, and debug your template projects just like you do for any of your other projects.
- You can quickly create a template from an existing project just by adding a `./template.config/template.json` configuration file to the project.

Files and folders stored in the template aren't limited to formal .NET project types. Source files and folders may consist of any content that you wish to create when the template is used, even if the template engine produces just one file as its output.

Files generated by the template can be modified based on logic and settings you've provided in the `template.json` configuration file. The user can override these settings by passing options to the `dotnet new <TEMPLATE>` command. A common example of custom logic is providing a name for a class or variable in the code file that's deployed by a template.

template.json

The `template.json` file is placed in a `.template.config` folder in the root directory of the template. The file provides configuration information to the template engine. The minimum configuration requires the members shown in the following table, which is sufficient to create a functional template.

MEMBER	TYPE	DESCRIPTION
<code>\$schema</code>	URI	The JSON schema for the <code>template.json</code> file. Editors that support JSON schemas enable JSON-editing features when the schema is specified. For example, Visual Studio Code requires this member to enable IntelliSense. Use a value of http://json.schemastore.org/template .
<code>author</code>	string	The author of the template.
<code>classifications</code>	array(string)	Zero or more characteristics of the template that a user might use to find the template when searching for it. The classifications also appear in the <code>Tags</code> column when it appears in a list of templates produced by using the <code>dotnet new -l</code> or <code>dotnet new --list</code> command.
<code>identity</code>	string	A unique name for this template.
<code>name</code>	string	The name for the template that users should see.
<code>shortName</code>	string	A default shorthand name for selecting the template that applies to environments where the template name is specified by the user, not selected via a GUI. For example, the short name is useful when using templates from a command prompt with CLI commands.

MEMBER	TYPE	DESCRIPTION
<code>sourceName</code>	string	The name in the source tree to replace with the name the user specifies. The template engine will look for any occurrence of the <code>sourceName</code> mentioned in the config file and replace it in file names and file contents. The value to be replaced with can be given using the <code>-n</code> or <code>--name</code> options while running a template. If no name is specified, the current directory is used.
<code>preferNameDirectory</code>	boolean	Indicates whether to create a directory for the template if name is specified but an output directory is not set (instead of creating the content directly in the current directory). The default value is false.

The full schema for the `template.json` file is found at the [JSON Schema Store](#). For more information about the `template.json` file, see the [dotnet templating wiki](#). For deeper examples and information on how to make your templates visible in Visual Studio, check out the [resources that Sayed Hashimi has created](#).

Example

For example, here is a template folder that contains two content files: `console.cs` and `readme.txt`. Take notice that there is the required folder named `.template.config` that contains the `template.json` file.

```

└── mytemplate
    ├── console.cs
    └── readme.txt
    └── .template.config
        └── template.json

```

The `template.json` file looks like the following:

```
{
  "$schema": "http://json.schemastore.org/template",
  "author": "Travis Chau",
  "classifications": [ "Common", "Console" ],
  "identity": "AdatumCorporation.ConsoleTemplate.CSharp",
  "name": "Adatum Corporation Console Application",
  "shortName": "adatumconsole"
}
```

The `mytemplate` folder is an installable template package. Once the package is installed, the `shortName` can be used with the `dotnet new` command. For example, `dotnet new adatumconsole` would output the `console.cs` and `readme.txt` files to the current folder.

Packing a template into a NuGet package (nupkg file)

A custom template is packed with the `dotnet pack` command and a `.csproj` file. Alternatively, [NuGet](#) can be used with the `nuget pack` command along with a `.nuspec` file. However, NuGet requires the .NET Framework on Windows and [Mono](#) on Linux and macOS.

The `.csproj` file is slightly different from a traditional code-project `.csproj` file. Note the following settings:

1. The `<PackageType>` setting is added and set to `Template`.
2. The `<PackageVersion>` setting is added and set to a valid [NuGet version number](#).
3. The `<PackageId>` setting is added and set to a unique identifier. This identifier is used to uninstall the template pack and is used by NuGet feeds to register your template pack.
4. Generic metadata settings should be set: `<Title>`, `<Authors>`, `<Description>`, and `<PackageTags>`.
5. The `<TargetFramework>` setting must be set, even though the binary produced by the template process isn't used. In the example below it's set to `netstandard2.0`.

A template package, in the form of a `.nupkg` NuGet package, requires that all templates be stored in the `content` folder within the package. There are a few more settings to add to a `.csproj` file to ensure that the generated `.nupkg` can be installed as a template pack:

1. The `<IncludeContentInPack>` setting is set to `true` to include any file the project sets as `content` in the NuGet package.
2. The `<IncludeBuildOutput>` setting is set to `false` to exclude all binaries generated by the compiler from the NuGet package.
3. The `<ContentTargetFolders>` setting is set to `content`. This makes sure that the files set as `content` are stored in the `content` folder in the NuGet package. This folder in the NuGet package is parsed by the dotnet template system.

An easy way to exclude all code files from being compiled by your template project is by using the `<Compile Remove="****" />` item in your project file, inside an `<ItemGroup>` element.

An easy way to structure your template pack is to put all templates in individual folders, and then each template folder inside of a `templates` folder that is located in the same directory as your `.csproj` file. This way, you can use a single project item to include all files and folders in the `templates` as `content`. Inside of an `<ItemGroup>` element, create a `<Content Include="templates****" Exclude="templates**\bin**;templates**\obj**" />` item.

Here is an example `.csproj` file that follows all of the guidelines above. It packs the `templates` child folder to the `content` package folder and excludes any code file from being compiled.

```

<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <PackageType>Template</PackageType>
    <PackageVersion>1.0</PackageVersion>
    <PackageId>AdatumCorporation.Utility.Templates</PackageId>
    <Title>AdatumCorporation Templates</Title>
    <Authors>Me</Authors>
    <Description>Templates to use when creating an application for Adatum Corporation.</Description>
    <PackageTags>dotnet-new;templates;contoso</PackageTags>
    <TargetFramework>netstandard2.0</TargetFramework>

    <IncludeContentInPack>true</IncludeContentInPack>
    <IncludeBuildOutput>false</IncludeBuildOutput>
    <ContentTargetFolders>content</ContentTargetFolders>
  </PropertyGroup>

  <ItemGroup>
    <Content Include="templates\**\**" Exclude="templates\**\bin\**;templates\**\obj\**" />
    <Compile Remove="**\**" />
  </ItemGroup>

</Project>

```

The example below demonstrates the file and folder structure of using a `.csproj` to create a template pack. The `MyDotnetTemplates.csproj` file and `templates` folder are both located at the root of a directory named `project_folder`. The `templates` folder contains two templates, `mytemplate1` and `mytemplate2`. Each template has

content files and a `.template.config` folder with a `template.json` config file.

```
project_folder
|   MyDotnetTemplates.csproj
|
└── templates
    ├── mytemplate1
    |   ├── console.cs
    |   └── readme.txt
    |
    └── .template.config
        template.json
    └── mytemplate2
        ├── otherfile.cs
        └── .template.config
            template.json
```

NOTE

To ensure that the template package appears in `dotnet new --search` result, set the NuGet package type to `Template`.

Installing a template package

Use the `dotnet new --install` command to install a template package.

To install a template package from a NuGet package stored at nuget.org

Use the NuGet package identifier to install a template package.

```
dotnet new --install <NUGET_PACKAGE_ID>
```

To install a template package from a local nupkg file

Provide the path to a `.nupkg` NuGet package file.

```
dotnet new --install <PATH_TO_NUPKG_FILE>
```

To install a template package from a file system directory

Templates can be installed from a template folder, such as the `mytemplate1` folder from the example above. Specify the folder path of the `.template.config` folder. The path to the template directory does not need to be absolute.

```
dotnet new --install <FILE_SYSTEM_DIRECTORY>
```

Get a list of installed template packages

The `uninstall` command, without any other parameters, will list all installed template packages and included templates.

```
dotnet new --uninstall
```

That command returns something similar to the following output:

```
Template Instantiation Commands for .NET CLI

Currently installed items:
Microsoft.DotNet.Common.ItemTemplates
    Templates:
        global.json file (globaljson)
        NuGet Config (nugetconfig)
        Solution File (sln)
        Dotnet local tool manifest file (tool-manifest)
        Web Config (webconfig)
Microsoft.DotNet.Common.ProjectTemplates.3.0
    Templates:
        Class library (classlib) C#
        Class library (classlib) F#
        Class library (classlib) VB
        Console Application (console) C#
        Console Application (console) F#
        Console Application (console) VB
...
...
```

The first level of items after `Currently installed items:` are the identifiers used in uninstalling a template package. And in the example above, `Microsoft.DotNet.Common.ItemTemplates` and `Microsoft.DotNet.Common.ProjectTemplates.3.0` are listed. If the template package was installed by using a file system path, this identifier will be the folder path of the `.template.config` folder.

Uninstalling a template package

Use the `dotnet new -u|--uninstall` command to uninstall a template package.

If the package was installed by either a NuGet feed or by a `.nupkg` file directly, provide the identifier.

```
dotnet new --uninstall <NUGET_PACKAGE_ID>
```

If the package was installed by specifying a path to the `.template.config` folder, use that path to uninstall the package. You can see the absolute path of the template package in the output provided by the `dotnet new --uninstall` command. For more information, see the [Get a list of installed templates](#) section above.

```
dotnet new --uninstall <FILE_SYSTEM_DIRECTORY>
```

Create a project using a custom template

After a template is installed, use the template by executing the `dotnet new <TEMPLATE>` command as you would with any other pre-installed template. You can also specify [options](#) to the `dotnet new` command, including template-specific options you configured in the template settings. Supply the template's short name directly to the command:

```
dotnet new <TEMPLATE>
```

See also

- [Create a custom template for dotnet new \(tutorial\)](#)
- [dotnettemplating GitHub repo Wiki](#)

- [dotnet/dotnet-template-samples GitHub repo](#)
- [How to create your own templates for dotnet new](#)
- [*template.json* schema at the JSON Schema Store](#)

dotnet nuget delete

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 3.1 SDK and later versions

Name

`dotnet nuget delete` - Deletes or unlists a package from the server.

Synopsis

```
dotnet nuget delete [<PACKAGE_NAME> <PACKAGE_VERSION>] [--force-english-output]
    [--interactive] [-k|--api-key <API_KEY>] [--no-service-endpoint]
    [--non-interactive] [-s|--source <SOURCE>]
```

```
dotnet nuget delete -h|--help
```

Description

The `dotnet nuget delete` command deletes or unlists a package from the server. For [nuget.org](#), the action is to unlist the package.

Arguments

- `<PACKAGE_NAME>`

Name/ID of the package to delete.

- `<PACKAGE_VERSION>`

Version of the package to delete.

Options

- `--force-english-output`

Forces the application to run using an invariant, English-based culture.

- `-?|-h|--help`

Prints out a description of how to use the command.

- `--interactive`

Allows the command to stop and wait for user input or action. For example, to complete authentication.

Available since .NET Core 3.0 SDK.

- `-k|--api-key <API_KEY>`

The API key for the server.

- `--no-service-endpoint`

Doesn't append "api/v2/package" to the source URL.

- `--non-interactive`

Doesn't prompt for user input or confirmations.

- `-s|--source <SOURCE>`

Specifies the server URL. Supported URLs for nuget.org include `https://www.nuget.org`, `https://www.nuget.org/api/v3`, and `https://www.nuget.org/api/v2/package`. For private feeds, replace the host name (for example, `%hostname%/api/v3`).

Examples

- Deletes version 1.0 of package `Microsoft.AspNetCore.Mvc`:

```
dotnet nuget delete Microsoft.AspNetCore.Mvc 1.0
```

- Deletes version 1.0 of package `Microsoft.AspNetCore.Mvc`, not prompting user for credentials or other input:

```
dotnet nuget delete Microsoft.AspNetCore.Mvc 1.0 --non-interactive
```

dotnet nuget locals

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 3.1 SDK and later versions

Name

`dotnet nuget locals` - Clears or lists local NuGet resources.

Synopsis

```
dotnet nuget locals <CACHE_LOCATION> [(-c|--clear)|(-l|--list)] [--force-english-output]
```

```
dotnet nuget locals -h|--help
```

Description

The `dotnet nuget locals` command clears or lists local NuGet resources in the http-request cache, temporary cache, or machine-wide global packages folder.

Arguments

- `CACHE_LOCATION`

The cache location to list or clear. It accepts one of the following values:

- `all` - Indicates that the specified operation is applied to all cache types: http-request cache, global packages cache, and the temporary cache.
- `http-cache` - Indicates that the specified operation is applied only to the http-request cache. The other cache locations aren't affected.
- `global-packages` - Indicates that the specified operation is applied only to the global packages cache. The other cache locations aren't affected.
- `temp` - Indicates that the specified operation is applied only to the temporary cache. The other cache locations aren't affected.

Options

- `--force-english-output`

Forces the application to run using an invariant, English-based culture.

- `-?|-h|--help`

Prints out a description of how to use the command.

- `-c|--clear`

The clear option executes a clear operation on the specified cache type. The contents of the cache directories are deleted recursively. The executing user/group must have permission to the files in the cache directories. If not, an error is displayed indicating the files/folders that weren't cleared.

- `-l|--list`

The list option is used to display the location of the specified cache type.

Examples

- Displays the paths of all the local cache directories (http-cache directory, global-packages cache directory, and temporary cache directory):

```
dotnet nuget locals all -l
```

- Displays the path for the local http-cache directory:

```
dotnet nuget locals http-cache --list
```

- Clears all files from all local cache directories (http-cache directory, global-packages cache directory, and temporary cache directory):

```
dotnet nuget locals all --clear
```

- Clears all files in local global-packages cache directory:

```
dotnet nuget locals global-packages -c
```

- Clears all files in local temporary cache directory:

```
dotnet nuget locals temp -c
```

Troubleshooting

For information on common problems and errors while using the `dotnet nuget locals` command, see [Managing the NuGet cache](#).

dotnet nuget push

9/20/2022 • 3 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 3.1 SDK and later versions

Name

`dotnet nuget push` - Pushes a package to the server and publishes it.

Synopsis

```
dotnet nuget push [<ROOT>] [-d|--disable-buffering] [--force-english-output]
    [--interactive] [-k|--api-key <API_KEY>] [-n|--no-symbols]
    [--no-service-endpoint] [-s|--source <SOURCE>] [--skip-duplicate]
    [-sk|--symbol-api-key <API_KEY>] [-ss|--symbol-source <SOURCE>]
    [-t|--timeout <TIMEOUT>]

dotnet nuget push -h|--help
```

Description

The `dotnet nuget push` command pushes a package to the server and publishes it. The push command uses server and credential details found in the system's NuGet config file or chain of config files. For more information on config files, see [Configuring NuGet Behavior](#). NuGet's default configuration is obtained by loading `%AppData%\NuGet\NuGet.config` (Windows) or `$HOME/.nuget/NuGet/NuGet.Config` (Linux/macOS), then loading any `nuget.config` or `.nuget\NuGet.config` starting from the root of drive and ending in the current directory.

The command pushes an existing package. It doesn't create a package. To create a package, use `dotnet pack`.

Arguments

- `ROOT`

Specifies the file path to the package to be pushed.

Options

- `-d|--disable-buffering`

Disables buffering when pushing to an HTTP(S) server to reduce memory usage.

- `--force-english-output`

Forces the application to run using an invariant, English-based culture.

- `-?|-h|--help`

Prints out a description of how to use the command.

- `--interactive`

Allows the command to stop and wait for user input or action. For example, to complete authentication.

Available since .NET Core 3.0 SDK.

- `-k|--api-key <API_KEY>`

The API key for the server.

- `-n|--no-symbols`

Doesn't push symbols (even if present).

- `--no-service-endpoint`

Doesn't append "api/v2/package" to the source URL.

- `-s|--source <SOURCE>`

Specifies the server URL. NuGet identifies a UNC or local folder source and simply copies the file there instead of pushing it using HTTP.

IMPORTANT

Starting with NuGet 3.4.2, this is a mandatory parameter unless the NuGet config file specifies a `DefaultPushSource` value. For more information, see [Configuring NuGet behavior](#).

- `--skip-duplicate`

When pushing multiple packages to an HTTP(S) server, treats any 409 Conflict response as a warning so that the push can continue.

- `-sk|--symbol-api-key <API_KEY>`

The API key for the symbol server.

- `-ss|--symbol-source <SOURCE>`

Specifies the symbol server URL.

- `-t|--timeout <TIMEOUT>`

Specifies the timeout for pushing to a server in seconds. Defaults to 300 seconds (5 minutes). Specifying 0 applies the default value.

Examples

- Push `foo.nupkg` to the default push source specified in the NuGet config file, using an API key:

```
dotnet nuget push foo.nupkg -k 4003d786-cc37-4004-bfdf-c4f3e8ef9b3a
```

- Push `foo.nupkg` to the official NuGet server, specifying an API key:

```
dotnet nuget push foo.nupkg -k 4003d786-cc37-4004-bfdf-c4f3e8ef9b3a -s  
https://api.nuget.org/v3/index.json
```

- Push `foo.nupkg` to the custom push source `https://customsource`, specifying an API key:

```
dotnet nuget push foo.nupkg -k 4003d786-cc37-4004-bfdf-c4f3e8ef9b3a -s https://customsource/
```

- Push *foo.nupkg* to the default push source specified in the NuGet config file:

```
dotnet nuget push foo.nupkg
```

- Push *foo.symbols.nupkg* to the default symbols source:

```
dotnet nuget push foo.symbols.nupkg
```

- Push *foo.nupkg* to the default push source specified in the NuGet config file, with a 360-second timeout:

```
dotnet nuget push foo.nupkg --timeout 360
```

- Push all *.nupkg* files in the current directory to the default push source specified in the NuGet config file:

```
dotnet nuget push "*.nupkg"
```

NOTE

If this command doesn't work, it might be due to a bug that existed in older versions of the SDK (.NET Core 2.1 SDK and earlier versions). To fix this, upgrade your SDK version or run the following command instead:

```
dotnet nuget push "**/*.nupkg"
```

NOTE

The enclosing quotes are required for shells such as bash that perform file globbing. For more information, see [NuGet/Home#4393](#).

- Push all *.nupkg* files to the default push source specified in the NuGet config file, even if a 409 Conflict response is returned by an HTTP(S) server:

```
dotnet nuget push "*.nupkg" --skip-duplicate
```

- Push all *.nupkg* files in the current directory to a local feed directory:

```
dotnet nuget push "*.nupkg" -s c:\mydir
```

- For pushing to Azure Artifacts, [see Azure Artifacts' push documentation](#).

This command doesn't store packages in a hierarchical folder structure, which is recommended to optimize performance. For more information, see [Local feeds](#).

dotnet nuget add source

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 3.1.200 SDK and later versions

Name

`dotnet nuget add source` - Add a NuGet source.

Synopsis

```
dotnet nuget add source <PACKAGE_SOURCE_PATH> [--name <SOURCE_NAME>] [--username <USER>]
  [--password <PASSWORD>] [--store-password-in-clear-text]
  [--valid-authentication-types <TYPES>] [--configfile <FILE>]

dotnet nuget add source -h|--help
```

Description

The `dotnet nuget add source` command adds a new package source to your NuGet configuration files.

WARNING

When adding multiple package sources, be careful not to introduce a [dependency confusion vulnerability](#).

Arguments

- `PACKAGE_SOURCE_PATH`

Path to the package source.

Options

- `--configfile <FILE>`

The NuGet configuration file (*nuget.config*) to use. If specified, only the settings from this file will be used. If not specified, the hierarchy of configuration files from the current directory will be used. For more information, see [Common NuGet Configurations](#).

- `-n|--name <SOURCE_NAME>`

Name of the source.

- `-p|--password <PASSWORD>`

Password to be used when connecting to an authenticated source.

- `--store-password-in-clear-text`

Enables storing portable package source credentials by disabling password encryption.

- `-u|--username <USER>`

Username to be used when connecting to an authenticated source.

- `--valid-authentication-types <TYPES>`

Comma-separated list of valid authentication types for this source. Set this to `basic` if the server advertises NTLM or Negotiate and your credentials must be sent using the Basic mechanism, for instance when using a PAT with on-premises Azure DevOps Server. Other valid values include `negotiate`, `kerberos`, `ntlm`, and `digest`, but these values are unlikely to be useful.

Examples

- Add `nuget.org` as a source:

```
dotnet nuget add source https://api.nuget.org/v3/index.json -n nuget.org
```

- Add `c:\packages` as a local source:

```
dotnet nuget add source c:\packages
```

- Add a source that needs authentication:

```
dotnet nuget add source https://someServer/myTeam -n myTeam -u myUsername -p myPassword --store-password-in-clear-text
```

- Add a source that needs authentication (then go install credential provider):

```
dotnet nuget add source https://azureartifacts.microsoft.com/myTeam -n myTeam
```

See also

- [Package source sections in NuGet.config files](#)
- [sources command \(nuget.exe\)](#)

dotnet nuget disable source

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 3.1.200 SDK and later versions

Name

`dotnet nuget disable source` - Disable a NuGet source.

Synopsis

```
dotnet nuget disable source <NAME> [--configfile <FILE>]
```

```
dotnet nuget disable source -h|--help
```

Description

The `dotnet nuget disable source` command disables an existing source in your NuGet configuration files.

Arguments

- `<NAME>`

Name of the source.

Options

- `--configfile <FILE>`

The NuGet configuration file (*nuget.config*) to use. If specified, only the settings from this file will be used.

If not specified, the hierarchy of configuration files from the current directory will be used. For more information, see [Common NuGet Configurations](#).

Examples

- Disable a source with name of `mySource`:

```
dotnet nuget disable source mySource
```

See also

- [Package source sections in NuGet.config files](#)
- [sources command \(nuget.exe\)](#)

dotnet nuget enable source

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 3.1.200 SDK and later versions

Name

`dotnet nuget enable source` - Enable a NuGet source.

Synopsis

```
dotnet nuget enable source <NAME> [--configfile <FILE>]
```

```
dotnet nuget enable source -h|--help
```

Description

The `dotnet nuget enable source` command enables an existing source in your NuGet configuration files.

Arguments

- `<NAME>`

Name of the source.

Options

- `--configfile <FILE>`

The NuGet configuration file (*nuget.config*) to use. If specified, only the settings from this file will be used.

If not specified, the hierarchy of configuration files from the current directory will be used. For more information, see [Common NuGet Configurations](#).

Examples

- Enable a source with name of `mySource` :

```
dotnet nuget enable source mySource
```

See also

- [Package source sections in NuGet.config files](#)
- [sources command \(nuget.exe\)](#)

dotnet nuget list source

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 3.1.200 SDK and later versions

Name

`dotnet nuget list source` - Lists all configured NuGet sources.

Synopsis

```
dotnet nuget list source [--format [Detailed|Short]] [--configfile <FILE>]
```

```
dotnet nuget list source -h|--help
```

Description

The `dotnet nuget list source` command lists all existing sources from your NuGet configuration files.

Options

- `--configfile <FILE>`

The NuGet configuration file (*nuget.config*) to use. If specified, only the settings from this file will be used. If not specified, the hierarchy of configuration files from the current directory will be used. For more information, see [Common NuGet Configurations](#).

- `--format [Detailed|Short]`

The format of the list command output: `Detailed` (the default) and `Short`.

Examples

- List configured sources from the current directory:

```
dotnet nuget list source
```

See also

- [Package source sections in NuGet.config files](#)
- [sources command \(nuget.exe\)](#)

dotnet nuget remove source

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 3.1.200 SDK and later versions

Name

`dotnet nuget remove source` - Remove a NuGet source.

Synopsis

```
dotnet nuget remove source <NAME> [--configfile <FILE>]
```

```
dotnet nuget remove source -h|--help
```

Description

The `dotnet nuget remove source` command removes an existing source from your NuGet configuration files.

Arguments

- `<NAME>`

Name of the source.

Options

- `--configfile <FILE>`

The NuGet configuration file (*nuget.config*) to use. If specified, only the settings from this file will be used.

If not specified, the hierarchy of configuration files from the current directory will be used. For more information, see [Common NuGet Configurations](#).

Examples

- Remove a source with name of `mySource` :

```
dotnet nuget remove source mySource
```

See also

- [Package source sections in NuGet.config files](#)
- [sources command \(nuget.exe\)](#)

dotnet nuget update source

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 3.1.200 SDK and later versions

Name

`dotnet nuget update source` - Update a NuGet source.

Synopsis

```
dotnet nuget update source <NAME> [--source <SOURCE>] [--username <USER>]  
    [--password <PASSWORD>] [--store-password-in-clear-text]  
    [--valid-authentication-types <TYPES>] [--configfile <FILE>]
```

```
dotnet nuget update source -h|--help
```

Description

The `dotnet nuget update source` command updates an existing source in your NuGet configuration files.

Arguments

- `<NAME>`

Name of the source.

Options

- `--configfile <FILE>`

The NuGet configuration file (*nuget.config*) to use. If specified, only the settings from this file will be used. If not specified, the hierarchy of configuration files from the current directory will be used. For more information, see [Common NuGet Configurations](#).

- `-p|--password <PASSWORD>`

Password to be used when connecting to an authenticated source.

- `-s|--source <SOURCE>`

Path to the package source.

- `--store-password-in-clear-text`

Enables storing portable package source credentials by disabling password encryption.

- `-u|--username <USER>`

Username to be used when connecting to an authenticated source.

- `--valid-authentication-types <TYPES>`

Comma-separated list of valid authentication types for this source. Set this to `basic` if the server

advertises NTLM or Negotiate and your credentials must be sent using the Basic mechanism, for instance when using a PAT with on-premises Azure DevOps Server. Other valid values include `negotiate`, `kerberos`, `ntlm`, and `digest`, but these values are unlikely to be useful.

Examples

- Update a source with name of `mySource`:

```
dotnet nuget update source mySource --source c:\packages
```

See also

- [Package source sections in NuGet.config files](#)
- [sources command \(nuget.exe\)](#)

dotnet nuget verify

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET 5.0.100-rc.2.x SDK and later versions

Name

`dotnet nuget verify` - Verifies a signed NuGet package.

Synopsis

```
dotnet nuget verify [<package-path(s)>]
  [--all]
  [--certificate-fingerprint <FINGERPRINT>]
  [-v|--verbosity <LEVEL>]
  [--configfile <FILE>]

dotnet nuget verify -h|--help
```

Description

The `dotnet nuget verify` command verifies a signed NuGet package.

Arguments

- `package-path(s)`

Specifies the file path to the package(s) to be verified. Multiple position arguments can be passed in to verify multiple packages.

Options

- `--all`

Specifies that all verifications possible should be performed on the package(s). By default, only `signatures` are verified.

NOTE

This command currently supports only `signature` verification.

- `--certificate-fingerprint <FINGERPRINT>`

Verify that the signer certificate matches with one of the specified `SHA256` fingerprints. This option can be supplied multiple times to provide multiple fingerprints.

- `-v|--verbosity <LEVEL>`

Sets the verbosity level of the command. Allowed values are `q[uiet]`, `m[inimal]`, `n[ormal]`, `d[etailed]`, and `diag[nostic]`. The default is `minimal`. For more information, see [LoggerVerbosity](#).

The following table shows what is displayed for each verbosity level.

	Q[UIET]	M[INIMAL]	N[ORMAL]	D[ETAILED]	DIAG[NOSTIC]
Certificate chain Information	✗	✗	✗	✓	✓
Path to package being verified	✗	✗	✓	✓	✓
Hashing algorithm used for signature	✗	✗	✓	✓	✓
Author/Repository Certificate -> SHA1 hash	✗	✗	✓	✓	✓
Author/Repository Certificate -> Issued By	✗	✗	✓	✓	✓
Timestamp Certificate -> Issued By	✗	✗	✓	✓	✓
Timestamp Certificate -> SHA-256 hash	✗	✗	✓	✓	✓
Timestamp Certificate -> Validity period	✗	✗	✓	✓	✓
Timestamp Certificate -> SHA1 hash	✗	✗	✓	✓	✓
Timestamp Certificate -> Subject name	✗	✗	✓	✓	✓
Author/Repository Certificate -> Subject name	✗	✓	✓	✓	✓
Author/Repository Certificate -> SHA-256 hash	✗	✓	✓	✓	✓
Author/Repository Certificate -> Validity period	✗	✓	✓	✓	✓
Author/Repository Certificate -> Service index URL (If applicable)	✗	✓	✓	✓	✓

Q[UIET]	M[INIMAL]	N[ORMAL]	D[ETAILED]	DIAG[NOSTIC]
---------	-----------	----------	------------	--------------

Package name being verified	✗	✓	✓	✓	✓
Type of signature (author or repository)	✗	✓	✓	✓	✓

✗ indicates details that are **not** displayed. ✓ indicates details that are displayed.

- `--configfile <FILE>`

The NuGet configuration file (*nuget.config*) to use. If specified, only the settings from this file will be used. If not specified, the hierarchy of configuration files from the current directory will be used. For more information, see [Common NuGet Configurations](#).

- `-?|-h|--help`

Prints out a description of how to use the command.

Examples

- Verify *foo.nupkg*:

```
dotnet nuget verify foo.nupkg
```

- Verify multiple NuGet packages - *foo.nupkg* and *all .nupkg files in the directory specified*:

```
dotnet nuget verify foo.nupkg c:\mydir\*.nupkg
```

- Verify *foo.nupkg* signature matches with the specified certificate fingerprint:

```
dotnet nuget verify foo.nupkg --certificate-fingerprint  
CE40881FF5F0AD3E58965DA20A9F571EF1651A56933748E1BF1C99E537C4E039
```

- Verify *foo.nupkg* signature matches with one of the specified certificate fingerprints:

```
dotnet nuget verify foo.nupkg --certificate-fingerprint  
CE40881FF5F0AD3E58965DA20A9F571EF1651A56933748E1BF1C99E537C4E039 --certificate-fingerprint  
EC10992GG5F0AD3E58965DA20A9F571EF1651A56933748E1BF1C99E537C4E027
```

- Verify the signature of *foo.nupkg* by using settings (`packagesources` and `trustedSigners`) only from the specified *nuget.config* file:

```
dotnet nuget verify foo.nupkg --configfile ..\Settings\NuGet.config
```

dotnet nuget trust

9/20/2022 • 6 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET 5.0.300 SDK and later versions

Name

`dotnet nuget trust` - Gets or sets trusted signers to the NuGet configuration.

Synopsis

```
dotnet nuget trust [command] [Options]
```

```
dotnet nuget trust -h|--help
```

Description

The `dotnet nuget trust` command manages the trusted signers. By default, NuGet accepts all authors and repositories. These commands allow you to specify only a specific subset of signers whose signatures will be accepted, while rejecting all others. For more information, see [Common NuGet configurations](#). For details on what the `nuget.config` schema looks like, refer to the [NuGet config file reference](#).

Options

- `-?|-h|--help`

Prints out a description of how to use the command.

Commands

If no command is specified, the command will default to `list`.

`list`

Lists all the trusted signers in the configuration. This option will include all the certificates (with fingerprint and fingerprint algorithm) each signer has. If a certificate has a preceding [U], it means that certificate entry has `allowUntrustedRoot` set as true.

Synopsis:

```
dotnet nuget trust list [--configfile <PATH>] [-h|--help] [-v, --verbosity <LEVEL>]
```

Options:

- `--configfile <FILE>`

The NuGet configuration file (`nuget.config`) to use. If specified, only the settings from this file will be used. If not specified, the hierarchy of configuration files from the current directory will be used. For more information, see [Common NuGet Configurations](#).

- `-?|-h|--help`

Prints out a description of how to use the command.

- `-v|--verbosity <LEVEL>`

Sets the verbosity level of the command. Allowed values are `q[uiet]`, `m[inimal]`, `n[ormal]`, `d[etailed]`, and `diag[nostic]`. The default is `minimal`. For more information, see [LoggerVerbosity](#).

sync

Deletes the current list of certificates and replaces them with an up-to-date list from the repository.

Synopsis

```
dotnet nuget trust sync <NAME> [--configfile <PATH>] [-h|--help] [-v, --verbosity <LEVEL>]
```

Arguments

- `NAME`

The name of the existing trusted signer to sync.

Options:

- `--configfile <FILE>`

The NuGet configuration file (*nuget.config*) to use. If specified, only the settings from this file will be used. If not specified, the hierarchy of configuration files from the current directory will be used. For more information, see [Common NuGet Configurations](#).

- `-?|-h|--help`

Prints out a description of how to use the command.

- `-v|--verbosity <LEVEL>`

Sets the verbosity level of the command. Allowed values are `q[uiet]`, `m[inimal]`, `n[ormal]`, `d[etailed]`, and `diag[nostic]`. The default is `minimal`. For more information, see [LoggerVerbosity](#).

remove

Removes any trusted signers that match the given name.

Synopsis

```
dotnet nuget trust remove <NAME> [--configfile <PATH>] [-h|--help] [-v, --verbosity <LEVEL>]
```

Arguments

- `NAME`

The name of the existing trusted signer to remove.

Options:

- `--configfile <FILE>`

The NuGet configuration file (*nuget.config*) to use. If specified, only the settings from this file will be used. If not specified, the hierarchy of configuration files from the current directory will be used. For more information, see [Common NuGet Configurations](#).

- `-?|-h|--help`

Prints out a description of how to use the command.

- `-v|--verbosity <LEVEL>`

Sets the verbosity level of the command. Allowed values are `q[uiet]`, `m[inimal]`, `n[ormal]`, `d[etailed]`,

and `diag[nostic]`. The default is `minimal`. For more information, see [LoggerVerbosity](#).

author

Adds a trusted signer with the given name, based on the author signature of the package.

Synopsis

```
dotnet nuget trust author <NAME> <PACKAGE> [--allow-untrusted-root] [--configfile <PATH>] [-h|--help] [-v, --verbosity <LEVEL>]
```

Arguments

- `NAME`

The name of the trusted signer to add. If `NAME` already exists in the configuration, the signature is appended.

- `PACKAGE`

The given `PACKAGE` should be a local path to the signed `.nupkg` file.

Options:

- `--allow-untrusted-root`

Specifies if the certificate for the trusted signer should be allowed to chain to an untrusted root. This is not recommended.

- `--configfile <FILE>`

The NuGet configuration file (`nuget.config`) to use. If specified, only the settings from this file will be used. If not specified, the hierarchy of configuration files from the current directory will be used. For more information, see [Common NuGet Configurations](#).

- `-?|-h|--help`

Prints out a description of how to use the command.

- `-v|--verbosity <LEVEL>`

Sets the verbosity level of the command. Allowed values are `q[uiet]`, `m[inimal]`, `n[ormal]`, `d[etailed]`, and `diag[nostic]`. The default is `minimal`. For more information, see [LoggerVerbosity](#).

repository

Adds a trusted signer with the given name, based on the repository signature or countersignature of a signed package.

Synopsis

```
dotnet nuget trust repository <NAME> <PACKAGE> [--allow-untrusted-root] [--configfile <PATH>] [-h|--help] [-owners <LIST>] [-v, --verbosity <LEVEL>]
```

Arguments

- `NAME`

The name of the trusted signer to add. If `NAME` already exists in the configuration, the signature is appended.

- `PACKAGE`

The given `PACKAGE` should be a local path to the signed `.nupkg` file.

Options:

- `--allow-untrusted-root`

Specifies if the certificate for the trusted signer should be allowed to chain to an untrusted root. This is not recommended.

- `--configfile <FILE>`

The NuGet configuration file (*nuget.config*) to use. If specified, only the settings from this file will be used. If not specified, the hierarchy of configuration files from the current directory will be used. For more information, see [Common NuGet Configurations](#).

- `-?|-h|--help`

Prints out a description of how to use the command.

- `--owners <LIST>`

Semicolon-separated list of trusted owners to further restrict the trust of a repository.

- `-v|--verbosity <LEVEL>`

Sets the verbosity level of the command. Allowed values are `q[uiet]`, `m[inimal]`, `n[ormal]`, `d[etailed]`, and `diag[nostic]`. The default is `minimal`. For more information, see [LoggerVerbosity](#).

`certificate`

Adds a trusted signer with the given name, based on a certificate fingerprint.

Synopsis

```
dotnet nuget trust certificate <NAME> <FINGERPRINT> [--algorithm <ALGORITHM>] [--allow-untrusted-root] [--configfile <PATH>] [-h|--help] [-v, --verbosity <LEVEL>]
```

Arguments

- `NAME`

The name of the trusted signer to add. If a trusted signer with the given name already exists, the certificate item is added to that signer. Otherwise a trusted author is created with a certificate item from the given certificate information.

- `FINGERPRINT`

The fingerprint of the certificate.

Options:

- `--algorithm <ALGORITHM>`

Specifies the hash algorithm used to calculate the certificate fingerprint. Defaults to SHA256. Values supported are SHA256, SHA384 and SHA512.

- `--allow-untrusted-root`

Specifies if the certificate for the trusted signer should be allowed to chain to an untrusted root. This is not recommended.

- `--configfile <FILE>`

The NuGet configuration file (*nuget.config*) to use. If specified, only the settings from this file will be used. If not specified, the hierarchy of configuration files from the current directory will be used. For more information, see [Common NuGet Configurations](#).

- `-?|-h|--help`

Prints out a description of how to use the command.

- `-v|--verbosity <LEVEL>`

Sets the verbosity level of the command. Allowed values are `q[uiet]`, `m[inimal]`, `n[ormal]`, `d[etailed]`, and `diag[nostic]`. The default is `minimal`. For more information, see [LoggerVerbosity](#).

source

Adds a trusted signer based on a given package source.

Synopsis

```
dotnet nuget trust source <NAME> [--configfile <PATH>] [-h|--help] [--owners <LIST>] [--source-url] [-v, --verbosity <LEVEL>]
```

Arguments

- `<NAME>`

The name of the trusted signer to add. If only `<NAME>` is provided without `--<source-url>`, the package source from your NuGet configuration files with the same name is added to the trusted list. If `<NAME>` already exists in the configuration, the package source is appended to it.

Options:

- `--configfile <FILE>`

The NuGet configuration file (*nuget.config*) to use. If specified, only the settings from this file will be used. If not specified, the hierarchy of configuration files from the current directory will be used. For more information, see [Common NuGet Configurations](#).

- `-?|-h|--help`

Prints out a description of how to use the command.

- `--owners <LIST>`

Semicolon-separated list of trusted owners to further restrict the trust of a repository.

- `--source-url`

If a `<source-url>` is provided, it must be a v3 package source URL (like `https://api.nuget.org/v3/index.json`). Other package source types are not supported.

- `-v|--verbosity <LEVEL>`

Sets the verbosity level of the command. Allowed values are `q[uiet]`, `m[inimal]`, `n[ormal]`, `d[etailed]`, and `diag[nostic]`. The default is `minimal`. For more information, see [LoggerVerbosity](#).

Examples

- List trusted signers:

```
dotnet nuget trust list
```

- Trust source *NuGet* in specified *nuget.config* file:

```
dotnet nuget trust source NuGet --configfile ..\nuget.config
```

- Trust an author from signed nupkg package file *foo.nupkg*:

```
dotnet nuget trust author PackageAuthor .\foo.nupkg
```

- Trust a repository from signed nupkg package file *foo.nupkg*:

```
dotnet nuget trust repository PackageRepository .\foo.nupkg
```

- Trust a package signing certificate using its SHA256 fingerprint:

```
dotnet nuget trust certificate MyCert  
F99EC8CDCE5642B380296A19E22FA8EB3AEF1C70079541A2B3D6E4A93F5E1AFD --algorithm SHA256
```

- Trust owners *NuGet* and *Microsoft* from the repository <https://api.nuget.org/v3/index.json>:

```
dotnet nuget trust source NuGetTrust --source-url https://api.nuget.org/v3/index.json --owners  
"Nuget;Microsoft"
```

- Remove trusted signer named *NuGet* from specified *nuget.config* file:

```
dotnet nuget trust remove NuGet --configfile ..\nuget.config
```

dotnet nuget sign

9/20/2022 • 3 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET 6 SDK and later versions

Name

`dotnet nuget sign` - Signs all the NuGet packages matching the first argument with a certificate.

Synopsis

```
dotnet nuget sign [<package-path(s)>]
  [--certificate-path <PATH>]
  [--certificate-store-name <STORENAME>]
  [--certificate-store-location <STORELOCATION>]
  [--certificate-subject-name <SUBJECTNAME>]
  [--certificate-fingerprint <FINGERPRINT>]
  [--certificate-password <PASSWORD>]
  [--hash-algorithm <HASHALGORITHM>]
  [-o|--output <OUTPUT DIRECTORY>]
  [--overwrite]
  [--timestamp-hash-algorithm <HASHALGORITHM>]
  [--timestamper <TIMESTAMPINGSERVER>]
  [-v|--verbosity <LEVEL>]
```

```
dotnet nuget sign -h|--help
```

Description

The `dotnet nuget sign` command signs all the packages matching the first argument with a certificate. The certificate with the private key can be obtained from a file or from a certificate installed in a certificate store by providing a subject name or a SHA-1 fingerprint.

Arguments

- `package-path(s)`

Specifies the file path to the package(s) to be signed. Multiple arguments can be passed in to sign multiple packages.

Options

- `--certificate-path <PATH>`

Specifies the file path to the certificate to be used in signing the package.

NOTE

This option currently supports only `PKCS12 (PFX)` files that contain the certificate's private key.

- `--certificate-store-name <STORENAME>`

Specifies the name of the X.509 certificate store to use to search for the certificate. Defaults to "My", the X.509 certificate store for personal certificates. This option should be used when specifying the certificate via `--certificate-subject-name` or `--certificate-fingerprint` options.

- `--certificate-store-location <STORELOCATION>`

Specifies the name of the X.509 certificate store use to search for the certificate. Defaults to "CurrentUser", the X.509 certificate store used by the current user. This option should be used when specifying the certificate via `--certificate-subject-name` or `--certificate-fingerprint` options.

- `--certificate-subject-name <SUBJECTNAME>`

Specifies the subject name of the certificate used to search a local certificate store for the certificate. The search is a case-insensitive string comparison using the supplied value, which will find all certificates with the subject name containing that string, regardless of other subject values. The certificate store can be specified by `--certificate-store-name` and `--certificate-store-location` options.

NOTE

This option currently supports only a single matching certificate in the result. If there are multiple matching certificates in the result, or no matching certificate in the result, the sign command will fail.

- `--certificate-fingerprint <FINGERPRINT>`

SHA-1 fingerprint of the certificate used to search a local certificate store for the certificate.

- `--certificate-password <PASSWORD>`

Specifies the certificate password, if needed. If a certificate is password protected but no password is provided, the sign command will fail.

NOTE

The `sign` command only supports non-interactive mode. There won't be any prompt for a password at run time.

- `--hash-algorithm <HASHALGORITHM>`

Hash algorithm to be used to sign the package. Defaults to SHA256. Possible values are SHA256, SHA384, and SHA512.

- `-o|--output`

Specifies the directory where the signed package should be saved. If this option is not specified, by default the original package is overwritten by the signed package.

- `--overwrite`

Indicate that the current signature should be overwritten. By default the command will fail if the package already has a signature.

- `--timestamp-hash-algorithm <HASHALGORITHM>`

Hash algorithm to be used by the RFC 3161 timestamp server. Defaults to SHA256.

- `--timestamper <TIMESTAMPINGSERVER>`

URL to an RFC 3161 timestamping server.

- `-v|--verbosity <LEVEL>`

Sets the verbosity level of the command. Allowed values are `q[uiet]`, `m[inimal]`, `n[ormal]`, `d[etailed]`, and `diag[nostic]`. The default is `minimal`. For more information, see [LoggerVerbosity](#).

- `-?|-h|--help`

Prints out a description of how to use the command.

Examples

- Sign `foo.nupkg` with certificate `cert.pfx` (not password protected):

```
dotnet nuget sign foo.nupkg --certificate-path cert.pfx
```

- Sign `foo.nupkg` with certificate `cert.pfx` (password protected):

```
dotnet nuget sign foo.nupkg --certificate-path cert.pfx --certificate-password password
```

- Sign `foo.nupkg` with certificate (password protected) matches with the specified SHA-1 fingerprint in the default certificate store (`CurrentUser\My`):

```
dotnet nuget sign foo.nupkg --certificate-fingerprint 89967D1DD995010B6C66AE24FF8E66885E6E03A8 --certificate-password password
```

- Sign `foo.nupkg` with certificate (password protected) matches with the specified subject name "Test certificate for testing signing" in the default certificate store (`CurrentUser\My`):

```
dotnet nuget sign foo.nupkg --certificate-subject-name "Test certificate for testing signing" --certificate-password password
```

- Sign `foo.nupkg` with certificate (password protected) matches with the specified SHA-1 fingerprint in the certificate store `CurrentUser\Root`:

```
dotnet nuget sign foo.nupkg --certificate-fingerprint 89967D1DD995010B6C66AE24FF8E66885E6E03A8 --certificate-password password --certificate-store-location CurrentUser --certificate-store-name Root
```

- Sign multiple NuGet packages - `foo.nupkg` and *all .nupkg files in the directory specified* with certificate `cert.pfx` (not password protected):

```
dotnet nuget sign foo.nupkg c:\mydir\*.nupkg --certificate-path cert.pfx
```

- Sign `foo.nupkg` with certificate `cert.pfx` (password protected), and timestamp with `http://timestamp.test`:

```
dotnet nuget sign foo.nupkg --certificate-path cert.pfx --certificate-password password --timestamper http://timestamp.test
```

- Sign `foo.nupkg` with certificate `cert.pfx` (not password protected) and save the signed package under specified directory:

```
dotnet nuget sign foo.nupkg --certificate-path cert.pfx --output c:\signed\
```

- Sign *foo.nupkg* with certificate *cert.pfx* (not password protected) and overwrite the current signature if the package is already signed:

```
dotnet nuget sign foo.nupkg --certificate-path cert.pfx --overwrite
```

dotnet pack

9/20/2022 • 5 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 3.1 SDK and later versions

Name

`dotnet pack` - Packs the code into a NuGet package.

Synopsis

```
dotnet pack [<PROJECT>|<SOLUTION>] [-c|--configuration <CONFIGURATION>]
  [--force] [--include-source] [--include-symbols] [--interactive]
  [--no-build] [--no-dependencies] [--no-restore] [--nologo]
  [-o|--output <OUTPUT_DIRECTORY>] [--runtime <RUNTIME_IDENTIFIER>]
  [-s|--serviceable] [-v|--verbosity <LEVEL>]
  [--version-suffix <VERSION_SUFFIX>]

dotnet pack -h|--help
```

Description

The `dotnet pack` command builds the project and creates NuGet packages. The result of this command is a NuGet package (that is, a `.nupkg` file).

If you want to generate a package that contains the debug symbols, you have two options available:

- `--include-symbols` - it creates the symbols package.
- `--include-source` - it creates the symbols package with a `src` folder inside containing the source files.

NuGet dependencies of the packed project are added to the `.nuspec` file, so they're properly resolved when the package is installed. If the packed project has references to other projects, the other projects are not included in the package. Currently, you must have a package per project if you have project-to-project dependencies.

By default, `dotnet pack` builds the project first. If you wish to avoid this behavior, pass the `--no-build` option.

This option is often useful in Continuous Integration (CI) build scenarios where you know the code was previously built.

NOTE

In some cases, the implicit build cannot be performed. This can occur when `GeneratePackageOnBuild` is set, to avoid a cyclic dependency between build and pack targets. The build can also fail if there is a locked file or other issue.

You can provide MSBuild properties to the `dotnet pack` command for the packing process. For more information, see [NuGet pack target properties](#) and the [MSBuild Command-Line Reference](#). The [Examples](#) section shows how to use the MSBuild `-p` switch for a couple of different scenarios.

NOTE

Web projects aren't packable.

Implicit restore

You don't have to run `dotnet restore` because it's run implicitly by all commands that require a restore to occur, such as `dotnet new`, `dotnet build`, `dotnet run`, `dotnet test`, `dotnet publish`, and `dotnet pack`. To disable implicit restore, use the `--no-restore` option.

The `dotnet restore` command is still useful in certain scenarios where explicitly restoring makes sense, such as [continuous integration builds in Azure DevOps Services](#) or in build systems that need to explicitly control when the restore occurs.

For information about how to manage NuGet feeds, see the [dotnet restore documentation](#).

This command supports the `dotnet restore` options when passed in the long form (for example, `--source`). Short form options, such as `-s`, are not supported.

Workload manifest downloads

When you run this command, it initiates an asynchronous background download of advertising manifests for workloads. If the download is still running when this command finishes, the download is stopped. For more information, see [Advertising manifests](#).

Arguments

`PROJECT | SOLUTION`

The project or solution to pack. It's either a path to a csproj, vbproj, or fsproj file, or to a solution file or directory. If not specified, the command searches the current directory for a project or solution file.

Options

- `-c|--configuration <CONFIGURATION>`

Defines the build configuration. The default for most projects is `Debug`, but you can override the build configuration settings in your project.

- `--force`

Forces all dependencies to be resolved even if the last restore was successful. Specifying this flag is the same as deleting the `project.assets.json` file.

- `-?|-h|--help`

Prints out a description of how to use the command.

- `--include-source`

Includes the debug symbols NuGet packages in addition to the regular NuGet packages in the output directory. The sources files are included in the `src` folder within the symbols package.

- `--include-symbols`

Includes the debug symbols NuGet packages in addition to the regular NuGet packages in the output directory.

- `--interactive`

Allows the command to stop and wait for user input or action. For example, to complete authentication. Available since .NET Core 3.0 SDK.

- `--no-build`

Doesn't build the project before packing. It also implicitly sets the `--no-restore` flag.

- `--no-dependencies`

Ignores project-to-project references and only restores the root project.

- `--no-restore`

Doesn't execute an implicit restore when running the command.

- `--nologo`

Doesn't display the startup banner or the copyright message.

- `-o|--output <OUTPUT_DIRECTORY>`

Places the built packages in the directory specified.

- `--runtime <RUNTIME_IDENTIFIER>`

Specifies the target runtime to restore packages for. For a list of Runtime Identifiers (RIDs), see the [RID catalog](#).

- `-s|--serviceable`

Sets the serviceable flag in the package. For more information, see [.NET Blog: .NET Framework 4.5.1 Supports Microsoft Security Updates for .NET NuGet Libraries](#).

- `-v|--verbosity <LEVEL>`

Sets the verbosity level of the command. Allowed values are `q[uiet]`, `m[inimal]`, `n[ormal]`, `d[etailed]`, and `diag[nostic]`. For more information, see [LoggerVerbosity](#).

- `--version-suffix <VERSION_SUFFIX>`

Defines the value for the `VersionSuffix` MSBuild property. The effect of this property on the package version depends on the values of the `Version` and `VersionPrefix` properties, as shown in the following table:

PROPERTIES WITH VALUES	PACKAGE VERSION
None	<code>1.0.0</code>
<code>Version</code>	<code>\$(Version)</code>
<code>VersionPrefix</code> only	<code>\$(VersionPrefix)</code>
<code>VersionSuffix</code> only	<code>1.0.0-\$(VersionSuffix)</code>
<code>VersionPrefix</code> and <code>VersionSuffix</code>	<code>\$(VersionPrefix)-\$(VersionSuffix)</code>

If you want to use `--version-suffix`, specify `VersionPrefix` and not `Version` in the project file. For example, if `versionPrefix` is `0.1.2` and you pass `--version-suffix rc.1` to `dotnet pack`, the package version will be `0.1.2-rc.1`.

If `Version` has a value and you pass `--version-suffix` to `dotnet pack`, the value specified for `--version-suffix` is ignored.

Examples

- Pack the project in the current directory:

```
dotnet pack
```

- Pack the `app1` project:

```
dotnet pack ~/projects/app1/project.csproj
```

- Pack the project in the current directory and place the resulting packages into the `nupkgs` folder:

```
dotnet pack --output nupkgs
```

- Pack the project in the current directory into the `nupkgs` folder and skip the build step:

```
dotnet pack --no-build --output nupkgs
```

- With the project's version suffix configured as `<VersionSuffix>$(VersionSuffix)</VersionSuffix>` in the `.csproj` file, pack the current project and update the resulting package version with the given suffix:

```
dotnet pack --version-suffix "ci-1234"
```

- Set the package version to `2.1.0` with the `PackageVersion` MSBuild property:

```
dotnet pack -p:PackageVersion=2.1.0
```

- Pack the project for a specific [target framework](#):

```
dotnet pack -p:TargetFrameworks=net45
```

- Pack the project and use a specific runtime (Windows 10) for the restore operation:

```
dotnet pack --runtime win10-x64
```

- Pack the project using a `.nuspec` file:

```
dotnet pack ~/projects/app1/project.csproj -p:NuspecFile=~/projects/app1/project.nuspec -  
p:NuspecbasePath=~/projects/app1/nuget
```

For information about how to use `NuspecFile`, `NuspecbasePath`, and `NuspecProperties`, see the following resources:

- [Packing using a .nuspec](#)
- [Advanced extension points to create customized package](#)
- [Global properties](#)

dotnet publish

9/20/2022 • 9 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 3.1 SDK and later versions

Name

`dotnet publish` - Publishes the application and its dependencies to a folder for deployment to a hosting system.

Synopsis

```
dotnet publish [<PROJECT>|<SOLUTION>] [-a|--arch <ARCHITECTURE>]
  [-c|--configuration <CONFIGURATION>]
  [-f|--framework <FRAMEWORK>] [--force] [--interactive]
  [--manifest <PATH_TO_MANIFEST_FILE>] [--no-build] [--no-dependencies]
  [--no-restore] [--nologo] [-o|--output <OUTPUT_DIRECTORY>]
  [--os <OS>] [-r|--runtime <RUNTIME_IDENTIFIER>]
  [--sc|--self-contained [true|false]] [--no-self-contained]
  [-s|--source <SOURCE>] [-v|--verbosity <LEVEL>]
  [--version-suffix <VERSION_SUFFIX>]

dotnet publish -h|--help
```

Description

`dotnet publish` compiles the application, reads through its dependencies specified in the project file, and publishes the resulting set of files to a directory. The output includes the following assets:

- Intermediate Language (IL) code in an assembly with a *dll* extension.
- A *.deps.json* file that includes all of the dependencies of the project.
- A *.runtimeconfig.json* file that specifies the shared runtime that the application expects, as well as other configuration options for the runtime (for example, garbage collection type).
- The application's dependencies, which are copied from the NuGet cache into the output folder.

The `dotnet publish` command's output is ready for deployment to a hosting system (for example, a server, PC, Mac, laptop) for execution. It's the only officially supported way to prepare the application for deployment. Depending on the type of deployment that the project specifies, the hosting system may or may not have the .NET shared runtime installed on it. For more information, see [Publish .NET apps with the .NET CLI](#).

Implicit restore

You don't have to run `dotnet restore` because it's run implicitly by all commands that require a restore to occur, such as `dotnet new`, `dotnet build`, `dotnet run`, `dotnet test`, `dotnet publish`, and `dotnet pack`. To disable implicit restore, use the `--no-restore` option.

The `dotnet restore` command is still useful in certain scenarios where explicitly restoring makes sense, such as [continuous integration builds in Azure DevOps Services](#) or in build systems that need to explicitly control when the restore occurs.

For information about how to manage NuGet feeds, see the [dotnet restore documentation](#).

MSBuild

The `dotnet publish` command calls MSBuild, which invokes the `Publish` target. If the `IsPublishable` property is set to `false` for a particular project, the `Publish` target can't be invoked, and the `dotnet publish` command only runs the implicit `dotnet restore` on the project.

Any parameters passed to `dotnet publish` are passed to MSBuild. The `-c` and `-o` parameters map to MSBuild's `Configuration` and `PublishDir` properties, respectively.

The `dotnet publish` command accepts MSBuild options, such as `-p` for setting properties and `-l` to define a logger. For example, you can set an MSBuild property by using the format: `-p:<NAME>=<VALUE>`.

You can also set publish-related properties by referring to a `.pubxml` file. For example:

```
dotnet publish -p:PublishProfile=FolderProfile
```

The preceding example uses the `FolderProfile.pubxml` file that is found in the `<project_folder>/Properties/PublishProfiles` folder. If you specify a path and file extension when setting the `PublishProfile` property, they are ignored. MSBuild by default looks in the `Properties/PublishProfiles` folder and assumes the `pubxml` file extension. To specify the path and filename including extension, set the `PublishProfileFullPath` property instead of the `PublishProfile` property.

The following MSBuild properties change the output of `dotnet publish`.

- `PublishReadyToRun`

Compiles application assemblies as ReadyToRun (R2R) format. R2R is a form of ahead-of-time (AOT) compilation. For more information, see [ReadyToRun images](#).

To see warnings about missing dependencies that could cause runtime failures, use

```
PublishReadyToRunShowWarnings=true
```

We recommend that you specify `PublishReadyToRun` in a publish profile rather than on the command line.

- `PublishSingleFile`

Packages the app into a platform-specific single-file executable. For more information about single-file publishing, see the [single-file bundler design document](#).

We recommend that you specify this option in the project file rather than on the command line.

- `PublishTrimmed`

Trims unused libraries to reduce the deployment size of an app when publishing a self-contained executable. For more information, see [Trim self-contained deployments and executables](#). Available since .NET 6 SDK.

We recommend that you specify this option in the project file rather than on the command line.

For more information, see the following resources:

- [MSBuild command-line reference](#)
- [Visual Studio publish profiles \(.pubxml\) for ASP.NET Core app deployment](#)
- [dotnet msbuild](#)

Workload manifest downloads

When you run this command, it initiates an asynchronous background download of advertising manifests for workloads. If the download is still running when this command finishes, the download is stopped. For more information, see [Advertising manifests](#).

Arguments

- `PROJECT|SOLUTION`

The project or solution to publish.

- `PROJECT` is the path and filename of a C#, F#, or Visual Basic project file, or the path to a directory that contains a C#, F#, or Visual Basic project file. If the directory is not specified, it defaults to the current directory.
- `SOLUTION` is the path and filename of a solution file (`.sln` extension), or the path to a directory that contains a solution file. If the directory is not specified, it defaults to the current directory.

Options

- `-a|--arch <ARCHITECTURE>`

Specifies the target architecture. This is a shorthand syntax for setting the [Runtime Identifier \(RID\)](#), where the provided value is combined with the default RID. For example, on a `win-x64` machine, specifying `--arch x86` sets the RID to `win-x86`. If you use this option, don't use the `-r|--runtime` option. Available since .NET 6 Preview 7.

- `-c|--configuration <CONFIGURATION>`

Defines the build configuration. The default for most projects is `Debug`, but you can override the build configuration settings in your project.

- `-f|--framework <FRAMEWORK>`

Publishes the application for the specified [target framework](#). You must specify the target framework in the project file.

- `--force`

Forces all dependencies to be resolved even if the last restore was successful. Specifying this flag is the same as deleting the `project.assets.json` file.

- `-?|-h|--help`

Prints out a description of how to use the command.

- `--interactive`

Allows the command to stop and wait for user input or action. For example, to complete authentication. Available since .NET Core 3.0 SDK.

- `--manifest <PATH_TO_MANIFEST_FILE>`

Specifies one or several [target manifests](#) to use to trim the set of packages published with the app. The manifest file is part of the output of the `dotnet store` command. To specify multiple manifests, add a `--manifest` option for each manifest.

- `--no-build`

Doesn't build the project before publishing. It also implicitly sets the `--no-restore` flag.

- `--no-dependencies`

Ignores project-to-project references and only restores the root project.

- `--nologo`

Doesn't display the startup banner or the copyright message.
- `--no-restore`

Doesn't execute an implicit restore when running the command.
- `-o|--output <OUTPUT_DIRECTORY>`

Specifies the path for the output directory.

If not specified, it defaults to `[project_file_folder]/bin/[configuration]/[framework]/publish` for a framework-dependent executable and cross-platform binaries. It defaults to `[project_file_folder]/bin/[configuration]/[framework]/[runtime]/publish` for a self-contained executable.

In a web project, if the output folder is in the project folder, successive `dotnet publish` commands result in nested output folders. For example, if the project folder is `myproject`, and the publish output folder is `myproject/publish`, and you run `dotnet publish` twice, the second run puts content files such as `.config` and `.json` files in `myproject/publish/publish`. To avoid nesting publish folders, specify a publish folder that is not **directly** under the project folder, or exclude the publish folder from the project. To exclude a publish folder named `publishoutput`, add the following element to a `PropertyGroup` element in the `.csproj` file:

```
<DefaultItemExcludes>$(<DefaultItemExcludes>);publishoutput**</DefaultItemExcludes>
```

 - .NET Core 3.x SDK and later

If you specify a relative path when publishing a project, the generated output directory is relative to the current working directory, not to the project file location.

If you specify a relative path when publishing a solution, all output for all projects goes into the specified folder relative to the current working directory. To make publish output go to separate folders for each project, specify a relative path by using the msbuild `PublishDir` property instead of the `--output` option. For example, `dotnet publish -p:PublishDir=.\publish` sends publish output for each project to a `publish` folder under the folder that contains the project file.
 - .NET Core 2.x SDK

If you specify a relative path when publishing a project, the generated output directory is relative to the project file location, not to the current working directory.

If you specify a relative path when publishing a solution, each project's output goes into a separate folder relative to the project file location. If you specify an absolute path when publishing a solution, all publish output for all projects goes into the specified folder.- `--os <OS>`

Specifies the target operating system (OS). This is a shorthand syntax for setting the [Runtime Identifier \(RID\)](#), where the provided value is combined with the default RID. For example, on a `win-x64` machine, specifying `--os linux` sets the RID to `linux-x64`. If you use this option, don't use the `-r|--runtime` option. Available since .NET 6.
- `--sc|--self-contained [true|false]`

Publishes the .NET runtime with your application so the runtime doesn't need to be installed on the target machine. Default is `true` if a runtime identifier is specified and the project is an executable project (not a library project). For more information, see [.NET application publishing](#) and [Publish .NET apps with the](#)

.NET CLI

If this option is used without specifying `true` or `false`, the default is `true`. In that case, don't put the solution or project argument immediately after `--self-contained`, because `true` or `false` is expected in that position.

- `--no-self-contained`

Equivalent to `--self-contained false`.

- `--source <SOURCE>`

The URI of the NuGet package source to use during the restore operation.

- `-r|--runtime <RUNTIME_IDENTIFIER>`

Publishes the application for a given runtime. For a list of Runtime Identifiers (RIDs), see the [RID catalog](#). For more information, see [.NET application publishing](#) and [Publish .NET apps with the .NET CLI](#). If you use this option, use `--self-contained` or `--no-self-contained` also.

- `-v|--verbosity <LEVEL>`

Sets the verbosity level of the command. Allowed values are `q[uiet]`, `m[inimal]`, `n[ormal]`, `d[etailed]`, and `diag[nostic]`. The default is `minimal`. For more information, see [LoggerVerbosity](#).

- `--version-suffix <VERSION_SUFFIX>`

Defines the version suffix to replace the asterisk (`*`) in the version field of the project file.

Examples

- Create a [framework-dependent cross-platform binary](#) for the project in the current directory:

```
dotnet publish
```

Starting with .NET Core 3.0 SDK, this example also creates a [framework-dependent executable](#) for the current platform.

- Create a [self-contained executable](#) for the project in the current directory, for a specific runtime:

```
dotnet publish --runtime osx.10.11-x64
```

The RID must be in the project file.

- Create a [framework-dependent executable](#) for the project in the current directory, for a specific platform:

```
dotnet publish --runtime osx.10.11-x64 --self-contained false
```

The RID must be in the project file. This example applies to .NET Core 3.0 SDK and later versions.

- Publish the project in the current directory, for a specific runtime and target framework:

```
dotnet publish --framework netcoreapp3.1 --runtime osx.10.11-x64
```

- Publish the specified project file:

```
dotnet publish ~/projects/app1/app1.csproj
```

- Publish the current application but don't restore project-to-project (P2P) references, just the root project during the restore operation:

```
dotnet publish --no-dependencies
```

See also

- [.NET application publishing overview](#)
- [Publish .NET apps with the .NET CLI](#)
- [Target frameworks](#)
- [Runtime Identifier \(RID\) catalog](#)
- [Working with macOS Catalina Notarization](#)
- [Directory structure of a published application](#)
- [MSBuild command-line reference](#)
- [Visual Studio publish profiles \(.pubxml\) for ASP.NET Core app deployment](#)
- [dotnet msbuild](#)
- [ILLink.Tasks](#)

dotnet restore

9/20/2022 • 4 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 3.1 SDK and later versions

Name

`dotnet restore` - Restores the dependencies and tools of a project.

Synopsis

```
dotnet restore [<ROOT>] [--configfile <FILE>] [--disable-parallel]
    [-f|--force] [--force-evaluate] [--ignore-failed-sources]
    [--interactive] [--lock-file-path <LOCK_FILE_PATH>] [--locked-mode]
    [--no-cache] [--no-dependencies] [--packages <PACKAGES_DIRECTORY>]
    [-r|--runtime <RUNTIME_IDENTIFIER>] [-s|--source <SOURCE>]
    [--use-lock-file] [-v|--verbosity <LEVEL>]

dotnet restore -h|--help
```

Description

The `dotnet restore` command uses NuGet to restore dependencies as well as project-specific tools that are specified in the project file. In most cases, you don't need to explicitly use the `dotnet restore` command, since a NuGet restore is run implicitly if necessary when you run the following commands:

- `dotnet new`
- `dotnet build`
- `dotnet build-server`
- `dotnet run`
- `dotnet test`
- `dotnet publish`
- `dotnet pack`

Sometimes, it might be inconvenient to run the implicit NuGet restore with these commands. For example, some automated systems, such as build systems, need to call `dotnet restore` explicitly to control when the restore occurs so that they can control network usage. To prevent the implicit NuGet restore, you can use the `--no-restore` flag with any of these commands to disable implicit restore.

Specify feeds

To restore the dependencies, NuGet needs the feeds where the packages are located. Feeds are usually provided via the *nuget.config* configuration file. A default configuration file is provided when the .NET SDK is installed. To specify additional feeds, do one of the following:

- Create your own *nuget.config* file in the project directory. For more information, see [Common NuGet configurations](#) and [nuget.config differences](#) later in this article.
- Use `dotnet nuget` commands such as `dotnet nuget add source`.

You can override the *nuget.config* feeds with the `-s` option.

For information about how to use authenticated feeds, see [Consuming packages from authenticated feeds](#).

Global packages folder

For dependencies, you can specify where the restored packages are placed during the restore operation using the `--packages` argument. If not specified, the default NuGet package cache is used, which is found in the `.nuget/packages` directory in the user's home directory on all operating systems. For example, `/home/user1` on Linux or `C:\Users\user1` on Windows.

Project-specific tooling

For project-specific tooling, `dotnet restore` first restores the package in which the tool is packed, and then proceeds to restore the tool's dependencies as specified in its project file.

nuget.config differences

The behavior of the `dotnet restore` command is affected by the settings in the *nuget.config* file, if present. For example, setting the `globalPackagesFolder` in *nuget.config* places the restored NuGet packages in the specified folder. This is an alternative to specifying the `--packages` option on the `dotnet restore` command. For more information, see the [nuget.config reference](#).

There are three specific settings that `dotnet restore` ignores:

- [bindingRedirects](#)

Binding redirects don't work with `<PackageReference>` elements and .NET only supports `<PackageReference>` elements for NuGet packages.

- [solution](#)

This setting is Visual Studio specific and doesn't apply to .NET. .NET doesn't use a `packages.config` file and instead uses `<PackageReference>` elements for NuGet packages.

- [trustedSigners](#)

Support for cross-platform package signature verification was added in the .NET 5.0.100 SDK.

Workload manifest downloads

When you run this command, it initiates an asynchronous background download of advertising manifests for workloads. If the download is still running when this command finishes, the download is stopped. For more information, see [Advertising manifests](#).

Arguments

- `ROOT`

Optional path to the project file to restore.

Options

- `--configfile <FILE>`

The NuGet configuration file (*nuget.config*) to use. If specified, only the settings from this file will be used. If not specified, the hierarchy of configuration files from the current directory will be used. For more information, see [Common NuGet Configurations](#).

- `--disable-parallel`

Disables restoring multiple projects in parallel.

- `--force`

Forces all dependencies to be resolved even if the last restore was successful. Specifying this flag is the same as deleting the `project.assets.json` file.

- `--force-evaluate`

Forces restore to reevaluate all dependencies even if a lock file already exists.

- `-?|-h|--help`

Prints out a description of how to use the command.

- `--ignore-failed-sources`

Only warn about failed sources if there are packages meeting the version requirement.

- `--interactive`

Allows the command to stop and wait for user input or action. For example, to complete authentication.

- `--lock-file-path <LOCK_FILE_PATH>`

Output location where project lock file is written. By default, this is `PROJECT_ROOT\packages\lock.json`.

- `--locked-mode`

Don't allow updating project lock file.

- `--no-cache`

Specifies to not cache HTTP requests.

- `--no-dependencies`

When restoring a project with project-to-project (P2P) references, restores the root project and not the references.

- `--packages <PACKAGES_DIRECTORY>`

Specifies the directory for restored packages.

- `-r|--runtime <RUNTIME_IDENTIFIER>`

Specifies a runtime for the package restore. This is used to restore packages for runtimes not explicitly listed in the `<RuntimeIdentifiers>` tag in the `.csproj` file. For a list of Runtime Identifiers (RIDs), see the [RID catalog](#).

- `-s|--source <SOURCE>`

Specifies the URI of the NuGet package source to use during the restore operation. This setting overrides all of the sources specified in the `nuget.config` files. Multiple sources can be provided by specifying this option multiple times.

- `--use-lock-file`

Enables project lock file to be generated and used with restore.

- `-v|--verbosity <LEVEL>`

Sets the verbosity level of the command. Allowed values are `q[uiet]`, `m[inimal]`, `n[ormal]`, `d[etailed]`, and `diag[nostic]`. The default is `minimal`. For more information, see [LoggerVerbosity](#).

Examples

- Restore dependencies and tools for the project in the current directory:

```
dotnet restore
```

- Restore dependencies and tools for the `app1` project found in the given path:

```
dotnet restore ./projects/app1/app1.csproj
```

- Restore the dependencies and tools for the project in the current directory using the file path provided as the source:

```
dotnet restore -s c:\packages\mypackages
```

- Restore the dependencies and tools for the project in the current directory using the two file paths provided as sources:

```
dotnet restore -s c:\packages\mypackages -s c:\packages\myotherpackages
```

- Restore dependencies and tools for the project in the current directory showing detailed output:

```
dotnet restore --verbosity detailed
```

dotnet run

9/20/2022 • 5 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 3.1 SDK and later versions

Name

`dotnet run` - Runs source code without any explicit compile or launch commands.

Synopsis

```
dotnet run [-a|--arch <ARCHITECTURE>] [-c|--configuration <CONFIGURATION>]
[-f|--framework <FRAMEWORK>] [--force] [--interactive]
[--launch-profile <NAME>] [--no-build]
[--no-dependencies] [--no-launch-profile] [--no-restore]
[--os <OS>] [--project <PATH>] [-r|--runtime <RUNTIME_IDENTIFIER>]
[-v|--verbosity <LEVEL>] [[--] [application arguments]]  
dotnet run -h|--help
```

Description

The `dotnet run` command provides a convenient option to run your application from the source code with one command. It's useful for fast iterative development from the command line. The command depends on the `dotnet build` command to build the code. Any requirements for the build, such as that the project must be restored first, apply to `dotnet run` as well.

NOTE

`dotnet run` doesn't respect arguments like `/property:property=value`, which are respected by `dotnet build`.

Output files are written into the default location, which is `bin/<configuration>/<target>`. For example if you have a `netcoreapp2.1` application and you run `dotnet run`, the output is placed in `bin/Debug/netcoreapp2.1`. Files are overwritten as needed. Temporary files are placed in the `obj` directory.

If the project specifies multiple frameworks, executing `dotnet run` results in an error unless the `-f|--framework <FRAMEWORK>` option is used to specify the framework.

The `dotnet run` command is used in the context of projects, not built assemblies. If you're trying to run a framework-dependent application DLL instead, you must use `dotnet` without a command. For example, to run `myapp.dll`, use:

```
dotnet myapp.dll
```

For more information on the `dotnet` driver, see the [.NET Command Line Tools \(CLI\)](#) topic.

To run the application, the `dotnet run` command resolves the dependencies of the application that are outside of the shared runtime from the NuGet cache. Because it uses cached dependencies, it's not recommended to use `dotnet run` to run applications in production. Instead, [create a deployment](#) using the `dotnet publish` command

and deploy the published output.

Implicit restore

You don't have to run `dotnet restore` because it's run implicitly by all commands that require a restore to occur, such as `dotnet new`, `dotnet build`, `dotnet run`, `dotnet test`, `dotnet publish`, and `dotnet pack`. To disable implicit restore, use the `--no-restore` option.

The `dotnet restore` command is still useful in certain scenarios where explicitly restoring makes sense, such as [continuous integration builds in Azure DevOps Services](#) or in build systems that need to explicitly control when the restore occurs.

For information about how to manage NuGet feeds, see the [dotnet restore documentation](#).

This command supports the `dotnet restore` options when passed in the long form (for example, `--source`). Short form options, such as `-s`, are not supported.

Workload manifest downloads

When you run this command, it initiates an asynchronous background download of advertising manifests for workloads. If the download is still running when this command finishes, the download is stopped. For more information, see [Advertising manifests](#).

Options

- `--`

Delimits arguments to `dotnet run` from arguments for the application being run. All arguments after this delimiter are passed to the application run.

- `-a|--arch <ARCHITECTURE>`

Specifies the target architecture. This is a shorthand syntax for setting the [Runtime Identifier \(RID\)](#), where the provided value is combined with the default RID. For example, on a `win-x64` machine, specifying `--arch x86` sets the RID to `win-x86`. If you use this option, don't use the `-r|--runtime` option. Available since .NET 6 Preview 7.

- `-c|--configuration <CONFIGURATION>`

Defines the build configuration. The default for most projects is `Debug`, but you can override the build configuration settings in your project.

- `-f|--framework <FRAMEWORK>`

Builds and runs the app using the specified [framework](#). The framework must be specified in the project file.

- `--force`

Forces all dependencies to be resolved even if the last restore was successful. Specifying this flag is the same as deleting the `project.assets.json` file.

- `-?|-h|--help`

Prints out a description of how to use the command.

- `--interactive`

Allows the command to stop and wait for user input or action. For example, to complete authentication. Available since .NET Core 3.0 SDK.

- `--launch-profile <NAME>`

The name of the launch profile (if any) to use when launching the application. Launch profiles are defined in the `launchSettings.json` file and are typically called `Development`, `Staging`, and `Production`. For more information, see [Working with multiple environments](#).

- `--no-build`

Doesn't build the project before running. It also implicitly sets the `--no-restore` flag.

- `--no-dependencies`

When restoring a project with project-to-project (P2P) references, restores the root project and not the references.

- `--no-launch-profile`

Doesn't try to use `launchSettings.json` to configure the application.

- `--no-restore`

Doesn't execute an implicit restore when running the command.

- `--os <OS>`

Specifies the target operating system (OS). This is a shorthand syntax for setting the [Runtime Identifier \(RID\)](#), where the provided value is combined with the default RID. For example, on a `win-x64` machine, specifying `--os linux` sets the RID to `linux-x64`. If you use this option, don't use the `-r|--runtime` option. Available since .NET 6.

- `--project <PATH>`

Specifies the path of the project file to run (folder name or full path). If not specified, it defaults to the current directory.

The `-p` abbreviation for `--project` is [deprecated](#) starting in .NET 6 SDK. For a limited time starting in .NET 6 RC1 SDK, `-p` can still be used for `--project` despite the deprecation warning. If the argument provided for the option doesn't contain `=`, the command accepts `-p` as short for `--project`. Otherwise, the command assumes that `-p` is short for `--property`. This flexible use of `-p` for `--project` will be phased out in .NET 7.

- `--property:<NAME>=<VALUE>`

Sets one or more MSBuild properties. Specify multiple properties delimited by semicolons or by repeating the option:

```
--property:<NAME1>=<VALUE1>;<NAME2>=<VALUE2>
--property:<NAME1>=<VALUE1> --property:<NAME2>=<VALUE2>
```

The short form `-p` can be used for `--property`. If the argument provided for the option contains `=`, `-p` is accepted as short for `--property`. Otherwise, the command assumes that `-p` is short for `--project`.

To pass `--property` to the application rather than set an MSBuild property, provide the option after the `--` syntax separator, for example:

```
dotnet run -- --property name=value
```

- `-r|--runtime <RUNTIME_IDENTIFIER>`

Specifies the target runtime to restore packages for. For a list of Runtime Identifiers (RIDs), see the [RID catalog](#).

- `-v|--verbosity <LEVEL>`

Sets the verbosity level of the command. Allowed values are `q[uiet]`, `m[inimal]`, `n[ormal]`, `d[etailed]`, and `diag[nostic]`. The default is `minimal`. For more information, see [LoggerVerbosity](#).

Examples

- Run the project in the current directory:

```
dotnet run
```

- Run the specified project:

```
dotnet run --project ./projects/proj1/proj1.csproj
```

- Run the project in the current directory, specifying Release configuration:

```
dotnet run --property:Configuration=Release
```

- Run the project in the current directory (the `--help` argument in this example is passed to the application, since the blank `--` option is used):

```
dotnet run --configuration Release -- --help
```

- Restore dependencies and tools for the project in the current directory only showing minimal output and then run the project:

```
dotnet run --verbosity m
```

dotnet sdk check

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET 6 and later versions

Name

`dotnet sdk check` - Lists the latest available version of the .NET SDK and .NET Runtime, for each feature band.

Synopsis

```
dotnet sdk check
```

```
dotnet sdk check -h|--help
```

Description

The `dotnet sdk check` command makes it easier to track when new versions of the SDK and Runtimes are available. Within each feature band it tells you:

- The latest available version of the .NET SDK and .NET Runtime.
- Whether your installed versions are up-to-date or out of support.

Here's an example of output from the command:

.NET SDKs:		
Version	Status	
2.1.816	Up to date.	
2.2.401	.NET 2.2 is out of support.	
3.1.410	Up to date.	
5.0.204	Up to date.	
5.0.301	Up to date.	
.NET Runtimes:		
Name	Version	Status
Microsoft.AspNetCore.All	2.1.28	Up to date.
Microsoft.AspNetCore.App	2.1.28	Up to date.
Microsoft.NETCore.App	2.1.28	Up to date.
Microsoft.AspNetCore.All	2.2.6	.NET 2.2 is out of support.
Microsoft.AspNetCore.App	2.2.6	.NET 2.2 is out of support.
Microsoft.NETCore.App	2.2.6	.NET 2.2 is out of support.
Microsoft.AspNetCore.App	3.1.16	Up to date.
Microsoft.NETCore.App	3.1.16	Up to date.
Microsoft.WindowsDesktop.App	3.1.16	Up to date.
Microsoft.AspNetCore.App	5.0.7	Up to date.
Microsoft.NETCore.App	5.0.7	Up to date.
Microsoft.WindowsDesktop.App	5.0.7	Up to date.

Options

- `-?|-h|--help`

Prints out a description of how to use the command.

Examples

- Show up-to-date status of installed .NET SDKs and .NET Runtimes.

```
dotnet sdk check
```

dotnet sln

9/20/2022 • 3 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 3.1 SDK and later versions

Name

`dotnet sln` - Lists or modifies the projects in a .NET solution file.

Synopsis

```
dotnet sln [<SOLUTION_FILE>] [command]
```

```
dotnet sln [command] -h|--help
```

Description

The `dotnet sln` command provides a convenient way to list and modify projects in a solution file.

Create a solution file

To use the `dotnet sln` command, the solution file must already exist. If you need to create one, use the `dotnet new` command with the `sln` template name.

The following example creates a `.sln` file in the current folder, with the same name as the folder:

```
dotnet new sln
```

The following example creates a `.sln` file in the current folder, with the specified file name:

```
dotnet new sln --name MySolution
```

The following example creates a `.sln` file in the specified folder, with the same name as the folder:

```
dotnet new sln --output MySolution
```

Arguments

- `SOLUTION_FILE`

The solution file to use. If this argument is omitted, the command searches the current directory for one. If it finds no solution file or multiple solution files, the command fails.

Options

- `-?|-h|--help`

Prints out a description of how to use the command.

Commands

list

Lists all projects in a solution file.

Synopsis

```
dotnet sln list [-h|--help]
```

Arguments

- `SOLUTION_FILE`

The solution file to use. If this argument is omitted, the command searches the current directory for one. If it finds no solution file or multiple solution files, the command fails.

Options

- `-?|-h|--help`

Prints out a description of how to use the command.

add

Adds one or more projects to the solution file.

Synopsis

```
dotnet sln [<SOLUTION_FILE>] add [--in-root] [-s|--solution-folder <PATH>] <PROJECT_PATH>
[<PROJECT_PATH>...]
dotnet sln add [-h|--help]
```

Arguments

- `SOLUTION_FILE`

The solution file to use. If it is unspecified, the command searches the current directory for one and fails if there are multiple solution files.

- `PROJECT_PATH`

The path to the project or projects to add to the solution. Unix/Linux shell [globbing pattern](#) expansions are processed correctly by the `dotnet sln` command.

If `PROJECT_PATH` includes folders that contain the project folder, that portion of the path is used to create [solution folders](#). For example, the following commands create a solution with `myapp` in solution folder `folder1/folder2`:

```
dotnet new sln
dotnet new console --output folder1/folder2/myapp
dotnet sln add folder1/folder2/myapp
```

You can override this default behavior by using the `--in-root` or the `-s|--solution-folder <PATH>` option.

Options

- `-?|-h|--help`

Prints out a description of how to use the command.

- `--in-root`

Places the projects in the root of the solution, rather than creating a [solution folder](#). Can't be used with `-s|--solution-folder`.

- `-s|--solution-folder <PATH>`

The destination [solution folder](#) path to add the projects to. Can't be used with `--in-root`.

`remove`

Removes a project or multiple projects from the solution file.

Synopsis

```
dotnet sln [<SOLUTION_FILE>] remove <PROJECT_PATH> [<PROJECT_PATH>...]  
dotnet sln [<SOLUTION_FILE>] remove [-h|--help]
```

Arguments

- `SOLUTION_FILE`

The solution file to use. If it is unspecified, the command searches the current directory for one and fails if there are multiple solution files.

- `PROJECT_PATH`

The path to the project or projects to remove from the solution. Unix/Linux shell [globbing pattern](#) expansions are processed correctly by the `dotnet sln` command.

Options

- `-?|-h|--help`

Prints out a description of how to use the command.

Examples

- List the projects in a solution:

```
dotnet sln todo.sln list
```

- Add a C# project to a solution:

```
dotnet sln add todo-app/todo-app.csproj
```

- Remove a C# project from a solution:

```
dotnet sln remove todo-app/todo-app.csproj
```

- Add multiple C# projects to the root of a solution:

```
dotnet sln todo.sln add todo-app/todo-app.csproj back-end/back-end.csproj --in-root
```

- Add multiple C# projects to a solution:

```
dotnet sln todo.sln add todo-app/todo-app.csproj back-end/back-end.csproj
```

- Remove multiple C# projects from a solution:

```
dotnet sln todo.sln remove todo-app/todo-app.csproj back-end/back-end.csproj
```

- Add multiple C# projects to a solution using a globbing pattern (Unix/Linux only):

```
dotnet sln todo.sln add **/*.csproj
```

- Add multiple C# projects to a solution using a globbing pattern (Windows PowerShell only):

```
dotnet sln todo.sln add (ls -r **/*.csproj)
```

- Remove multiple C# projects from a solution using a globbing pattern (Unix/Linux only):

```
dotnet sln todo.sln remove **/*.csproj
```

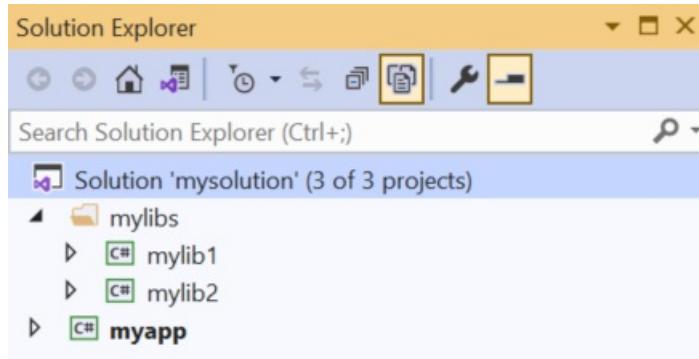
- Remove multiple C# projects from a solution using a globbing pattern (Windows PowerShell only):

```
dotnet sln todo.sln remove (ls -r **/*.csproj)
```

- Create a solution, a console app, and two class libraries. Add the projects to the solution, and use the `--solution-folder` option of `dotnet sln` to organize the class libraries into a solution folder.

```
dotnet new sln -n mysolution
dotnet new console -o myapp
dotnet new classlib -o mylib1
dotnet new classlib -o mylib2
dotnet sln mysolution.sln add myapp\myapp.csproj
dotnet sln mysolution.sln add mylib1\mylib1.csproj --solution-folder mylibs
dotnet sln mysolution.sln add mylib2\mylib2.csproj --solution-folder mylibs
```

The following screenshot shows the result in Visual Studio 2019 Solution Explorer:



See also

- [dotnet/sdk GitHub repo](#) (.NET CLI source)

dotnet store

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 3.1 SDK and later versions

Name

`dotnet store` - Stores the specified assemblies in the [runtime package store](#).

Synopsis

```
dotnet store -m|--manifest <PATH_TO_MANIFEST_FILE>
-f|--framework <FRAMEWORK_VERSION> -r|--runtime <RUNTIME_IDENTIFIER>
[--framework-version <FRAMEWORK_VERSION>] [--output <OUTPUT_DIRECTORY>]
[--skip-optimization] [--skip-symbols] [-v|--verbosity <LEVEL>]
[--working-dir <WORKING_DIRECTORY>]

dotnet store -h|--help
```

Description

`dotnet store` stores the specified assemblies in the [runtime package store](#). By default, assemblies are optimized for the target runtime and framework. For more information, see the [runtime package store](#) topic.

Required options

- `-f|--framework <FRAMEWORK>`

Specifies the [target framework](#). The target framework has to be specified in the project file.

- `-m|--manifest <PATH_TO_MANIFEST_FILE>`

The *package store manifest file* is an XML file that contains the list of packages to store. The format of the manifest file is compatible with the SDK-style project format. So, a project file that references the desired packages can be used with the `-m|--manifest` option to store assemblies in the runtime package store. To specify multiple manifest files, repeat the option and path for each file. For example:

```
--manifest packages1.csproj --manifest packages2.csproj .
```

- `-r|--runtime <RUNTIME_IDENTIFIER>`

The [runtime identifier](#) to target.

Optional options

- `--framework-version <FRAMEWORK_VERSION>`

Specifies the .NET SDK version. This option enables you to select a specific framework version beyond the framework specified by the `-f|--framework` option.

- `-?|-h|--help`

Prints out a description of how to use the command.

- `-o|--output <OUTPUT_DIRECTORY>`

Specifies the path to the runtime package store. If not specified, it defaults to the *store* subdirectory of the user profile .NET installation directory.

- `--skip-optimization`

Skips the optimization phase. For more information about optimization, see [Preparing a runtime environment](#).

- `--skip-symbols`

Skips symbol generation. Currently, you can only generate symbols on Windows and Linux.

- `-v|--verbosity <LEVEL>`

Sets the verbosity level of the command. Allowed values are `q[uiet]`, `m[inimal]`, `n[ormal]`, `d[etailed]`, and `diag[nostic]`. For more information, see [LoggerVerbosity](#).

- `-w|--working-dir <WORKING_DIRECTORY>`

The working directory used by the command. If not specified, it uses the *obj* subdirectory of the current directory.

Examples

- Store the packages specified in the *packages.csproj* project file for .NET 6.0.1:

```
dotnet store --manifest packages.csproj --framework-version 6.0.1 --framework net6.0 --runtime win-x64
```

- Store the packages specified in the *packages.csproj* without optimization:

```
dotnet store --manifest packages.csproj --skip-optimization --framework net6.0 --runtime linux-x64
```

See also

- [Runtime package store](#)

dotnet test

9/20/2022 • 11 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 3.1 SDK and later versions

Name

`dotnet test` - .NET test driver used to execute unit tests.

Synopsis

```
dotnet test [<PROJECT> | <SOLUTION> | <DIRECTORY> | <DLL> | <EXE>]
[-a|--test-adapter-path <ADAPTER_PATH>]
[--arch <ARCHITECTURE>]
[--blame]
[--blame-crash]
[--blame-crash-dump-type <DUMP_TYPE>]
[--blame-crash-collect-always]
[--blame-hang]
[--blame-hang-dump-type <DUMP_TYPE>]
[--blame-hang-timeout <TIMESPAN>]
[-c|--configuration <CONFIGURATION>]
[--collect <DATA_COLLECTOR_NAME>]
[-d|--diag <LOG_FILE>]
[-f|--framework <FRAMEWORK>]
[-e|--environment <NAME="VALUE">]
[--filter <EXPRESSION>]
[--interactive]
[-l|--logger <LOGGER>]
[--no-build]
[--nologo]
[--no-restore]
[-o|--output <OUTPUT_DIRECTORY>]
[--os <OS>]
[-r|--results-directory <RESULTS_DIR>]
[--runtime <RUNTIME_IDENTIFIER>]
[-s|--settings <SETTINGS_FILE>]
[-t|--list-tests]
[-v|--verbosity <LEVEL>]
[<args>...]
[[--] <RunSettings arguments>]

dotnet test -h|--help
```

Description

The `dotnet test` command is used to execute unit tests in a given solution. The `dotnet test` command builds the solution and runs a test host application for each test project in the solution. The test host executes tests in the given project using a test framework, for example: MSTest, NUnit, or xUnit, and reports the success or failure of each test. If all tests are successful, the test runner returns 0 as an exit code; otherwise if any test fails, it returns 1.

For multi-targeted projects, tests are run for each targeted framework. The test host and the unit test framework are packaged as NuGet packages and are restored as ordinary dependencies for the project.

Test projects specify the test runner using an ordinary `<PackageReference>` element, as seen in the following

sample project file:

```
<Project Sdk="Microsoft.NET.Sdk">

<PropertyGroup>
  <TargetFramework>netcoreapp3.1</TargetFramework>
</PropertyGroup>

<ItemGroup>
  <PackageReference Include="Microsoft.NET.Test.Sdk" Version="17.3.1" />
  <PackageReference Include="xunit" Version="2.4.2" />
  <PackageReference Include="xunit.runner.visualstudio" Version="2.4.5" />
</ItemGroup>

</Project>
```

Where `Microsoft.NET.Test.Sdk` is the test host, `xunit` is the test framework. And `xunit.runner.visualstudio` is a test adapter, which allows the xUnit framework to work with the test host.

Implicit restore

You don't have to run `dotnet restore` because it's run implicitly by all commands that require a restore to occur, such as `dotnet new`, `dotnet build`, `dotnet run`, `dotnet test`, `dotnet publish`, and `dotnet pack`. To disable implicit restore, use the `--no-restore` option.

The `dotnet restore` command is still useful in certain scenarios where explicitly restoring makes sense, such as [continuous integration builds in Azure DevOps Services](#) or in build systems that need to explicitly control when the restore occurs.

For information about how to manage NuGet feeds, see the [dotnet restore documentation](#).

Workload manifest downloads

When you run this command, it initiates an asynchronous background download of advertising manifests for workloads. If the download is still running when this command finishes, the download is stopped. For more information, see [Advertising manifests](#).

Arguments

- `PROJECT | SOLUTION | DIRECTORY | DLL | EXE`

- Path to the test project.
- Path to the solution.
- Path to a directory that contains a project or a solution.
- Path to a test project `.dll` file.
- Path to a test project `.exe` file.

If not specified, the effect is the same as using the `DIRECTORY` argument to specify the current directory.

Options

- `-a|--test-adapter-path <ADAPTER_PATH>`

Path to a directory to be searched for additional test adapters. Only `.dll` files with suffix `.TestAdapter.dll` are inspected. If not specified, the directory of the test `.dll` is searched.

- `--arch <ARCHITECTURE>`

Specifies the target architecture. This is a shorthand syntax for setting the [Runtime Identifier \(RID\)](#), where the provided value is combined with the default RID. For example, on a `win-x64` machine, specifying

--arch x86 sets the RID to win-x86. If you use this option, don't use the -r|--runtime option. Available since .NET 6 Preview 7.

- --blame

Runs the tests in blame mode. This option is helpful in isolating problematic tests that cause the test host to crash. When a crash is detected, it creates a sequence file in TestResults/<Guid>/<Guid>_Sequence.xml that captures the order of tests that were run before the crash.

This option does not create a memory dump and is not helpful when the test is hanging.

- --blame-crash (Available since .NET 5.0 SDK)

Runs the tests in blame mode and collects a crash dump when the test host exits unexpectedly. This option depends on the version of .NET used, the type of error, and the operating system.

For exceptions in managed code, a dump will be automatically collected on .NET 5.0 and later versions. It will generate a dump for testhost or any child process that also ran on .NET 5.0 and crashed. Crashes in native code will not generate a dump. This option works on Windows, macOS, and Linux.

Crash dumps in native code, or when using .NET Core 3.1 or earlier versions, can only be collected on Windows, by using Procdump. A directory that contains procdump.exe and procdump64.exe must be in the PATH or PROCDUMP_PATH environment variable. [Download the tools](#). Implies --blame.

To collect a crash dump from a native application running on .NET 5.0 or later, the usage of Procdump can be forced by setting the VTEST_DUMP_FORCEPROCDUMP environment variable to 1.

- --blame-crash-dump-type <DUMP_TYPE> (Available since .NET 5.0 SDK)

The type of crash dump to be collected. Supported dump types are full (default), and mini. Implies --blame-crash.

- --blame-crash-collect-always (Available since .NET 5.0 SDK)

Collects a crash dump on expected as well as unexpected test host exit.

- --blame-hang (Available since .NET 5.0 SDK)

Run the tests in blame mode and collects a hang dump when a test exceeds the given timeout.

- --blame-hang-dump-type <DUMP_TYPE> (Available since .NET 5.0 SDK)

The type of crash dump to be collected. It should be full, mini, or none. When none is specified, test host is terminated on timeout, but no dump is collected. Implies --blame-hang.

- --blame-hang-timeout <TIMESPAN> (Available since .NET 5.0 SDK)

Per-test timeout, after which a hang dump is triggered and the test host process and all of its child processes are dumped and terminated. The timeout value is specified in one of the following formats:

- 1.5h, 1.5hour, 1.5hours
- 90m, 90min, 90minute, 90minutes
- 5400s, 5400sec, 5400second, 5400seconds
- 5400000ms, 5400000mil, 5400000millisecond, 5400000milliseconds

When no unit is used (for example, 5400000), the value is assumed to be in milliseconds. When used together with data driven tests, the timeout behavior depends on the test adapter used. For xUnit, NUnit, and MSTest 2.2.4+, the timeout is renewed after every test case. For MSTest before version 2.2.4, the timeout is used for all test cases. This option is supported on Windows with netcoreapp2.1 and later, on Linux with netcoreapp3.1 and later, and on macOS with net5.0 or later. Implies --blame and

- `--blame-hang`.
- `-c|--configuration <CONFIGURATION>`

Defines the build configuration. The default for most projects is `Debug`, but you can override the build configuration settings in your project.
- `--collect <DATA_COLLECTOR_NAME>`

Enables data collector for the test run. For more information, see [Monitor and analyze test run](#).

On Windows (x86, x64 and Arm64), Linux (x64) and macOS (x64), you can collect code coverage by using the `--collect "Code Coverage"` option. For more information, see [Use code coverage](#) and [Customize code coverage analysis](#).

To collect code coverage on any platform that is supported by .NET Core, install [Coverlet](#) and use the `--collect "XPlat Code Coverage"` option.
- `-d|--diag <LOG_FILE>`

Enables diagnostic mode for the test platform and writes diagnostic messages to the specified file and to files next to it. The process that is logging the messages determines which files are created, such as `*.host_<date>.txt` for test host log, and `*.datacollector_<date>.txt` for data collector log.
- `-e|--environment <NAME="VALUE">`

Sets the value of an environment variable. Creates the variable if it does not exist, overrides if it does exist. Use of this option will force the tests to be run in an isolated process. The option can be specified multiple times to provide multiple variables.
- `-f|--framework <FRAMEWORK>`

The [target framework moniker \(TFM\)](#) of the target framework to run tests for. The target framework must also be specified in the project file.
- `--filter <EXPRESSION>`

Filters out tests in the current project using the given expression. For more information, see the [Filter option details](#) section. For more information and examples on how to use selective unit test filtering, see [Running selective unit tests](#).
- `-?|-h|--help`

Prints out a description of how to use the command.
- `--interactive`

Allows the command to stop and wait for user input or action. For example, to complete authentication. Available since .NET Core 3.0 SDK.
- `-l|--logger <LOGGER>`

Specifies a logger for test results. Unlike MSBuild, `dotnet test` doesn't accept abbreviations: instead of `-l "console;v=d"` use `-l "console;verbosity=detailed"`. Specify the parameter multiple times to enable multiple loggers. For more information, see [Reporting test results](#), [Switches for loggers](#), and the [examples](#) later in this article.
- `--no-build`

Doesn't build the test project before running it. It also implicitly sets the `--no-restore` flag.

- `--nologo`

Run tests without displaying the Microsoft TestPlatform banner. Available since .NET Core 3.0 SDK.
- `--no-restore`

Doesn't execute an implicit restore when running the command.
- `-o|--output <OUTPUT_DIRECTORY>`

Directory in which to find the binaries to run. If not specified, the default path is `./bin/<configuration>/<framework>/`. For projects with multiple target frameworks (via the `TargetFrameworks` property), you also need to define `--framework` when you specify this option. `dotnet test` always runs tests from the output directory. You can use [AppDomain.BaseDirectory](#) to consume test assets in the output directory.
- `--os <OS>`

Specifies the target operating system (OS). This is a shorthand syntax for setting the [Runtime Identifier \(RID\)](#), where the provided value is combined with the default RID. For example, on a `win-x64` machine, specifying `--os linux` sets the RID to `linux-x64`. If you use this option, don't use the `-r|--runtime` option. Available since .NET 6.
- `-r|--results-directory <RESULTS_DIR>`

The directory where the test results are going to be placed. If the specified directory doesn't exist, it's created. The default is `TestResults` in the directory that contains the project file.
- `--runtime <RUNTIME_IDENTIFIER>`

The target runtime to test for.
- `-s|--settings <SETTINGS_FILE>`

The `.runsettings` file to use for running the tests. The `TargetPlatform` element (x86|x64) has no effect for `dotnet test`. To run tests that target x86, install the x86 version of .NET Core. The bitness of the `dotnet.exe` that is on the path is what will be used for running tests. For more information, see the following resources:

 - [Configure unit tests by using a `.runsettings` file.](#)
 - [Configure a test run](#)
- `-t|--list-tests`

List the discovered tests instead of running the tests.
- `-v|--verbosity <LEVEL>`

Sets the verbosity level of the command. Allowed values are `q[uiet]`, `m[inimal]`, `n[ormal]`, `d[etailed]`, and `diag[nostic]`. The default is `minimal`. For more information, see [LoggerVerbosity](#).
- `args`

Specifies extra arguments to pass to the adapter. Use a space to separate multiple arguments.

The list of possible arguments depends upon the specified behavior:

 - When you specify a project, solution, or a directory, or if you omit this argument, the call is forwarded to `msbuild`. In that case, the available arguments can be found in [the dotnet msbuild documentation](#).
 - When you specify a `.dll` or an `.exe`, the call is forwarded to `vstest`. In that case, the available

arguments can be found in [the dotnet vstest documentation](#).

- `RunSettings` arguments

Inline `RunSettings` are passed as the last arguments on the command line after "-- " (note the space after --).

Inline `RunSettings` are specified as `[name]=[value]` pairs. A space is used to separate multiple `[name]=[value]` pairs.

Example: `dotnet test -- MSTest.DeploymentEnabled=false MSTest.MapInconclusiveToFailed=True`

For more information, see [Passing RunSettings arguments through command line](#).

Examples

- Run the tests in the project in the current directory:

```
dotnet test
```

- Run the tests in the `test1` project:

```
dotnet test ~/projects/test1/test1.csproj
```

- Run the tests using `test1.dll` assembly:

```
dotnet test ~/projects/test1/bin/debug/test1.dll
```

- Run the tests in the project in the current directory, and generate a test results file in the `trx` format:

```
dotnet test --logger trx
```

- Run the tests in the project in the current directory, and generate a code coverage file (after installing [Coverlet](#) collectors integration):

```
dotnet test --collect:"XPlat Code Coverage"
```

- Run the tests in the project in the current directory, and generate a code coverage file (Windows only):

```
dotnet test --collect "Code Coverage"
```

- Run the tests in the project in the current directory, and log with detailed verbosity to the console:

```
dotnet test --logger "console;verbosity=detailed"
```

- Run the tests in the project in the current directory, and log with the `trx` logger to `testResults.trx` in the `TestResults` folder:

```
dotnet test --logger "trx;logfilename=testResults.trx"
```

Since the log file name is specified, the same name is used for each target framework in the case of a multi-targeted project. The output for each target framework overwrites the output for preceding target frameworks. The file is created in the `TestResults` folder in the test project folder, because relative paths

are relative to that folder. The following example shows how to produce a separate file for each target framework.

- Run the tests in the project in the current directory, and log with the trx logger to files in the *TestResults* folder, with file names that are unique for each target framework:

```
dotnet test --logger:"trx;LogFilePrefix=testResults"
```

- Run the tests in the project in the current directory, and log with the html logger to *testResults.htm* in the *TestResults* folder:

```
dotnet test --logger "html;logfilename=testResults.html"
```

- Run the tests in the project in the current directory, and report tests that were in progress when the test host crashed:

```
dotnet test --blame
```

- Run the tests in the `test1` project, providing the `-bl` (binary log) argument to `msbuild`:

```
dotnet test ~/projects/test1/test1.csproj -bl
```

- Run the tests in the `test1` project, setting the MSBuild `DefineConstants` property to `DEV`:

```
dotnet test ~/projects/test1/test1.csproj -p:DefineConstants="DEV"
```

Filter option details

```
--filter <EXPRESSION>
```

`<Expression>` has the format `<property><operator><value>[|&<Expression>]`.

`<property>` is an attribute of the `Test Case`. The following are the properties supported by popular unit test frameworks:

TEST FRAMEWORK	SUPPORTED PROPERTIES
MSTest	<ul style="list-style-type: none">FullyQualifiedNameNameClassNamePriorityTestCategory
xUnit	<ul style="list-style-type: none">FullyQualifiedNameDisplayNameCategory

TEST FRAMEWORK	SUPPORTED PROPERTIES
NUnit	<ul style="list-style-type: none"> • FullyQualifiedName • Name • TestCategory • Priority

The `<operator>` describes the relationship between the property and the value:

OPERATOR	FUNCTION
<code>=</code>	Exact match
<code>!=</code>	Not exact match
<code>~</code>	Contains
<code>!~</code>	Not contains

`<value>` is a string. All the lookups are case insensitive.

An expression without an `<operator>` is automatically considered as a `contains` on `FullyQualifiedName` property (for example, `dotnet test --filter xyz` is same as `dotnet test --filter FullyQualifiedName~xyz`).

Expressions can be joined with conditional operators:

OPERATOR	FUNCTION
<code> </code>	OR
<code>&</code>	AND

You can enclose expressions in parenthesis when using conditional operators (for example, `(Name~TestMethod1) | (Name~TestMethod2)`).

For more information and examples on how to use selective unit test filtering, see [Running selective unit tests](#).

See also

- [Frameworks and Targets](#)
- [.NET Runtime Identifier \(RID\) catalog](#)
- [Passing runsettings arguments through commandline](#)

dotnet tool install

9/20/2022 • 3 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 3.1 SDK and later versions

Name

`dotnet tool install` - Installs the specified .NET tool on your machine.

Synopsis

```
dotnet tool install <PACKAGE_NAME> -g|--global  
  [--add-source <SOURCE>] [--configfile <FILE>] [--disable-parallel]  
  [--framework <FRAMEWORK>] [--ignore-failed-sources] [--interactive]  
  [--no-cache] [--prerelease]  
  [--tool-manifest <PATH>] [-v|--verbosity <LEVEL>]  
  [--version <VERSION_NUMBER>]  
  
dotnet tool install <PACKAGE_NAME> --tool-path <PATH>  
  [--add-source <SOURCE>] [--configfile <FILE>] [--disable-parallel]  
  [--framework <FRAMEWORK>] [--ignore-failed-sources] [--interactive]  
  [--no-cache] [--prerelease]  
  [--tool-manifest <PATH>] [-v|--verbosity <LEVEL>]  
  [--version <VERSION_NUMBER>]  
  
dotnet tool install <PACKAGE_NAME> [--local]  
  [--add-source <SOURCE>] [--configfile <FILE>] [--disable-parallel]  
  [--framework <FRAMEWORK>] [--ignore-failed-sources] [--interactive]  
  [--no-cache] [--prerelease]  
  [--tool-manifest <PATH>] [-v|--verbosity <LEVEL>]  
  [--version <VERSION_NUMBER>]  
  
dotnet tool install -h|--help
```

Description

The `dotnet tool install` command provides a way for you to install .NET tools on your machine. To use the command, you specify one of the following installation options:

- To install a global tool in the default location, use the `--global` option.
- To install a global tool in a custom location, use the `--tool-path` option.
- To install a local tool, omit the `--global` and `--tool-path` options.

Global tools are installed in the following directories by default when you specify the `-g` or `--global` option:

OS	Path
Linux/macOS	<code>\$HOME/.dotnet/tools</code>
Windows	<code>%USERPROFILE%\.dotnet\tools</code>

Local tools are added to a `dotnet-tools.json` file in a `.config` directory under the current directory. If a manifest file doesn't exist yet, create it by running the following command:

```
dotnet new tool-manifest
```

For more information, see [Install a local tool](#).

Arguments

- `PACKAGE_NAME`

Name/ID of the NuGet package that contains the .NET tool to install.

Options

- `--add-source <SOURCE>`

Adds an additional NuGet package source to use during installation. Feeds are accessed in parallel, not sequentially in some order of precedence. If the same package and version is in multiple feeds, the fastest feed wins. For more information, see [What happens when a NuGet package is installed?](#).

- `--configfile <FILE>`

The NuGet configuration file (*nuget.config*) to use. If specified, only the settings from this file will be used. If not specified, the hierarchy of configuration files from the current directory will be used. For more information, see [Common NuGet Configurations](#).

- `--disable-parallel`

Prevent restoring multiple projects in parallel.

- `--framework <FRAMEWORK>`

Specifies the [target framework](#) to install the tool for. By default, the .NET SDK tries to choose the most appropriate target framework.

- `-g|--global`

Specifies that the installation is user wide. Can't be combined with the `--tool-path` option. Omitting both `--global` and `--tool-path` specifies a local tool installation.

- `-?|-h|--help`

Prints out a description of how to use the command.

- `--ignore-failed-sources`

Treat package source failures as warnings.

- `--interactive`

Allows the command to stop and wait for user input or action. For example, to complete authentication.

- `--local`

Update the tool and the local tool manifest. Can't be combined with the `--global` option or the `--tool-path` option.

- `--no-cache`

Do not cache packages and HTTP requests.

- `--prerelease`

Include prerelease packages.

- `--tool-manifest <PATH>`

Path to the manifest file.

- `--tool-path <PATH>`

Specifies the location where to install the Global Tool. PATH can be absolute or relative. If PATH doesn't exist, the command tries to create it. Omitting both `--global` and `--tool-path` specifies a local tool installation.

- `-v|--verbosity <LEVEL>`

Sets the verbosity level of the command. Allowed values are `q[uiet]`, `m[inimal]`, `n[ormal]`, `d[etailed]`, and `diag[nostic]`. For more information, see [LoggerVerbosity](#).

- `--version <VERSION_NUMBER>`

The version of the tool to install. By default, the latest stable package version is installed. Use this option to install preview or older versions of the tool.

Examples

- `dotnet tool install -g dotnetsay`

Installs `dotnetsay` as a global tool in the default location.

- `dotnet tool install dotnetsay --tool-path c:\global-tools`

Installs `dotnetsay` as a global tool in a specific Windows directory.

- `dotnet tool install dotnetsay --tool-path ~/bin`

Installs `dotnetsay` as a global tool in a specific Linux/macOS directory.

- `dotnet tool install -g dotnetsay --version 2.0.0`

Installs version 2.0.0 of `dotnetsay` as a global tool.

- `dotnet tool install dotnetsay`

Installs `dotnetsay` as a local tool for the current directory.

See also

- [.NET tools](#)
- [Tutorial: Install and use a .NET global tool using the .NET CLI](#)
- [Tutorial: Install and use a .NET local tool using the .NET CLI](#)

dotnet tool list

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 3.1 SDK and later versions

Name

`dotnet tool list` - Lists all .NET tools of the specified type currently installed on your machine.

Synopsis

```
dotnet tool list -g|--global  
dotnet tool list --tool-path <PATH>  
dotnet tool list --local  
dotnet tool list  
dotnet tool list -h|--help
```

Description

The `dotnet tool list` command provides a way for you to list all .NET global, tool-path, or local tools installed on your machine. The command lists the package name, version installed, and the tool command. To use the command, you specify one of the following:

- To list global tools installed in the default location, use the `--global` option
- To list global tools installed in a custom location, use the `--tool-path` option.
- To list local tools, use the `--local` option or omit the `--global`, `--tool-path`, and `--local` options.

Options

- `-g|--global`

Lists user-wide global tools. Can't be combined with the `--tool-path` option. Omitting both `--global` and `--tool-path` lists local tools.

- `-?|-h|--help`

Prints out a description of how to use the command.

- `--local`

Lists local tools for the current directory. Can't be combined with the `--global` or `--tool-path` options. Omitting both `--global` and `--tool-path` lists local tools even if `--local` is not specified.

- `--tool-path <PATH>`

Specifies a custom location where to find global tools. PATH can be absolute or relative. Can't be combined with the `--global` option. Omitting both `--global` and `--tool-path` lists local tools.

Examples

- `dotnet tool list -g`

Lists all global tools installed user-wide on your machine (current user profile).

- `dotnet tool list --tool-path c:\global-tools`

Lists the global tools from a specific Windows directory.

- `dotnet tool list --tool-path ~/bin`

Lists the global tools from a specific Linux/macOS directory.

- `dotnet tool list` or `dotnet tool list --local`

Lists all local tools available in the current directory.

See also

- [.NET tools](#)
- [Tutorial: Install and use a .NET global tool using the .NET CLI](#)
- [Tutorial: Install and use a .NET local tool using the .NET CLI](#)

dotnet tool restore

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 3.1 SDK and later versions

Name

`dotnet tool restore` - Installs the .NET local tools that are in scope for the current directory.

Synopsis

```
dotnet tool restore
  [--configfile <FILE>] [--add-source <SOURCE>]
  [--tool-manifest <PATH_TO_MANIFEST_FILE>] [--disable-parallel]
  [--ignore-failed-sources] [--no-cache] [--interactive]
  [-v|--verbosity <LEVEL>]

dotnet tool restore -h|--help
```

Description

The `dotnet tool restore` command finds the tool manifest file that is in scope for the current directory and installs the tools that are listed in it. For information about manifest files, see [Install a local tool](#) and [Invoke a local tool](#).

Options

- `--configfile <FILE>`

The NuGet configuration file (*nuget.config*) to use. If specified, only the settings from this file will be used. If not specified, the hierarchy of configuration files from the current directory will be used. For more information, see [Common NuGet Configurations](#).

- `--add-source <SOURCE>`

Adds an additional NuGet package source to use during installation. Feeds are accessed in parallel, not sequentially in some order of precedence. If the same package and version is in multiple feeds, the fastest feed wins. For more information, see [What happens when a NuGet package is installed?](#).

- `--tool-manifest <PATH>`

Path to the manifest file.

- `--disable-parallel`

Prevent restoring multiple projects in parallel.

- `--ignore-failed-sources`

Treat package source failures as warnings.

- `--no-cache`

Do not cache packages and http requests.

- `--interactive`

Allows the command to stop and wait for user input or action. For example, to complete authentication.

- `-?|-h|--help`

Prints out a description of how to use the command.

- `-v|--verbosity <LEVEL>`

Sets the verbosity level of the command. Allowed values are `q[uiet]` , `m[inimal]` , `n[ormal]` , `d[etailed]` , and `diag[nostic]` . For more information, see [LoggerVerbosity](#).

Example

- `dotnet tool restore`

Restores local tools for the current directory.

See also

- [.NET tools](#)
- [Tutorial: Install and use a .NET local tool using the .NET CLI](#)

dotnet tool run

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 3.1 SDK and later versions

Name

`dotnet tool run` - Invokes a local tool.

Synopsis

```
dotnet tool run <COMMAND NAME>
```

```
dotnet tool run -h|--help
```

Description

The `dotnet tool run` command searches tool manifest files that are in scope for the current directory. When it finds a reference to the specified tool, it runs the tool. For more information, see [Invoke a local tool](#).

Arguments

- `COMMAND_NAME`

The command name of the tool to run.

Options

- `-?|-h|--help`

Prints out a description of how to use the command.

Example

- `dotnet tool run dotnetsay`

Runs the `dotnetsay` local tool.

See also

- [.NET tools](#)
- [Tutorial: Install and use a .NET local tool using the .NET CLI](#)

dotnet tool search

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET 5 SDK and later versions

Name

`dotnet tool search` - Searches all .NET tools that are published to NuGet.

Synopsis

```
dotnet tool search [--detail] [--prerelease]
                   [--skip <NUMBER>] [--take <NUMBER>] <SEARCH TERM>

dotnet tool search -h|--help
```

Description

The `dotnet tool search` command provides a way for you to search NuGet for tools that can be used as .NET global, tool-path, or local tools. The command searches the tool names and metadata such as titles, descriptions, and tags.

The command uses the [NuGet Search API](#). It filters on `packageType=dotnettool` to select only .NET tool packages.

Options

- `--detail`

Shows detailed results from the query.

- `--prerelease`

Includes pre-release packages.

- `--skip <NUMBER>`

Specifies the number of query results to skip. Used for pagination.

- `--take <NUMBER>`

Specifies the number of query results to show. Used for pagination.

- `-?|-h|--help`

Prints out a description of how to use the command.

Examples

- Search NuGet.org for .NET tools that have "format" in their package name or description:

```
dotnet tool search format
```

The output looks like the following example:

Package ID	Downloads	Verified	Latest Version	Authors
dotnet-format	496746		4.1.131201	Microsoft
bsoa.generator	533		1.0.0	Microsoft

- Search NuGet.org for .NET tools that have "format" in their package name or metadata, show only the first result, and show a detailed view.

```
dotnet tool search format --take 1 --detail
```

The output looks like the following example:

```
-----  
dotnet-format  
Latest Version: 4.1.131201  
Authors: Microsoft  
Tags:  
Downloads: 496746  
Verified: False  
Description: Command line tool for formatting C# and Visual Basic code files based on .editorconfig settings.  
Versions:  
    3.0.2 Downloads: 1973  
    3.0.4 Downloads: 9064  
    3.1.37601 Downloads: 114730  
    3.2.107702 Downloads: 13423  
    3.3.111304 Downloads: 131195  
    4.0.130203 Downloads: 78610  
    4.1.131201 Downloads: 145927
```

See also

- [.NET tools](#)
- [Tutorial: Install and use a .NET global tool using the .NET CLI](#)
- [Tutorial: Install and use a .NET local tool using the .NET CLI](#)

dotnet tool uninstall

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 3.1 SDK and later versions

Name

`dotnet tool uninstall` - Uninstalls the specified .NET tool from your machine.

Synopsis

```
dotnet tool uninstall <PACKAGE_NAME> -g|--global  
dotnet tool uninstall <PACKAGE_NAME> --tool-path <PATH>  
dotnet tool uninstall <PACKAGE_NAME>  
dotnet tool uninstall -h|--help
```

Description

The `dotnet tool uninstall` command provides a way for you to uninstall .NET tools from your machine. To use the command, you specify one of the following options:

- To uninstall a global tool that was installed in the default location, use the `--global` option.
- To uninstall a global tool that was installed in a custom location, use the `--tool-path` option.
- To uninstall a local tool, omit the `--global` and `--tool-path` options.

Arguments

- `<PACKAGE_NAME>`

Name/ID of the NuGet package that contains the .NET tool to uninstall. You can find the package name using the `dotnet tool list` command.

Options

- `-g|--global`

Specifies that the tool to be removed is from a user-wide installation. Can't be combined with the `--tool-path` option. Omitting both `--global` and `--tool-path` specifies that the tool to be removed is a local tool.

- `-?|-h|--help`

Prints out a description of how to use the command.

- `--tool-path <PATH>`

Specifies the location where to uninstall the tool. PATH can be absolute or relative. Can't be combined with the `--global` option. Omitting both `--global` and `--tool-path` specifies that the tool to be removed is a local tool.

Examples

- `dotnet tool uninstall -g dotnetsay`

Uninstalls the `dotnetsay` global tool.

- `dotnet tool uninstall dotnetsay --tool-path c:\global-tools`

Uninstalls the `dotnetsay` global tool from a specific Windows directory.

- `dotnet tool uninstall dotnetsay --tool-path ~/bin`

Uninstalls the `dotnetsay` global tool from a specific Linux/macOS directory.

- `dotnet tool uninstall dotnetsay`

Uninstalls the `dotnetsay` local tool from the current directory.

See also

- [.NET tools](#)
- [Tutorial: Install and use a .NET global tool using the .NET CLI](#)
- [Tutorial: Install and use a .NET local tool using the .NET CLI](#)

dotnet tool update

9/20/2022 • 3 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 3.1 SDK and later versions

Name

`dotnet tool update` - Updates the specified .NET tool on your machine.

Synopsis

```
dotnet tool update <PACKAGE_ID> -g|--global  
  [--add-source <SOURCE>] [--configfile <FILE>]  
  [--disable-parallel] [--framework <FRAMEWORK>]  
  [--ignore-failed-sources] [--interactive] [--no-cache]  
  [-v|--verbosity <LEVEL>] [--version <VERSION>]  
  
dotnet tool update <PACKAGE_ID> --tool-path <PATH>  
  [--add-source <SOURCE>] [--configfile <FILE>]  
  [--disable-parallel] [--framework <FRAMEWORK>]  
  [--ignore-failed-sources] [--interactive] [--no-cache]  
  [-v|--verbosity <LEVEL>] [--version <VERSION>]  
  
dotnet tool update <PACKAGE_ID> --local  
  [--add-source <SOURCE>] [--configfile <FILE>]  
  [--disable-parallel] [--framework <FRAMEWORK>]  
  [--ignore-failed-sources] [--interactive] [--no-cache]  
  [--tool-manifest <PATH>]  
  [-v|--verbosity <LEVEL>] [--version <VERSION>]  
  
dotnet tool update -h|--help
```

Description

The `dotnet tool update` command provides a way for you to update .NET tools on your machine to the latest stable version of the package. The command uninstalls and reinstalls a tool, effectively updating it. To use the command, you specify one of the following options:

- To update a global tool that was installed in the default location, use the `-global` option
- To update a global tool that was installed in a custom location, use the `--tool-path` option.
- To update a local tool, use the `--local` option.

Arguments

- `<PACKAGE_ID>`

Name/ID of the NuGet package that contains the .NET global tool to update. You can find the package name using the [dotnet tool list](#) command.

Options

- `--add-source <SOURCE>`

Adds an additional NuGet package source to use during installation. Feeds are accessed in parallel, not sequentially in some order of precedence. If the same package and version is in multiple feeds, the fastest feed wins. For more information, see [What happens when a NuGet package is installed?](#)

- `--configfile <FILE>`

The NuGet configuration file (*nuget.config*) to use. If specified, only the settings from this file will be used. If not specified, the hierarchy of configuration files from the current directory will be used. For more information, see [Common NuGet Configurations](#).

- `--disable-parallel`

Prevent restoring multiple projects in parallel.

- `--framework <FRAMEWORK>`

Specifies the [target framework](#) to update the tool for.

- `-g|--global`

Specifies that the update is for a user-wide tool. Can't be combined with the `--tool-path` option. Omitting both `--global` and `--tool-path` specifies that the tool to be updated is a local tool.

- `-?|-h|--help`

Prints out a description of how to use the command.

- `--ignore-failed-sources`

Treat package source failures as warnings.

- `--interactive`

Allows the command to stop and wait for user input or action. For example, to complete authentication.

- `--local`

Update the tool and the local tool manifest. Can't be combined with the `--global` option or the `--tool-path` option.

- `--no-cache`

Do not cache packages and HTTP requests.

- `--tool-manifest <PATH>`

Path to the manifest file.

- `--tool-path <PATH>`

Specifies the location where the global tool is installed. PATH can be absolute or relative. Can't be combined with the `--global` option. Omitting both `--global` and `--tool-path` specifies that the tool to be updated is a local tool.

- `-v|--verbosity <LEVEL>`

Sets the verbosity level of the command. Allowed values are `q[uiet]`, `m[inimal]`, `n[ormal]`, `d[etailed]`, and `diag[nostic]`. For more information, see [LoggerVerbosity](#).

- `--version <VERSION>`

The version range of the tool package to update to. This cannot be used to downgrade versions, you must

`uninstall` newer versions first.

Examples

- `dotnet tool update -g dotnetsay`

Updates the `dotnetsay` global tool.

- `dotnet tool update dotnetsay --tool-path c:\global-tools`

Updates the `dotnetsay` global tool located in a specific Windows directory.

- `dotnet tool update dotnetsay --tool-path ~/bin`

Updates the `dotnetsay` global tool located in a specific Linux/macOS directory.

- `dotnet tool update dotnetsay`

Updates the `dotnetsay` local tool installed for the current directory.

- `dotnet tool update -g dotnetsay --version 2.0.*`

Updates the `dotnetsay` global tool to the latest patch version, with a major version of `2`, and a minor version of `0`.

- `dotnet tool update -g dotnetsay --version (2.0.*,2.1.4)`

Updates the `dotnetsay` global tool to the lowest version within the specified range `(> 2.0.0 && < 2.1.4)`, version `2.1.0` would be installed. For more information on semantic versioning ranges, see [NuGet packaging version ranges](#).

See also

- [.NET tools](#)
- [Semantic versioning](#)
- [Tutorial: Install and use a .NET global tool using the .NET CLI](#)
- [Tutorial: Install and use a .NET local tool using the .NET CLI](#)

dotnet vstest

9/20/2022 • 3 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 3.1 SDK and later versions

IMPORTANT

The `dotnet vstest` command is superseded by `dotnet test`, which can now be used to run assemblies. See [dotnet test](#).

Name

`dotnet vstest` - Runs tests from the specified assemblies.

Synopsis

```
dotnet vstest [<TEST_FILE_NAMES>] [--Blame] [--Diag <PATH_TO_LOG_FILE>]
    [--Framework <FRAMEWORK>] [--InIsolation] [-lt|--ListTests <FILE_NAME>]
    [--logger <LOGGER_URI/FRIENDLY_NAME>] [--Parallel]
    [--ParentProcessId <PROCESS_ID>] [--Platform] <PLATFORM_TYPE>
    [--Port <PORT>] [--ResultsDirectory<PATH>] [--Settings <SETTINGS_FILE>]
    [--TestAdapterPath <PATH>] [--TestCaseFilter <EXPRESSION>]
    [--Tests <TEST_NAMES>] [[--] <args>...]]]

dotnet vstest -?|--Help
```

Description

The `dotnet vstest` command runs the `vstest.console` command-line application to run automated unit tests.

Arguments

- `TEST_FILE_NAMES`

Run tests from the specified assemblies. Separate multiple test assembly names with spaces. Wildcards are supported.

Options

- `--Blame`

Runs the tests in blame mode. This option is helpful in isolating the problematic tests causing test host to crash. It creates an output file in the current directory as *Sequence.xml* that captures the order of tests execution before the crash.

- `--Diag <PATH_TO_LOG_FILE>`

Enables verbose logs for the test platform. Logs are written to the provided file.

- `--Framework <FRAMEWORK>`

Target .NET Framework version used for test execution. Examples of valid values are

.NETFramework,Version=v4.6 or .NETCoreApp,Version=v1.0. Other supported values are Framework40, Framework45, FrameworkCore10, and FrameworkUap10.

- --InIsolation

Runs the tests in an isolated process. This makes *vstest.console.exe* process less likely to be stopped on an error in the tests, but tests may run slower.

- -lt|--ListTests <FILE_NAME>

Lists all discovered tests from the given test container.

- --logger <LOGGER_URI/FRIENDLY_NAME>

Specify a logger for test results.

- To publish test results to Team Foundation Server, use the TfsPublisher logger provider:

```
/logger:TfsPublisher;
    Collection=<team project collection url>;
    BuildName=<build name>;
    TeamProject=<team project name>
    [;Platform=<Defaults to "Any CPU">]
    [;Flavor=<Defaults to "Debug">]
    [;RunTitle=<title>]
```

- To log results to a Visual Studio Test Results File (TRX), use the trx logger provider. This switch creates a file in the test results directory with given log file name. If LogFileName isn't provided, a unique file name is created to hold the test results.

```
/logger:trx [;LogFileName=<Defaults to unique file name>]
```

- --Parallel

Run tests in parallel. By default, all available cores on the machine are available for use. Specify an explicit number of cores by setting the MaxCpuCount property under the RunConfiguration node in the *runsettings* file.

- --ParentProcessId <PROCESS_ID>

Process ID of the parent process responsible for launching the current process.

- --Platform <PLATFORM_TYPE>

Target platform architecture used for test execution. Valid values are x86, x64, and ARM.

- --Port <PORT>

Specifies the port for the socket connection and receiving the event messages.

- --ResultsDirectory:<PATH>

Test results directory will be created in specified path if not exists.

- --Settings <SETTINGS_FILE>

Settings to use when running tests.

- --TestAdapterPath <PATH>

Use custom test adapters from a given path (if any) in the test run.

- `--TestCaseFilter <EXPRESSION>`

Run tests that match the given expression. `<EXPRESSION>` is of the format `<property>Operator<value>[|&<EXPRESSION>]`, where Operator is one of `=`, `!=`, or `~`. Operator `~` has 'contains' semantics and is applicable for string properties like `DisplayName`. Parentheses `()` are used to group subexpressions. For more information, see [TestCase filter](#).

- `--Tests <TEST_NAMES>`

Run tests with names that match the provided values. Separate multiple values with commas.

- `-?|--Help`

Prints out a short help for the command.

- `@<file>`

Reads response file for more options.

- `args`

Specifies extra arguments to pass to the adapter. Arguments are specified as name-value pairs of the form `<n>=<v>`, where `<n>` is the argument name and `<v>` is the argument value. Use a space to separate multiple arguments.

Examples

Run tests in `mytestproject.dll`:

```
dotnet vstest mytestproject.dll
```

Run tests in `mytestproject.dll`, exporting to custom folder with custom name:

```
dotnet vstest mytestproject.dll --logger:"trx;LogFileName=custom_file_name.trx" --
ResultsDirectory:custom/file/path
```

Run tests in `mytestproject.dll` and `myothertestproject.exe`:

```
dotnet vstest mytestproject.dll myothertestproject.exe
```

Run `TestMethod1` tests:

```
dotnet vstest /Tests:TestMethod1
```

Run `TestMethod1` and `TestMethod2` tests:

```
dotnet vstest /Tests:TestMethod1,TestMethod2
```

See also

- [VSTest.Console.exe command-line options](#)

dotnet watch

9/20/2022 • 6 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 3.1 SDK and later versions

Name

`dotnet watch` - Restarts or [hot reloads](#) the specified application when changes in the source code are detected.

Synopsis

```
dotnet watch [--list]
[--no-hot-reload] [--non-interactive]
[--project <PROJECT>]
[-q|--quiet] [-v|--verbose]
[--version]
[--] <forwarded arguments>

dotnet watch -?|-h|--help
```

Description

The `dotnet watch` command is a file watcher. When it detects a change that is supported for [hot reload](#), it hot reloads the specified application. When it detects an unsupported change, it restarts the application. This process enables fast iterative development from the command line.

While running `dotnet watch`, you can force the app to rebuild and restart by pressing Ctrl+R in the command shell. This feature is available only while the app is running. For example, if you run `dotnet watch` on a console app that ends before you press Ctrl+R, pressing Ctrl+R has no effect. However, in that case `dotnet watch` is still watching files and will restart the app if a file is updated.

Arguments

- `forwarded arguments`

Arguments to pass to the child `dotnet` process. For example: `run` with options for [dotnet run](#) or `test` with options for [dotnet test](#). If the child command isn't specified, the default is `run` for `dotnet run`.

Options

- `--list`

Lists all discovered files without starting the watcher.

- `--no-hot-reload`

Suppress [hot reload](#) for supported apps.

- `--non-interactive`

Runs `dotnet watch` in non-interactive mode. Use this option to prevent console input from being requested. When hot reload is enabled and a [rude edit](#) is detected, dotnet watch restarts the app.

- `--project <PATH>`
Specifies the path of the project file to run (folder only or including the project file name). If not specified, it defaults to the current directory.
- `-q|--quiet`
Suppresses all output that is generated by the `dotnet watch` command except warnings and errors. The option is not passed on to child commands. For example, output from `dotnet restore` and `dotnet run` continues to be output.
- `-v|--verbose`
Shows verbose output for debugging.
- `--version`
Shows the version of `dotnet watch`.
- `--`
The **double-dash option ('--')** can be used to delimit `dotnet watch` options from arguments that will be passed to the child process. Its use is optional. When the double-dash option isn't used, `dotnet watch` considers the first unrecognized argument to be the beginning of arguments that it should pass into the child `dotnet` process.

Environment variables

`dotnet watch` uses the following environment variables:

- `DOTNET_HOTRELOAD_NAMEDPIPE_NAME`
This value is configured by `dotnet watch` when the app is to be launched, and it specifies the named pipe.
- `DOTNET_USE_POLLING_FILE_WATCHER`
When set to `1` or `true`, `dotnet watch` uses a polling file watcher instead of `System.IO.FileSystemWatcher`. Polling is required for some file systems, such as network shares, Docker mounted volumes, and other virtual file systems. The `PhysicalFileProvider` class uses `DOTNET_USE_POLLING_FILE_WATCHER` to determine whether the `PhysicalFileProvider.Watch` method will rely on the `PollingFileChangeToken`.
- `DOTNET_WATCH`
`dotnet watch` sets this variable to `1` on all child processes that it launches.
- `DOTNET_WATCH_AUTO_RELOAD_WS_HOSTNAME`
As part of `dotnet watch`, the browser refresh server mechanism reads this value to determine the WebSocket host environment. The value `127.0.0.1` is replaced by `localhost`, and the `http://` and `https://` schemes are replaced with `ws://` and `wss://` respectively.
- `DOTNET_WATCH_ITERATION`
`dotnet watch` sets this variable to `1` and increments by one each time a file is changed and the command restarts or hot reloads the application.
- `DOTNET_WATCH_SUPPRESS_BROWSER_REFRESH`
When set to `1` or `true`, `dotnet watch` won't refresh browsers when it detects file changes.

- `DOTNET_WATCH_SUPPRESS_EMOJIS`

With the .NET SDK 6.0.300 and later, `dotnet watch` emits non-ASCII characters to the console, as shown in the following example:

```
dotnet watch [ Hot reload enabled. For a list of supported edits, see https://aka.ms/dotnet/hot-reload.  
[ Press "Ctrl + R" to restart.  
dotnet watch [ Building...  
dotnet watch [ Started  
dotnet watch [ Exited  
dotnet watch [ Waiting for a file to change before restarting dotnet...]
```

On certain console hosts, these characters may appear garbled. To avoid seeing garbled characters, set this variable to `1` or `true`.

- `DOTNET_WATCH_SUPPRESS_LAUNCH_BROWSER`

When set to `1` or `true`, `dotnet watch` won't launch or refresh browsers for web apps that have `launchBrowser` configured in `launchSettings.json`.

- `DOTNET_WATCH_SUPPRESS_MSBUILD_INCREMENTALISM`

By default, `dotnet watch` optimizes the build by avoiding certain operations, such as running restore or re-evaluating the set of watched files on every file change. If this variable is set to `1` or `true`, these optimizations are disabled.

- `DOTNET_WATCH_SUPPRESS_STATIC_FILE_HANDLING`

When set to `1` or `true`, `dotnet watch` won't do special handling for static content files. `dotnet watch` sets MSBuild property `DotNetWatchContentFiles` to `false`.

Files watched by default

`dotnet watch` watches all items in the `Watch` item group in the project file. By default, this group includes all items in the `Compile` and `EmbeddedResource` groups. `dotnet watch` also scans the entire graph of project references and watches all files within those projects.

By default, the `Compile` and `EmbeddedResource` groups include all files matching the following glob patterns:

- `**/*.cs`
- `*.csproj`
- `**/*.resx`
- Content files in web apps: `wwwroot/**`

By default, `.config`, and `.json` files don't trigger a `dotnet watch` restart because the configuration system has its own mechanisms for handling configuration changes.

Files can be added to the watch list or removed from the list by editing the project file. Files can be specified individually or by using glob patterns.

Watch additional files

More files can be watched by adding items to the `Watch` group. For example, the following markup extends that group to include JavaScript files:

```
<ItemGroup>
  <Watch Include="**\*.js" Exclude="node_modules\**\*;**\*.js.map;obj\**\*;bin\**\*" />
</ItemGroup>
```

Ignore specified files

`dotnet watch` will ignore `Compile` and `EmbeddedResource` items that have the `Watch="false"` attribute, as shown in the following example:

```
<ItemGroup>
  <Compile Update="Generated.cs" Watch="false" />
  <EmbeddedResource Update="Strings.resx" Watch="false" />
</ItemGroup>
```

`dotnet watch` will ignore project references that have the `Watch="false"` attribute, as shown in the following example:

```
<ItemGroup>
  <ProjectReference Include="..\ClassLibrary1\ClassLibrary1.csproj" Watch="false" />
</ItemGroup>
```

Advanced configuration

`dotnet watch` performs a design-time build to find items to watch. When this build is run, `dotnet watch` sets the property `DotNetWatchBuild=true`. This property can be used as shown in the following example:

```
<ItemGroup Condition="'$(DotNetWatchBuild)'=='true'">
  <!-- only included in the project when dotnet-watch is running -->
</ItemGroup>
```

Hot Reload

Starting in .NET 6, `dotnet watch` includes support for *hot reload*. Hot reload is a feature that lets you apply changes to a running app without having to rebuild and restart it. The changes may be to code files or static assets, such as stylesheet files and JavaScript files. This feature streamlines the local development experience, as it gives immediate feedback when you modify your app.

For information about app types and .NET versions that support hot reload, see [Supported .NET app frameworks and scenarios](#).

Rude edits

When a file is modified, `dotnet watch` determines if the app can be hot reloaded. If it can't be hot reloaded, the change is called a *rude edit* and `dotnet watch` asks if you want to restart the app:

```
dotnet watch ↵ Unable to apply hot reload because of a rude edit.
 ↵ Do you want to restart your app - Yes (y) / No (n) / Always (a) / Never (v)?
```

- **Yes:** Restarts the app.
- **No:** Leaves the app running without the changes applied.
- **Always:** Restarts the app and doesn't prompt anymore for rude edits.
- **Never:** Leaves the app running without the changes applied and doesn't prompt anymore for rude edits.

For information about what kinds of changes are considered rude edits, see [Edit code and continue debugging](#) and [Unsupported changes to code](#).

To disable hot reload when you run `dotnet watch`, use the `--no-hot-reload` option, as shown in the following example:

```
dotnet watch --no-hot-reload
```

Examples

- Run `dotnet run` for the project in the current directory whenever source code changes:

```
dotnet watch
```

Or:

```
dotnet watch run
```

- Run `dotnet test` for the project in the current directory whenever source code changes:

```
dotnet watch test
```

- Run `dotnet run --project ./HelloWorld.csproj` whenever source code changes:

```
dotnet watch run --project ./HelloWorld.csproj
```

- Run `dotnet run -- arg0` for the project in the current directory whenever source code changes:

```
dotnet watch run -- arg0
```

Or:

```
dotnet watch -- run arg0
```

See also

- [Tutorial: Develop ASP.NET Core apps using a file watcher](#)
- [Hot reload in Visual Studio](#)
- [Hot reload supported apps](#)
- [Hot reload supported code changes](#)
- [Hot reload test execution](#)
- [Hot reload support for ASP.NET Core](#)

dotnet workload install

9/20/2022 • 3 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET 6 SDK and later versions

Name

`dotnet workload install` - Installs optional workloads.

Synopsis

```
dotnet workload install <WORKLOAD_ID>...
  [--configfile <FILE>] [--disable-parallel]
  [--ignore-failed-sources] [--include-previews] [--interactive]
  [--no-cache] [--skip-manifest-update]
  [--source <SOURCE>] [--temp-dir <PATH>] [-v|--verbosity <LEVEL>]

dotnet workload install -?|-h|--help
```

Description

The `dotnet workload install` command installs one or more *optional workloads*. Optional workloads can be installed on top of the .NET SDK to provide support for various application types, such as [.NET MAUI](#) and [Blazor WebAssembly AOT](#).

Use [dotnet workload search](#) to learn what workloads are available to install.

When to run elevated

For macOS and Linux SDK installations that are installed to a protected directory, the command needs to run elevated (use the `sudo` command). On Windows, the command doesn't need to run elevated even if the SDK is installed to the *Program Files* directory. For Windows, the command uses MSI installers for that location.

Results vary by SDK version

The `dotnet workload` commands operate in the context of specific SDK versions. Suppose you have both .NET 6.0.100 SDK and .NET 6.0.200 SDK installed. The `dotnet workload` commands will give different results depending on which SDK version you select. This behavior applies to major and minor version and feature band differences, not to patch version differences. For example, .NET SDK 6.0.101 and 6.0.102 give the same results, whereas 6.0.100 and 6.0.200 give different results. You can specify the SDK version by using the [`global.json` file](#) or the `--sdk-version` option of the `dotnet workload` commands.

Advertising manifests

The names and versions of the assets that a workload installation requires are maintained in *manifests*. By default, the `dotnet workload install` command downloads the latest available manifests before it installs a workload. The local copy of a manifest then provides the information needed to find and download the assets for a workload.

The `dotnet workload list` command compares the versions of installed workloads with the currently available versions. When it finds that a version newer than the installed version is available, it advertises that fact in the command output. These newer-version notifications in `dotnet workload list` are available starting in .NET 6.

To enable these notifications, the latest available versions of the manifests are downloaded and stored as

advertising manifests. These downloads happen asynchronously in the background when any of the following commands are run.

- `dotnet build`
- `dotnet pack`
- `dotnet publish`
- `dotnet restore`
- `dotnet run`
- `dotnet test`

If a command finishes before the manifest download finishes, the download is stopped. The download is tried again the next time one of these commands is run. You can set environment variables to [disable these background downloads](#) or [control their frequency](#). By default, they don't happen more than once a day.

You can prevent the `dotnet workload install` command from doing manifest downloads by using the `--skip-manifest-update` option.

The `dotnet workload update` command also downloads advertising manifests. The downloads are required to learn if an update is available, so there is no option to prevent them from running. However, you can use the `--advertising-manifests-only` option to skip workload updates and only do the manifest downloads. This option is available starting in .NET 6.

Arguments

- `WORKLOAD_ID` ...

The workload ID or multiple IDs to install. Use [dotnet workload search](#) to learn what workloads are available.

Options

- `--configfile <FILE>`

The NuGet configuration file (*nuget.config*) to use. If specified, only the settings from this file will be used. If not specified, the hierarchy of configuration files from the current directory will be used. For more information, see [Common NuGet Configurations](#).

- `--disable-parallel`

Prevents restoring multiple projects in parallel.

- `-?|-h|--help`

Prints out a description of how to use the command.

- `--ignore-failed-sources`

Treats package source failures as warnings.

- `--include-previews`

Allows prerelease workload manifests.

- `--interactive`

Allows the command to stop and wait for user input or action. For example, to complete authentication.

- `--no-cache`

Prevents caching of packages and http requests.

- `--skip-manifest-update`

Skip updating the workload manifests. The workload manifests define what assets and versions need to be installed for each workload.

- `-s|--source <SOURCE>`

Specifies the URI of the NuGet package source to use. This setting overrides all of the sources specified in the *nuget.config* files. Multiple sources can be provided by specifying this option multiple times.

- `--temp-dir <PATH>`

Specify the temporary directory used to download and extract NuGet packages (must be secure).

- `-v|--verbosity <LEVEL>`

Sets the verbosity level of the command. Allowed values are `q[uiet]`, `m[inimal]`, `n[ormal]`, `d[etailed]`, and `diag[nostic]`. The default is `minimal`. If you specify `detailed` or `diagnostic` verbosity, the command displays information about the Nuget packages that it downloads.

Examples

- Install the `maui` workload:

```
dotnet workload install maui
```

- Install the `maui-android` and `maui-ios` workloads:

```
dotnet workload install maui-android maui-ios
```

dotnet workload list

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET 6 SDK and later versions

Name

`dotnet workload list` - Lists installed workloads.

Synopsis

```
dotnet workload list [-v|--verbosity <LEVEL>]  
dotnet workload list [-?|-h|--help]
```

Description

The `dotnet workload list` command lists all installed workloads.

For more information about the `dotnet workload` commands, see the [dotnet workload install](#) command.

Options

- `-?|-h|--help`

Prints out a description of how to use the command.

- `-v|--verbosity <LEVEL>`

Sets the verbosity level of the command. Allowed values are `q[uiet]`, `m[inimal]`, `n[ormal]`, `d[etailed]`, and `diag[nostic]`. The default is `minimal`. For more information, see [LoggerVerbosity](#).

Examples

- List the installed workloads:

```
dotnet workload list
```

dotnet workload repair

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET 6 SDK and later versions

Name

`dotnet workload repair` - Repairs workloads installations.

Synopsis

```
dotnet workload repair  
  [--configfile] [--disable-parallel] [--ignore-failed-sources]  
  [--interactive] [--no-cache]  
  [-s|--source <SOURCE>] [--temp-dir <PATH>]  
  [-v|--verbosity <LEVEL>]  
  
dotnet workload repair -?|-h|--help
```

Description

The `dotnet workload repair` command reinstalls all installed workloads. Workloads are made up of multiple workload packs and it's possible to get into a state where some installed successfully but others didn't. For example, a `dotnet workload install` command might not finish installing because of a dropped internet connection.

For more information about the `dotnet workload` commands, see the [dotnet workload install](#) command.

Options

- `--configfile <FILE>`

The NuGet configuration file (*nuget.config*) to use. If specified, only the settings from this file will be used. If not specified, the hierarchy of configuration files from the current directory will be used. For more information, see [Common NuGet Configurations](#).

- `--disable-parallel`

Prevents restoring multiple projects in parallel.

- `-?|-h|--help`

Prints out a description of how to use the command.

- `--ignore-failed-sources`

Treats package source failures as warnings.

- `--include-previews`

Allows prerelease workload manifests.

- `--interactive`

Allows the command to stop and wait for user input or action. For example, to complete authentication.

- `--no-cache`

Prevents caching of packages and http requests.

- `-s|--source <SOURCE>`

Specifies the URI of the NuGet package source to use. This setting overrides all of the sources specified in the *nuget.config* files. Multiple sources can be provided by specifying this option multiple times.

- `--temp-dir <PATH>`

Specify the temporary directory used to download and extract NuGet packages (must be secure).

- `-v|--verbosity <LEVEL>`

Sets the verbosity level of the command. Allowed values are `q[uiet]`, `m[inimal]`, `n[ormal]`, `d[etailed]`, and `diag[nostic]`. The default is `minimal`. For more information, see [LoggerVerbosity](#).

Examples

- Repair all installed workloads:

```
dotnet workload repair
```

dotnet workload restore

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET 6 SDK and later versions

Name

`dotnet workload restore` - Installs workloads needed for a project or a solution.

Synopsis

```
dotnet workload restore [<PROJECT | SOLUTION>]
  [--configfile <FILE>] [--disable-parallel]
  [--ignore-failed-sources] [--include-previews] [--interactive]
  [--no-cache] [--skip-manifest-update]
  [-s|--source <SOURCE>] [--temp-dir <PATH>] [-v|--verbosity <LEVEL>]

dotnet workload restore -?|-h|--help
```

Description

The `dotnet workload restore` command analyzes a project or solution to determine which workloads it needs, then installs any workloads that are missing.

For more information about the `dotnet workload` commands, see the [dotnet workload install](#) command.

Arguments

- `PROJECT | SOLUTION`

The project or solution file to install workloads for. If a file is not specified, the command searches the current directory for one.

Options

- `--configfile <FILE>`

The NuGet configuration file (*nuget.config*) to use. If specified, only the settings from this file will be used. If not specified, the hierarchy of configuration files from the current directory will be used. For more information, see [Common NuGet Configurations](#).

- `--disable-parallel`

Prevents restoring multiple projects in parallel.

- `-?|-h|--help`

Prints out a description of how to use the command.

- `--ignore-failed-sources`

Treats package source failures as warnings.

- `--include-previews`

Allows prerelease workload manifests.
- `--interactive`

Allows the command to stop and wait for user input or action. For example, to complete authentication.
- `--no-cache`

Prevents caching of packages and http requests.
- `--skip-manifest-update`

Skip updating the workload manifests. The workload manifests define what assets and versions need to be installed for each workload.
- `-s|--source <SOURCE>`

Specifies the URI of the NuGet package source to use. This setting overrides all of the sources specified in the *nuget.config* files. Multiple sources can be provided by specifying this option multiple times.
- `--temp-dir <PATH>`

Specify the temporary directory used to download and extract NuGet packages (must be secure).
- `-v|--verbosity <LEVEL>`

Sets the verbosity level of the command. Allowed values are `q[uiet]`, `m[inimal]`, `n[ormal]`, `d[etailed]`, and `diag[nostic]`. The default is `minimal`. For more information, see [LoggerVerbosity](#).

Example

- Restore workloads needed by `MyApp.csproj`:

```
dotnet workload restore MyApp.csproj
```

dotnet workload search

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET 6 SDK and later versions

Name

`dotnet workload search` - Searches for optional workloads.

Synopsis

```
dotnet workload search [<SEARCH_STRING>] [-v|--verbosity <LEVEL>]
```

```
dotnet workload search -?|-h|--help
```

Description

The `dotnet workload search` command lists available workloads. You can filter the list by specifying all or part of the workload ID you're looking for.

For more information about the `dotnet workload` commands, see the [dotnet workload install](#) command.

Arguments

- `<SEARCH_STRING>`

The workload ID to search for, or part of it. For example, if you specify `maui`, the command lists all of the workload IDs that have `maui` in their workload ID.

Options

- `-?|-h|--help`

Prints out a description of how to use the command.

- `-v|--verbosity <LEVEL>`

Sets the verbosity level of the command. Allowed values are `q[uiet]`, `m[inimal]`, `n[ormal]`, `d[etailed]`, and `diag[nostic]`. The default is `minimal`. For more information, see [LoggerVerbosity](#).

Examples

- List all available workloads:

```
dotnet workload search
```

- List all available workloads that have "maui" in their workload ID:

```
dotnet workload search maui
```

dotnet workload uninstall

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET 6 SDK and later versions

Name

`dotnet workload uninstall` - Uninstalls a specified workload.

Synopsis

```
dotnet workload uninstall <WORKLOAD_ID...>
```

```
dotnet workload uninstall -?|-h|--help
```

Description

The `dotnet workload uninstall` command uninstalls one or more workloads.

For more information about the `dotnet workload` commands, see the [dotnet workload install](#) command.

Arguments

- `WORKLOAD_ID...`

The workload ID or multiple IDs to uninstall.

Options

- `-?|-h|--help`

Prints out a description of how to use the command.

Examples

- Uninstall the `maui` workload:

```
dotnet workload uninstall maui
```

- Uninstall the `maui-android` and `maui-ios` workloads:

```
dotnet workload uninstall maui-android maui-ios
```

dotnet workload update

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET 6 SDK and later versions

Name

`dotnet workload update` - Updates installed workloads.

Synopsis

```
dotnet workload update
  [--advertising-manifests-only]
  [--configfile <FILE>] [--disable-parallel]
  [--from-previous-sdk] [--ignore-failed-sources]
  [--include-previews] [--interactive] [--no-cache]
  [-s|--source <SOURCE>] [--temp-dir <PATH>]
  [-v|--verbosity <LEVEL>]

dotnet workload update -?|-h|--help
```

Description

The `dotnet workload update` command updates all installed workloads to the newest available versions. It queries Nuget.org for updated workload manifests. It then updates local manifests, downloads new versions of the installed workloads, and removes all old versions of each workload.

For more information about the `dotnet workload` commands, see the [dotnet workload install](#) command.

Options

- `--advertising-manifests-only`

Downloads [advertising manifests](#) but doesn't update any workloads.

- `--configfile <FILE>`

The NuGet configuration file (*nuget.config*) to use. If specified, only the settings from this file will be used. If not specified, the hierarchy of configuration files from the current directory will be used. For more information, see [Common NuGet Configurations](#).

- `--disable-parallel`

Prevents restoring multiple projects in parallel.

- `--from-previous-sdk`

Include workloads installed with previous SDK versions in the update.

- `-?|-h|--help`

Prints out a description of how to use the command.

- `--ignore-failed-sources`

Treats package source failures as warnings.

- `--include-previews`

Allows prerelease workload manifests.

- `--interactive`

Allows the command to stop and wait for user input or action. For example, to complete authentication.

- `--no-cache`

Prevents caching of packages and http requests.

- `-s|--source <SOURCE>`

Specifies the URI of the NuGet package source to use. This setting overrides all of the sources specified in the *nuget.config* files. Multiple sources can be provided by specifying this option multiple times.

- `--temp-dir <PATH>`

Specify the temporary directory used to download and extract NuGet packages (must be secure).

- `-v|--verbosity <LEVEL>`

Sets the verbosity level of the command. Allowed values are `q[uiet]`, `m[inimal]`, `n[ormal]`, `d[etailed]`, and `diag[nostic]`. The default is `minimal`. If you specify `detailed` or `diagnostic` verbosity, the command displays information about the Nuget packages that it downloads.

Examples

- Update the installed workloads:

```
dotnet workload update
```

Elevated access for dotnet commands

9/20/2022 • 4 minutes to read • [Edit Online](#)

Software development best practices guide developers to writing software that requires the least amount of privilege. However, some software, like performance monitoring tools, requires admin permission because of operating system rules. The following guidance describes supported scenarios for writing such software with .NET Core.

The following commands can be run elevated:

- `dotnet tool` commands, such as [dotnet tool install](#).
- `dotnet run --no-build`
- `dotnet-core-uninstall`

We don't recommend running other commands elevated. In particular, we don't recommend elevation with commands that use MSBuild, such as [dotnet restore](#), [dotnet build](#), and [dotnet run](#). The primary issue is permission management problems when a user transitions back and forth between root and a restricted account after issuing dotnet commands. You may find as a restricted user that you don't have access to the file built by a root user. There are ways to resolve this situation, but they're unnecessary to get into in the first place.

You can run commands as root as long as you don't transition back and forth between root and a restricted account. For example, Docker containers run as root by default, so they have this characteristic.

Global tool installation

The following instructions demonstrate the recommended way to install, run, and uninstall .NET tools that require elevated permissions to execute.

- [Windows](#)
- [Linux](#)
- [macOS](#)

Install the tool

If the folder `%ProgramFiles%\dotnet-tools` already exists, do the following to check whether the "Users" group has permission to write or modify that directory:

- Right-click the `%ProgramFiles%\dotnet-tools` folder and select **Properties**. The **Common Properties** dialog box opens.
- Select the **Security** tab. Under **Group or user names**, check whether the "Users" group has permission to write or modify the directory.
- If the "Users" group can write or modify the directory, use a different directory name when installing the tools rather than *dotnet-tools*.

To install tools, run the following command in elevated prompt. It will create the *dotnet-tools* folder during the installation.

```
dotnet tool install PACKAGEID --tool-path "%ProgramFiles%\dotnet-tools".
```

Run the global tool

Option 1 Use the full path with elevated prompt:

```
"%ProgramFiles%\dotnet-tools\TOOLCOMMAND"
```

Option 2 Add the newly created folder to `%Path%`. You only need to do this operation once.

```
setx Path "%Path%;%ProgramFiles%\dotnet-tools\"
```

And run with:

```
TOOLCOMMAND
```

Uninstall the global tool

In an elevated prompt, type the following command:

```
dotnet tool uninstall PACKAGEID --tool-path "%ProgramFiles%\dotnet-tools"
```

Local tools

Local tools are scoped per subdirectory tree, per user. When run elevated, local tools share a restricted user environment to the elevated environment. In Linux and macOS, this results in files being set with root user-only access. If the user switches back to a restricted account, the user can no longer access or write to the files. So installing tools that require elevation as local tools isn't recommended. Instead, use the `--tool-path` option and the previous guidelines for global tools.

Elevation during development

During development, you may need elevated access to test your application. This scenario is common for IoT apps, for example. We recommend that you build the application without elevation and then run it with elevation. There are a few patterns, as follows:

- Using generated executable (it provides the best startup performance):

```
dotnet build
sudo ./bin/Debug/netcoreapp3.0/APPLICATIONNAME
```

- Using the `dotnet run` command with the `--no-build` flag to avoid generating new binaries:

```
dotnet build
sudo dotnet run --no-build
```

See also

- [.NET tools overview](#)

How to enable TAB completion for the .NET CLI

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 2.1 SDK and later versions

This article describes how to configure tab completion for four shells, PowerShell, Bash, zsh, and fish. For other shells, refer to their documentation on how to configure tab completion.

Once set up, tab completion for the .NET CLI is triggered by typing a `dotnet` command in the shell, and then pressing the TAB key. The current command line is sent to the `dotnet complete` command, and the results are processed by your shell. You can test the results without enabling tab completion by sending something directly to the `dotnet complete` command. For example:

```
> dotnet complete "dotnet a"
add
clean
--diagnostics
migrate
pack
```

If that command doesn't work, make sure that .NET Core 2.0 SDK or above is installed. If it's installed, but that command still doesn't work, make sure that the `dotnet` command resolves to a version of .NET Core 2.0 SDK and above. Use the `dotnet --version` command to see what version of `dotnet` your current path is resolving to. For more information, see [Select the .NET version to use](#).

Examples

Here are some examples of what tab completion provides:

INPUT	BECOMES	BECAUSE
<code>dotnet a*</code>	<code>dotnet add</code>	<code>add</code> is the first subcommand, alphabetically.
<code>dotnet add p*</code>	<code>dotnet add --help</code>	Tab completion matches substrings and <code>--help</code> comes first alphabetically.
<code>dotnet add p**</code>	<code>dotnet add package</code>	Pressing tab a second time brings up the next suggestion.
<code>dotnet add package Microsoft*</code>	<code>dotnet add package Microsoft.ApplicationInsights.Web</code>	Results are returned alphabetically.
<code>dotnet remove reference *</code>	<code>dotnet remove reference ..\src\OmniSharp.DotNet\OmniSharp.DotNet.csproj</code>	Tab completion is project file aware.

PowerShell

To add tab completion to PowerShell for the .NET CLI, create or edit the profile stored in the variable `$PROFILE`. For more information, see [How to create your profile](#) and [Profiles and execution policy](#).

Add the following code to your profile:

```
# PowerShell parameter completion shim for the dotnet CLI
Register-ArgumentCompleter -Native -CommandName dotnet -ScriptBlock {
    param($commandName, $wordToComplete, $cursorPosition)
        dotnet complete --position $cursorPosition "$wordToComplete" | ForEach-Object {
            [System.Management.Automation.CompletionResult]::new($_, $_, 'ParameterValue', $_)
        }
}
```

bash

To add tab completion to your **bash** shell for the .NET CLI, add the following code to your `.bashrc` file:

```
# bash parameter completion for the dotnet CLI

function _dotnet_bash_complete()
{
    local cur="${COMP_WORDS[COMP_CWORD]}" IFS=$'\n'
    local candidates

    read -d '' -ra candidates < <(dotnet complete --position "${COMP_POINT}" "${COMP_LINE}" 2>/dev/null)

    read -d '' -ra COMPREPLY < <(compgen -W "${candidates[*]}:-" -- "$cur")
}

complete -f -F _dotnet_bash_complete dotnet
```

zsh

To add tab completion to your **zsh** shell for the .NET CLI, add the following code to your `.zshrc` file:

```
# zsh parameter completion for the dotnet CLI

_dotnet_zsh_complete()
{
    local completions=$(dotnet complete "$words")

    reply=( "${(ps:\n:)completions}" )
}

compctl -K _dotnet_zsh_complete dotnet
```

fish

To add tab completion to your **fish** shell for the .NET CLI, add the following code to your `config.fish` file:

```
complete -f -c dotnet -a "(dotnet complete (commandline -cp))"
```

Develop libraries with the .NET CLI

9/20/2022 • 9 minutes to read • [Edit Online](#)

This article covers how to write libraries for .NET using the .NET CLI. The CLI provides an efficient and low-level experience that works across any supported OS. You can still build libraries with Visual Studio, and if that is your preferred experience [refer to the Visual Studio guide](#).

Prerequisites

You need the [.NET SDK](#) installed on your machine.

For the sections of this document dealing with .NET Framework versions, you need the [.NET Framework](#) installed on a Windows machine.

Additionally, if you wish to support older .NET Framework targets, you need to install targeting packs or developer packs from the [.NET Framework downloads page](#). Refer to this table:

.NET FRAMEWORK VERSION	WHAT TO DOWNLOAD
4.6.1	.NET Framework 4.6.1 Targeting Pack
4.6	.NET Framework 4.6 Targeting Pack
4.5.2	.NET Framework 4.5.2 Developer Pack
4.5.1	.NET Framework 4.5.1 Developer Pack
4.5	Windows Software Development Kit for Windows 8
4.0	Windows SDK for Windows 7 and .NET Framework 4
2.0, 3.0, and 3.5	.NET Framework 3.5 SP1 Runtime (or Windows 8+ version)

How to target .NET 5+ or .NET Standard

You control your project's target framework by adding it to your project file (`.csproj` or `.fsproj`). For guidance on how to choose between targeting .NET 5+ or .NET Standard see [.NET 5+ and .NET Standard](#).

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <TargetFramework>net6.0</TargetFramework>
  </PropertyGroup>
</Project>
```

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <TargetFramework>netstandard2.0</TargetFramework>
  </PropertyGroup>
</Project>
```

If you want to target .NET Framework versions 4.0 or below, or you wish to use an API available in .NET Framework but not in .NET Standard (for example, `System.Drawing`), read the following sections and learn how to multitarget.

How to target .NET Framework

NOTE

These instructions assume you have .NET Framework installed on your machine. Refer to the [Prerequisites](#) to get dependencies installed.

Keep in mind that some of the .NET Framework versions used here are no longer supported. Refer to the [.NET Framework Support Lifecycle Policy FAQ](#) about unsupported versions.

If you want to reach the maximum number of developers and projects, use .NET Framework 4.0 as your baseline target. To target .NET Framework, begin by using the correct Target Framework Moniker (TFM) that corresponds to the .NET Framework version you wish to support.

.NET FRAMEWORK VERSION	TFM
.NET Framework 2.0	<code>net20</code>
.NET Framework 3.0	<code>net30</code>
.NET Framework 3.5	<code>net35</code>
.NET Framework 4.0	<code>net40</code>
.NET Framework 4.5	<code>net45</code>
.NET Framework 4.5.1	<code>net451</code>
.NET Framework 4.5.2	<code>net452</code>
.NET Framework 4.6	<code>net46</code>
.NET Framework 4.6.1	<code>net461</code>
.NET Framework 4.6.2	<code>net462</code>
.NET Framework 4.7	<code>net47</code>
.NET Framework 4.8	<code>net48</code>

You then insert this TFM into the `TargetFramework` section of your project file. For example, here's how you would write a library that targets .NET Framework 4.0:

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <TargetFramework>net40</TargetFramework>
  </PropertyGroup>
</Project>
```

And that's it! Although this compiled only for .NET Framework 4, you can use the library on newer versions of .NET Framework.

How to multitarget

NOTE

The following instructions assume you have the .NET Framework installed on your machine. Refer to the [Prerequisites](#) section to learn which dependencies you need to install and where to download them from.

You may need to target older versions of the .NET Framework when your project supports both the .NET Framework and .NET. In this scenario, if you want to use newer APIs and language constructs for the newer targets, use `#if` directives in your code. You also might need to add different packages and dependencies for each platform you're targeting to include the different APIs needed for each case.

For example, let's say you have a library that performs networking operations over HTTP. For .NET Standard and the .NET Framework versions 4.5 or higher, you can use the `HttpClient` class from the `System.Net.Http` namespace. However, earlier versions of the .NET Framework don't have the `HttpClient` class, so you could use the `WebClient` class from the `System.Net` namespace for those instead.

Your project file could look like this:

```
<Project Sdk="Microsoft.NET.Sdk">
<PropertyGroup>
    <TargetFrameworks>netstandard2.0;net40;net45</TargetFrameworks>
</PropertyGroup>

<!-- Need to conditionally bring in references for the .NET Framework 4.0 target -->
<ItemGroup Condition="<$(TargetFramework)> == 'net40'">
    <Reference Include="System.Net" />
</ItemGroup>

<!-- Need to conditionally bring in references for the .NET Framework 4.5 target -->
<ItemGroup Condition="<$(TargetFramework)> == 'net45'">
    <Reference Include="System.Net.Http" />
    <Reference Include="System.Threading.Tasks" />
</ItemGroup>
</Project>
```

You'll notice three major changes here:

1. The `TargetFramework` node has been replaced by `TargetFrameworks`, and three TFM's are expressed inside.
2. There is an `<ItemGroup>` node for the `net40` target pulling in one .NET Framework reference.
3. There is an `<ItemGroup>` node for the `net45` target pulling in two .NET Framework references.

Preprocessor Symbols

The build system is aware of the following preprocessor symbols used in `#if` directives:

TARGET FRAMEWORKS

SYMBOLS

ADDITIONAL SYMBOLS AVAILABLE IN
.NET 5+ SDK

TARGET FRAMEWORKS	SYMBOLS	ADDITIONAL SYMBOLS AVAILABLE IN .NET 5+ SDK
.NET Framework	NETFRAMEWORK , NET48 , NET472 , NET471 , NET47 , NET462 , NET461 , NET46 , NET452 , NET451 , NET45 , NET40 , NET35 , NET20	NET48_OR_GREATER , NET472_OR_GREATER , NET471_OR_GREATER , NET47_OR_GREATER , NET462_OR_GREATER , NET461_OR_GREATER , NET46_OR_GREATER , NET452_OR_GREATER , NET451_OR_GREATER , NET45_OR_GREATER , NET40_OR_GREATER , NET35_OR_GREATER , NET20_OR_GREATER
.NET Standard	NETSTANDARD , NETSTANDARD2_1 , NETSTANDARD2_0 , NETSTANDARD1_6 , NETSTANDARD1_5 , NETSTANDARD1_4 , NETSTANDARD1_3 , NETSTANDARD1_2 , NETSTANDARD1_1 , NETSTANDARD1_0	NETSTANDARD2_1_OR_GREATER , NETSTANDARD2_0_OR_GREATER , NETSTANDARD1_6_OR_GREATER , NETSTANDARD1_5_OR_GREATER , NETSTANDARD1_4_OR_GREATER , NETSTANDARD1_3_OR_GREATER , NETSTANDARD1_2_OR_GREATER , NETSTANDARD1_1_OR_GREATER , NETSTANDARD1_0_OR_GREATER
.NET 5+ (and .NET Core)	NET , NET7_0 , NET6_0 , NET5_0 , NETCOREAPP , NETCOREAPP3_1 , NETCOREAPP3_0 , NETCOREAPP2_2 , NETCOREAPP2_1 , NETCOREAPP2_0 , NETCOREAPP1_1 , NETCOREAPP1_0	NET7_0_OR_GREATER , NET6_0_OR_GREATER , NET5_0_OR_GREATER , NETCOREAPP3_1_OR_GREATER , NETCOREAPP3_0_OR_GREATER , NETCOREAPP2_2_OR_GREATER , NETCOREAPP2_1_OR_GREATER , NETCOREAPP2_0_OR_GREATER , NETCOREAPP1_1_OR_GREATER , NETCOREAPP1_0_OR_GREATER

NOTE

- Versionless symbols are defined regardless of the version you're targeting.
- Version-specific symbols are only defined for the version you're targeting.
- The `<framework>_OR_GREATER` symbols are defined for the version you're targeting and all earlier versions. For example, if you're targeting .NET Framework 2.0, the following symbols are defined: `NET20` , `NET20_OR_GREATER` , `NET11_OR_GREATER` , and `NET10_OR_GREATER` .
- These are different from the target framework monikers (TFMs) used by the MSBuild `TargetFramework` property and NuGet.

Here is an example making use of conditional compilation per-target:

```

using System;
using System.Text.RegularExpressions;
#ifndef NET40
// This only compiles for the .NET Framework 4 targets
using System.Net;
#else
// This compiles for all other targets
using System.Net.Http;
using System.Threading.Tasks;
#endif

namespace MultitargetLib
{
    public class Library
    {
#ifndef NET40
        private readonly WebClient _client = new WebClient();
        private readonly object _locker = new object();
#else
        private readonly HttpClient _client = new HttpClient();
#endif

#ifndef NET40
        // .NET Framework 4.0 does not have async/await
        public string GetDotNetCount()
        {
            string url = "https://www.dotnetfoundation.org/";

            var uri = new Uri(url);

            string result = "";

            // Lock here to provide thread-safety.
            lock(_locker)
            {
                result = _client.DownloadString(uri);
            }

            int dotNetCount = Regex.Matches(result, ".NET").Count;

            return $"Dotnet Foundation mentions .NET {dotNetCount} times!";
        }
#else
        // .NET Framework 4.5+ can use async/await!
        public async Task<string> GetDotNetCountAsync()
        {
            string url = "https://www.dotnetfoundation.org/";

            // HttpClient is thread-safe, so no need to explicitly lock here
            var result = await _client.GetStringAsync(url);

            int dotNetCount = Regex.Matches(result, ".NET").Count;

            return $"dotnetfoundation.org mentions .NET {dotNetCount} times in its HTML!";
        }
#endif
    }
}

```

If you build this project with `dotnet build`, you'll notice three directories under the `bin/` folder:

```

net40/
net45/
netstandard2.0/

```

Each of these contains the `.dll` files for each target.

How to test libraries on .NET

It's important to be able to test across platforms. You can use either [xUnit](#) or MSTest out of the box. Both are perfectly suitable for unit testing your library on .NET. How you set up your solution with test projects will depend on the [structure of your solution](#). The following example assumes that the test and source directories live in the same top-level directory.

NOTE

This uses some [.NET CLI](#) commands. See [dotnet new](#) and [dotnet sln](#) for more information.

1. Set up your solution. You can do so with the following commands:

```
mkdir SolutionWithSrcAndTest
cd SolutionWithSrcAndTest
dotnet new sln
dotnet new classlib -o MyProject
dotnet new xunit -o MyProject.Test
dotnet sln add MyProject/MyProject.csproj
dotnet sln add MyProject.Test/MyProject.Test.csproj
```

This will create projects and link them together in a solution. Your directory for `SolutionWithSrcAndTest` should look like this:

```
/SolutionWithSrcAndTest
|__SolutionWithSrcAndTest.sln
|__MyProject/
|__MyProject.Test/
```

2. Navigate to the test project's directory and add a reference to `MyProject.Test` from `MyProject`.

```
cd MyProject.Test
dotnet add reference ../MyProject/MyProject.csproj
```

3. Restore packages and build projects:

```
dotnet restore
dotnet build
```

4. Verify that xUnit runs by executing the `dotnet test` command. If you chose to use MSTest, then the MSTest console runner should run instead.

And that's it! You can now test your library across all platforms using command-line tools. To continue testing now that you have everything set up, testing your library is very simple:

1. Make changes to your library.
2. Run tests from the command line, in your test directory, with `dotnet test` command.

Your code will be automatically rebuilt when you invoke `dotnet test` command.

How to use multiple projects

A common need for larger libraries is to place functionality in different projects.

Imagine you want to build a library that could be consumed in idiomatic C# and F#. That would mean that consumers of your library consume it in ways that are natural to C# or F#. For example, in C# you might consume the library like this:

```
using AwesomeLibrary.CSharp;

public Task DoThings(Data data)
{
    var convertResult = await AwesomeLibrary.ConvertAsync(data);
    var result = AwesomeLibrary.Process(convertResult);
    // do something with result
}
```

In F#, it might look like this:

```
open AwesomeLibrary.FSharp

let doWork data = async {
    let! result = AwesomeLibrary.AsyncConvert data // Uses an F# async function rather than C# async method
    // do something with result
}
```

Consumption scenarios like this mean that the APIs being accessed have to have a different structure for C# and F#. A common approach to accomplishing this is to factor all of the logic of a library into a core project, with C# and F# projects defining the API layers that call into that core project. The rest of the section will use the following names:

- **AwesomeLibrary.Core** - A core project that contains all logic for the library
- **AwesomeLibrary.CSharp** - A project with public APIs intended for consumption in C#
- **AwesomeLibrary.FSharp** - A project with public APIs intended for consumption in F#

You can run the following commands in your terminal to produce the same structure as this guide:

```
mkdir AwesomeLibrary && cd AwesomeLibrary
dotnet new sln
mkdir AwesomeLibrary.Core && cd AwesomeLibrary.Core && dotnet new classlib
cd ..
mkdir AwesomeLibrary.CSharp && cd AwesomeLibrary.CSharp && dotnet new classlib
cd ..
mkdir AwesomeLibrary.FSharp && cd AwesomeLibrary.FSharp && dotnet new classlib -lang "F#"
cd ..
dotnet sln add AwesomeLibrary.Core/AwesomeLibrary.Core.csproj
dotnet sln add AwesomeLibrary.CSharp/AwesomeLibrary.CSharp.csproj
dotnet sln add AwesomeLibrary.FSharp/AwesomeLibrary.FSharp.fsproj
```

This will add the three projects above and a solution file that links them together. Creating the solution file and linking projects will allow you to restore and build projects from a top level.

Project-to-project referencing

The best way to reference a project is to use the .NET CLI to add a project reference. From the **AwesomeLibrary.CSharp** and **AwesomeLibrary.FSharp** project directories, you can run the following command:

```
dotnet add reference ../AwesomeLibrary.Core/AwesomeLibrary.Core.csproj
```

The project files for both `AwesomeLibrary.CSharp` and `AwesomeLibrary.FSharp` will now reference `AwesomeLibrary.Core` as a `ProjectReference` target. You can verify this by inspecting the project files and seeing the following in them:

```
<ItemGroup>
  <ProjectReference Include="..\AwesomeLibrary.Core\AwesomeLibrary.Core.csproj" />
</ItemGroup>
```

You can add this section to each project file manually if you prefer not to use the .NET CLI.

Structuring a solution

Another important aspect of multi-project solutions is establishing a good overall project structure. You can organize code however you like, and as long as you link each project to your solution file with `dotnet sln add`, you will be able to run `dotnet restore` and `dotnet build` at the solution level.

Tutorial: Create an item template

9/20/2022 • 5 minutes to read • [Edit Online](#)

With .NET, you can create and deploy templates that generate projects, files, even resources. This tutorial is part one of a series that teaches you how to create, install, and uninstall templates for use with the `dotnet new` command.

You can view the completed template in the [.NET Samples GitHub repository](#).

In this part of the series, you'll learn how to:

- Create a class for an item template
- Create the template config folder and file
- Install a template from a file path
- Test an item template
- Uninstall an item template

Prerequisites

- [.NET 6.0 SDK](#) or a later version.
- Read the reference article [Custom templates for dotnet new](#).

The reference article explains the basics about templates and how they're put together. Some of this information will be reiterated here.

- Open a terminal and navigate to the `working\templates` folder.

Create the required folders

This series uses a "working folder" where your template source is contained and a "testing folder" used to test your templates. The working folder and testing folder should be under the same parent folder.

First, create the parent folder, the name does not matter. Then, create a subfolder named `working`. Inside of the `working` folder, create a subfolder named `templates`.

Next, create a folder under the parent folder named `test`. The folder structure should look like the following.

```
parent_folder
├── test
└── working
    └── templates
```

Create an item template

An item template is a specific type of template that contains one or more files. These types of templates are useful when you want to generate something like a config, code, or solution file. In this example, you'll create a class that adds an extension method to the string type.

In your terminal, navigate to the `working\templates` folder and create a new subfolder named `extensions`. Enter the folder.

```
working
└─templates
    └─extensions
```

Create a new file named *CommonExtensions.cs* and open it with your favorite text editor. This class will provide an extension method named `Reverse` that reverses the contents of a string. Paste in the following code and save the file:

```
using System;

namespace System
{
    public static class StringExtensions
    {
        public static string Reverse(this string value)
        {
            var tempArray = value.ToCharArray();
            Array.Reverse(tempArray);
            return new string(tempArray);
        }
    }
}
```

Now that you have the content of the template created, you need to create the template config at the root folder of the template.

Create the template config

Templates are recognized by a special folder and config file that exist at the root of your template. In this tutorial, your template folder is located at *working\templates\extensions*.

When you create a template, all files and folders in the template folder are included as part of the template except for the special config folder. This config folder is named *.template.config*.

First, create a new subfolder named *.template.config*, enter it. Then, create a new file named *template.json*. Your folder structure should look like this:

```
working
└─templates
    └─extensions
        └─.template.config
            template.json
```

Open the *template.json* with your favorite text editor and paste in the following JSON code and save it.

```
{
    "$schema": "http://json.schemastore.org/template",
    "author": "Me",
    "classifications": [ "Common", "Code" ],
    "identity": "ExampleTemplate.StringExtensions",
    "name": "Example templates: string extensions",
    "shortName": "stringext",
    "tags": {
        "language": "C#",
        "type": "item"
    }
}
```

This config file contains all the settings for your template. You can see the basic settings, such as `name` and `shortName`, but there's also a `tags/type` value that is set to `item`. This categorizes your template as an item template. There's no restriction on the type of template you create. The `item` and `project` values are common names that .NET recommends so that users can easily filter the type of template they're searching for.

The `classifications` item represents the `tags` column you see when you run `dotnet new` and get a list of templates. Users can also search based on classification tags. Don't confuse the `tags` property in the `*.json` file with the `classifications` tags list. They're two different things unfortunately named similarly. The full schema for the `template.json` file is found at the [JSON Schema Store](#). For more information about the `template.json` file, see the [dotnet templating wiki](#).

Now that you have a valid `.template.config/template.json` file, your template is ready to be installed. In your terminal, navigate to the `extensions` folder and run the following command to install the template located at the current folder:

- **On Windows:** `dotnet new --install .\`
- **On Linux or macOS:** `dotnet new --install ./`

This command outputs the list of templates installed, which should include yours.

```
The following template packages will be installed:  
<root path>\working\templates\extensions  
  
Success: <root path>\working\templates\extensions installed the following templates:  
Templates Short Name Language Tags  
----- ----- ----- -----  
-----  
Example templates: string extensions stringext [C#] Common/Code
```

Test the item template

Now that you have an item template installed, test it. Navigate to the `test`/folder and create a new console application with `dotnet new console`. This generates a working project you can easily test with the `dotnet run` command.

```
dotnet new console
```

You get output similar to the following.

```
The template "Console Application" was created successfully.  
  
Processing post-creation actions...  
Running 'dotnet restore' on C:\test\test.csproj...  
Restore completed in 54.82 ms for C:\test\test.csproj.  
  
Restore succeeded.
```

Run the project with.

```
dotnet run
```

You get the following output.

```
Hello World!
```

Next, run `dotnet new stringext` to generate the `CommonExtensions.cs` from the template.

```
dotnet new stringext
```

You get the following output.

```
The template "Example templates: string extensions" was created successfully.
```

Change the code in `Program.cs` to reverse the `"Hello World"` string with the extension method provided by the template.

```
Console.WriteLine("Hello World!".Reverse());
```

Run the program again and you'll see that the result is reversed.

```
dotnet run
```

You get the following output.

```
!dlroW olleH
```

Congratulations! You created and deployed an item template with .NET. In preparation for the next part of this tutorial series, you must uninstall the template you created. Make sure to delete all files from the `test` folder too. This will get you back to a clean state ready for the next major section of this tutorial.

Uninstall the template

In your terminal, navigate to the `extensions` folder and run the following command to uninstall the template located at the current folder:

- On Windows: `dotnet new --uninstall .\`
- On Linux or macOS: `dotnet new --uninstall ./`

This command outputs a list of the templates that were uninstalled, which should include yours.

```
Success: <root path>\working\templates\extensions was uninstalled.
```

At any time, you can use `dotnet new --uninstall` to see a list of installed template packages, including for each template package the command to uninstall it.

Next steps

In this tutorial, you created an item template. To learn how to create a project template, continue this tutorial series.

[Create a project template](#)

Tutorial: Create a project template

9/20/2022 • 5 minutes to read • [Edit Online](#)

With .NET, you can create and deploy templates that generate projects, files, even resources. This tutorial is part two of a series that teaches you how to create, install, and uninstall, templates for use with the `dotnet new` command.

You can view the completed template in the [.NET Samples GitHub repository](#).

In this part of the series you'll learn how to:

- Create the resources of a project template
- Create the template config folder and file
- Install a template from a file path
- Test an item template
- Uninstall an item template

Prerequisites

- Complete [part 1](#) of this tutorial series.
- Open a terminal and navigate to the `working\templates` folder.

Create a project template

Project templates produce ready-to-run projects that make it easy for users to start with a working set of code. .NET includes a few project templates such as a console application or a class library. In this example, you'll create a new console project that replaces the standard "Hello World" console output with one that runs asynchronously.

In your terminal, navigate to the `working\templates` folder and create a new subfolder named `consoleasync`. Enter the subfolder and run `dotnet new console` to generate the standard console application. You'll be editing the files produced by this template to create a new template.

```
working
└── templates
    └── consoleasync
        ├── consoleasync.csproj
        └── Program.cs
```

Modify Program.cs

Open up the `Program.cs` file. The standard console project doesn't asynchronously write to the console output, so let's add that. Change the code to the following and save the file:

```
await Console.Out.WriteLineAsync("Hello World with C#");
```

Build the project

Before you complete a project template, you should test it to make sure it compiles and runs correctly.

In your terminal, run the following command.

```
dotnet run
```

You get the following output.

```
Hello World with C#
```

You can delete the `obj` and `bin` folders created by using `dotnet run`. Deleting these files ensures your template only includes the files related to your template and not any files that result from a build action.

Now that you have the content of the template created, you need to create the template config at the root folder of the template.

Create the template config

Templates are recognized in .NET by a special folder and config file that exist at the root of your template. In this tutorial, your template folder is located at `working\templates\consoleasync`.

When you create a template, all files and folders in the template folder are included as part of the template except for the special config folder. This config folder is named `.template.config`.

First, create a new subfolder named `.template.config`, enter it. Then, create a new file named `template.json`. Your folder structure should look like this.

```
working
└── templates
    └── consoleasync
        └── .template.config
            template.json
```

Open the `template.json` with your favorite text editor and paste in the following json code and save it.

```
{
  "$schema": "http://json.schemastore.org/template",
  "author": "Me",
  "classifications": [ "Common", "Console" ],
  "identity": "ExampleTemplate.AsyncProject",
  "name": "Example templates: async project",
  "shortName": "consoleasync",
  "tags": {
    "language": "C#",
    "type": "project"
  }
}
```

This config file contains all of the settings for your template. You can see the basic settings such as `name` and `shortName` but also there's a `tags/type` value that's set to `project`. This designates your template as a project template. There's no restriction on the type of template you create. The `item` and `project` values are common names that .NET recommends so that users can easily filter the type of template they're searching for.

The `classifications` item represents the `tags` column you see when you run `dotnet new` and get a list of templates. Users can also search based on classification tags. Don't confuse the `tags` property in the json file with the `classifications` tags list. They're two different things unfortunately named similarly. The full schema for the `template.json` file is found at the [JSON Schema Store](#). For more information about the `template.json` file, see the [dotnet templating wiki](#).

Now that you have a valid `.template.config/template.json` file, your template is ready to be installed. Before you install the template, make sure that you delete any extra folders and files you don't want included in your template, like the `bin` or `obj` folders. In your terminal, navigate to the `consoleasync` folder and run `dotnet new --install .\` to install the template located at the current folder. If you're using a Linux or macOS operating system, use a forward slash: `dotnet new --install ./`.

```
dotnet new --install .\
```

This command outputs a list of the installed templates, which should include yours.

```
The following template packages will be installed:  
<root path>\working\templates\consoleasync  
  
Success: <root path>\working\templates\consoleasync installed the following templates:  
Templates Short Name Language Tags  
-----  
-----  
Example templates: async project consoleasync [C#] Common/Console
```

Test the project template

Now that you have a project template installed, test it.

1. Navigate to the `test` folder
2. Create a new console application with the following command which generates a working project you can easily test with the `dotnet run` command.

```
dotnet new consoleasync
```

You get the following output.

```
The template "Example templates: async project" was created successfully.
```

3. Run the project using the following command.

```
dotnet run
```

You get the following output.

```
Hello World with C#
```

Congratulations! You created and deployed a project template with .NET. In preparation for the next part of this tutorial series, you must uninstall the template you created. Make sure to delete all files from the `test` folder too. This will get you back to a clean state ready for the next major section of this tutorial.

Uninstall the template

In your terminal, navigate to the `consoleasync` folder and run the following command to uninstall the template located in the current folder:

- On Windows: `dotnet new --uninstall .\`
- On Linux or macOS: `dotnet new --uninstall ./`

This command outputs a list of the templates that were uninstalled, which should include yours.

```
Success: <root path>\working\templates\consoleasync was uninstalled.
```

At any time, you can use `dotnet new --uninstall` to see a list of installed template packages, including for each template package the command to uninstall it.

Next steps

In this tutorial, you created a project template. To learn how to package both the item and project templates into an easy-to-use file, continue this tutorial series.

[Create a template package](#)

Tutorial: Create a template package

9/20/2022 • 6 minutes to read • [Edit Online](#)

With .NET, you can create and deploy templates that generate projects, files, and even resources. This tutorial is part three of a series that teaches you how to create, install, and uninstall templates for use with the `dotnet new` command.

You can view the completed template in the [.NET Samples GitHub repository](#).

In this part of the series you'll learn how to:

- Create a *.csproj project to build a template package
- Configure the project file for packing
- Install a template package from a NuGet package file
- Uninstall a template package by package ID

Prerequisites

- Complete [part 1](#) and [part 2](#) of this tutorial series.

This tutorial uses the two templates created in the first two parts of this tutorial. You can use a different template as long as you copy the template, as a folder, into the `working\templates` folder.

- Open a terminal and navigate to the `working\` folder.

Create a template package project

A template package is one or more templates packaged into a NuGet package. When you install or uninstall a template package, all templates contained in the package are added or removed, respectively. The previous parts of this tutorial series only worked with individual templates. To share a non-packed template, you have to copy the template folder and install via that folder. Because a template package can have more than one template in it, and is a single file, sharing is easier.

Template packages are represented by a NuGet package (.nupkg) file. And, like any NuGet package, you can upload the template package to a NuGet feed. The `dotnet new --install` command supports installing template package from a NuGet package feed. Additionally, you can install a template package from a .nupkg file directly.

Normally you use a C# project file to compile code and produce a binary. However, the project can also be used to generate a template package. By changing the settings of the `.csproj`, you can prevent it from compiling any code and instead include all the assets of your templates as resources. When this project is built, it produces a template package NuGet package.

The package you'll create will include the [item template](#) and [package template](#) previously created. Because we grouped the two templates into the `working\templates` folder, we can use the `working` folder for the `.csproj` file.

In your terminal, navigate to the `working` folder. Create a new project and set the name to `templatepack` and the output folder to the current folder.

```
dotnet new console -n templatepack -o .
```

The `-n` parameter sets the `.csproj` filename to `templatepack.csproj`. The `-o` parameter creates the files in the current directory. You should see a result similar to the following output.

```
The template "Console Application" was created successfully.

Processing post-creation actions...
Running 'dotnet restore' on ./templatepack.csproj...
Restore completed in 52.38 ms for C:\working\templatepack.csproj.

Restore succeeded.
```

The new project template generates a *Program.cs* file. You can safely delete this file as it's not used by the templates.

Next, open the *templatepack.csproj* file in your favorite editor and replace the content with the following XML:

```
<Project Sdk="Microsoft.NET.Sdk">

<PropertyGroup>
  <PackageType>Template</PackageType>
  <PackageVersion>1.0</PackageVersion>
  <PackageId>AdatumCorporation.Utility.Templates</PackageId>
  <Title>AdatumCorporation Templates</Title>
  <Authors>Me</Authors>
  <Description>Templates to use when creating an application for Adatum Corporation.</Description>
  <PackageTags>dotnet-new;templates;contoso</PackageTags>

  <TargetFramework>netstandard2.0</TargetFramework>

  <IncludeContentInPack>true</IncludeContentInPack>
  <IncludeBuildOutput>false</IncludeBuildOutput>
  <ContentTargetFolders>content</ContentTargetFolders>
  <NoWarn>$(NoWarn);NU5128</NoWarn>
  <NoDefaultExcludes>true</NoDefaultExcludes>
</PropertyGroup>

<ItemGroup>
  <Content Include="templates\**\*" Exclude="templates\**\bin\**;templates\**\obj\**" />
  <Compile Remove="**\**" />
</ItemGroup>

</Project>
```

The settings under `<PropertyGroup>` in the XML snippet are broken into three groups.

The first group deals with properties required for a NuGet package. The three `<Package*>` settings have to do with the NuGet package properties to identify your package on a NuGet feed. Specifically the `<PackageId>` value is used to uninstall the template package with a single name instead of a directory path. It can also be used to install the template package from a NuGet feed. The remaining settings, such as `<Title>` and `<PackageTags>`, have to do with metadata displayed on the NuGet feed. For more information about NuGet settings, see [NuGet and MSBuild properties](#).

NOTE

To ensure that the template package appears in `dotnet new --search` results, set `<PackageType>` to `Template`.

In the second group, the `<TargetFramework>` setting ensures that MSBuild executes properly when you run the pack command to compile and pack the project.

The third group includes settings that have to do with configuring the project to include the templates in the appropriate folder in the NuGet pack when it's created:

- The `<NoWarn>` setting suppresses a warning message that doesn't apply to template package projects.
- The `<NoDefaultExcludes>` setting ensures that files and folders that start with a `.` (like `.gitignore`) are part of the template. The *default* behavior of NuGet packages is to ignore those files and folders.

`<ItemGroup>` contains two items. First, the `<Content>` item includes everything in the *templates* folder as content. It's also set to exclude any *bin* folder or *obj* folder to prevent any compiled code (if you tested and compiled your templates) from being included. Second, the `<Compile>` item excludes all code files from compiling no matter where they're located. This setting prevents the project that's used to create the template package from trying to compile the code in the *templates* folder hierarchy.

Build and install

Save the project file. Before building the template package, verify that your folder structure is correct. Any template you want to pack should be placed in the *templates* folder, in its own folder. The folder structure should look similar to the following hierarchy:

```
working
|   templatepack.csproj
└── templates
    ├── extensions
    |   └── .template.config
    |       template.json
    └── consoleasync
        └── .template.config
            template.json
```

The *templates* folder has two folders: *extensions* and *consoleasync*.

In your terminal, from the *working* folder, run the `dotnet pack` command. This command builds your project and creates a NuGet package in the *working\bin\Debug* folder, as indicated by the following output:

```
C:\working> dotnet pack

Microsoft (R) Build Engine version 16.8.0+126527ff1 for .NET
Copyright (C) Microsoft Corporation. All rights reserved.

Restore completed in 123.86 ms for C:\working\templatepack.csproj.

templatepack -> C:\working\bin\Debug\netstandard2.0\templatepack.dll
Successfully created package 'C:\working\bin\Debug\AdatumCorporation.Utility.Templates.1.0.0.nupkg'.
```

Next, install the template package with the `dotnet new --install PATH_TO_NUPKG_FILE` command.

```
C:\working> dotnet new -i C:\working\bin\Debug\AdatumCorporation.Utility.Templates.1.0.0.nupkg
The following template packages will be installed:
  C:\working\bin\Debug\AdatumCorporation.Utility.Templates.1.0.0.nupkg

Success: AdatumCorporation.Utility.Templates::1.0.0 installed the following templates:
  Templates           Short Name      Language      Tags
  -----
  Example templates: string extensions      stringext      [C#]      Common/Code
  Example templates: async project          consoleasync      [C#]      Common/Console/C#9
```

If you uploaded the NuGet package to a NuGet feed, you can use the `dotnet new --install PACKAGEID` command where `PACKAGEID` is the same as the `<PackageId>` setting from the *.csproj* file. This package ID is the same as the

NuGet package identifier.

Uninstall the template package

No matter how you installed the template package, either with the `.nupkg` file directly or by NuGet feed, removing a template package is the same. Use the `<PackageId>` of the template you want to uninstall. You can get a list of templates that are installed by running the `dotnet new --uninstall` command.

```
C:\working> dotnet new --uninstall

Template Instantiation Commands for .NET CLI

Currently installed items:
Microsoft.DotNet.Common.ProjectTemplates.2.2
  Details:
    NuGetPackageId: Microsoft.DotNet.Common.ProjectTemplates.2.2
    Version: 1.0.2-beta4
    Author: Microsoft
  Templates:
    Class library (classlib) C#
    Class library (classlib) F#
    Class library (classlib) VB
    Console Application (console) C#
    Console Application (console) F#
    Console Application (console) VB
  Uninstall Command:
    dotnet new --uninstall Microsoft.DotNet.Common.ProjectTemplates.2.2

... cut to save space ...

AdatumCorporation.Utility.Templates
  Details:
    NuGetPackageId: AdatumCorporation.Utility.Templates
    Version: 1.0.0
    Author: Me
  Templates:
    Example templates: async project (consoleasync) C#
    Example templates: string extensions (stringext) C#
  Uninstall Command:
    dotnet new --uninstall AdatumCorporation.Utility.Templates
```

Run `dotnet new --uninstall AdatumCorporation.Utility.Templates` to uninstall the template package. The command will output information about what template packages were uninstalled.

Congratulations! You've installed and uninstalled a template package.

Next steps

To learn more about templates, most of which you've already learned, see the [Custom templates for dotnet new](#) article.

- [dotnettemplating GitHub repo Wiki](#)
- [dotnet/dotnet-template-samples GitHub repo](#)
- [template.json schema at the JSON Schema Store](#)

Obsolete features in .NET 5+

9/20/2022 • 5 minutes to read • [Edit Online](#)

Starting in .NET 5, some APIs that are newly marked as obsolete make use of two new properties on [ObsoleteAttribute](#).

- The [ObsoleteAttribute.DiagnosticId](#) property tells the compiler to generate build warnings using a custom diagnostic ID. The custom ID allows for obsolescence warning to be suppressed specifically and separately from one another. In the case of the .NET 5+ obsolescences, the format for the custom diagnostic ID is `SYSLIB0XXX`.
- The [ObsoleteAttribute.UrlFormat](#) property tells the compiler to include a URL link to learn more about the obsolescence.

If you encounter build warnings or errors due to usage of an obsolete API, follow the specific guidance provided for the diagnostic ID listed in the [Reference](#) section. Warnings or errors for these obsolescences *can't* be suppressed using the [standard diagnostic ID \(CS0618\)](#) for obsolete types or members; use the custom `SYSLIB0XXX` diagnostic ID values instead. For more information, see [Suppress warnings](#).

Reference

The following table provides an index to the `SYSLIB0XXX` obsolescences in .NET 5+.

DIAGNOSTIC ID	WARNING OR ERROR	DESCRIPTION
SYSLIB0001	Warning	The UTF-7 encoding is insecure and should not be used. Consider using UTF-8 instead.
SYSLIB0002	Error	PrincipalPermissionAttribute is not honored by the runtime and must not be used.
SYSLIB0003	Warning	Code access security (CAS) is not supported or honored by the runtime.
SYSLIB0004	Warning	The constrained execution region (CER) feature is not supported.
SYSLIB0005	Warning	The global assembly cache (GAC) is not supported.
SYSLIB0006	Warning	Thread.Abort() is not supported and throws PlatformNotSupportedException .
SYSLIB0007	Warning	The default implementation of this cryptography algorithm is not supported.

DIAGNOSTIC ID	WARNING OR ERROR	DESCRIPTION
SYSLIB0008	Warning	The CreatePdbGenerator() API is not supported and throws PlatformNotSupportedException .
SYSLIB0009	Warning	The AuthenticationManager.Authenticate and AuthenticationManager.PreAuthenticate methods are not supported and throw PlatformNotSupportedException .
SYSLIB0010	Warning	Some remoting APIs are not supported and throw PlatformNotSupportedException .
SYSLIB0011	Warning	BinaryFormatter serialization is obsolete and should not be used.
SYSLIB0012	Warning	Assembly.CodeBase and Assembly.EscapedCodeBase are only included for .NET Framework compatibility. Use Assembly.Location instead.
SYSLIB0013	Warning	Uri.EscapeUriString(String) can corrupt the Uri string in some cases. Consider using Uri.EscapeDataString(String) for query string components instead.
SYSLIB0014	Warning	WebRequest , HttpWebRequest , ServicePoint , and WebClient are obsolete. Use HttpClient instead.
SYSLIB0015	Warning	DisablePrivateReflectionAttribute has no effect in .NET 6+.
SYSLIB0016	Warning	Use the Graphics.GetContextInfo overloads that accept arguments for better performance and fewer allocations.
SYSLIB0017	Warning	Strong-name signing is not supported and throws PlatformNotSupportedException .
SYSLIB0018	Warning	Reflection-only loading is not supported and throws PlatformNotSupportedException .

Diagnostic ID	Warning or Error	Description
SYSLIB0019	Warning	The System.Runtime.InteropServices.RuntimeEnvironment members SystemConfigurationFile , GetRuntimeInterfaceAsIntPtr(Guid, Guid) , and GetRuntimeInterfaceAsObject(Guid, Guid) are no longer supported and throw PlatformNotSupportedException .
SYSLIB0020	Warning	JsonSerializerOptions.IgnoreNullValues is obsolete. To ignore null values when serializing, set DefaultIgnoreCondition to JsonIgnoreCondition.WhenWritingNull .
SYSLIB0021	Warning	Derived cryptographic types are obsolete. Use the create method on the base type instead.
SYSLIB0022	Warning	The Rijndael and RijndaelManaged types are obsolete. Use Aes instead.
SYSLIB0023	Warning	RNGCryptoServiceProvider is obsolete. To generate a random number, use one of the RandomNumberGenerator static methods instead.
SYSLIB0024	Warning	Creating and unloading AppDomains is not supported and throws an exception.
SYSLIB0025	Warning	SuppressIldasmAttribute has no effect in .NET 6+.
SYSLIB0026	Warning	X509Certificate and X509Certificate2 are immutable. Use the appropriate constructor to create a new certificate.
SYSLIB0027	Warning	PublicKey.Key is obsolete. Use the appropriate method to get the public key, such as GetRSAPublicKey() .
SYSLIB0028	Warning	X509Certificate2.PrivateKey is obsolete. Use the appropriate method to get the private key, such as RSACertificateExtensions.GetRSAPrivateKey(X509Certificate2) , or use the X509Certificate2.CopyWithPrivateKey(ECDiffieHellman) method to create a new instance with a private key.
SYSLIB0029	Warning	ProduceLegacyHmacValues is obsolete. Producing legacy HMAC values is no longer supported.

DIAGNOSTIC ID	WARNING OR ERROR	DESCRIPTION
SYSLIB0030	Warning	<code>HMACSHA1</code> always uses the algorithm implementation provided by the platform. Use a constructor without the <code>useManagedSha1</code> parameter.
SYSLIB0031	Warning	<code>CryptoConfig.EncodeOID(String)</code> is obsolete. Use the ASN.1 functionality provided in <code>System.Formats.Asn1</code> .
SYSLIB0032	Warning	Recovery from corrupted process state exceptions is not supported; <code>HandleProcessCorruptedStateExceptionsAttribute</code> is ignored.
SYSLIB0033	Warning	<code>Rfc2898DeriveBytes.CryptDeriveKey(String, String, Int32, Byte[])</code> is obsolete and is not supported. Use <code>PasswordDeriveBytes.CryptDeriveKey(String, String, Int32, Byte[])</code> instead.
SYSLIB0034	Warning	<code>CmsSigner(CspParameters)</code> is obsolete. Use an alternative constructor instead.
SYSLIB0035	Warning	<code>SignerInfo.ComputeCounterSignature()</code> is obsolete. Use the overload that accepts a <code>CmsSigner</code> instead.
SYSLIB0036	Warning	<code>Regex.CompileToAssembly</code> is obsolete and not supported. Use <code>RegexGeneratorAttribute</code> with the regular expression source generator instead.
SYSLIB0037	Warning	<code>AssemblyName</code> members <code>HashAlgorithm</code> , <code>ProcessorArchitecture</code> , and <code>VersionCompatibility</code> are obsolete and not supported.
SYSLIB0038	Warning	<code>SerializationFormat.Binary</code> is obsolete and should not be used.
SYSLIB0039	Warning	TLS versions 1.0 and 1.1 have known vulnerabilities and are not recommended. Use a newer TLS version instead, or use <code>SslProtocols.None</code> to defer to OS defaults.
SYSLIB0040	Warning	<code>EncryptionPolicy.NoEncryption</code> and <code>EncryptionPolicy.AllowNoEncryption</code> significantly reduce security and should not be used in production code.

DIAGNOSTIC ID	WARNING OR ERROR	DESCRIPTION
SYSLIB0041	Warning	The default hash algorithm and iteration counts in Rfc2898DeriveBytes constructors are outdated and insecure. Use a constructor that accepts the hash algorithm and the number of iterations.
SYSLIB0042	Warning	<code>ToXmlString</code> and <code>FromXmlString</code> have no implementation for elliptic curve cryptography (ECC) types, and are obsolete. Use a standard import and export format such as <code>ExportSubjectPublicKeyInfo</code> or <code>ImportSubjectPublicKeyInfo</code> for public keys, and <code>ExportPkcs8PrivateKey</code> or <code>ImportPkcs8PrivateKey</code> for private keys.
SYSLIB0043	Warning	<code>ECDiffieHellmanPublicKey.ToByteArray()</code> and the associated constructor do not have a consistent and interoperable implementation on all platforms. Use <code>ECDiffieHellmanPublicKey.ExportSubjectPublicKeyInfo()</code> instead.
SYSLIB0047	Warning	<code>XmlSecureResolver</code> is obsolete. Use <code>XmlResolver.ThrowingResolver</code> instead when attempting to forbid XML external entity resolution.

Suppress warnings

It's recommended that you use an available workaround whenever possible. However, if you cannot change your code, you can suppress warnings through a `#pragma` directive or a `<NoWarn>` project setting. If you must use the obsolete APIs and the `SYSLIB0XXX` diagnostic does not surface as an error, you can suppress the warning in code or in your project file.

To suppress the warnings in code:

```
// Disable the warning.
#pragma warning disable SYSLIB0001

// Code that uses obsolete API.
//...

// Re-enable the warning.
#pragma warning restore SYSLIB0001
```

To suppress the warnings in a project file:

```
<Project Sdk="Microsoft.NET.Sdk">
<PropertyGroup>
<TargetFramework>net6.0</TargetFramework>
<!-- NoWarn below suppresses SYSLIB0001 project-wide -->
<NoWarn>$(NoWarn);SYSLIB0001</NoWarn>
<!-- To suppress multiple warnings, you can use multiple NoWarn elements -->
<NoWarn>$(NoWarn);SYSLIB0002</NoWarn>
<NoWarn>$(NoWarn);SYSLIB0003</NoWarn>
<!-- Alternatively, you can suppress multiple warnings by using a semicolon-delimited list -->
<NoWarn>$(NoWarn);SYSLIB0001;SYSLIB0002;SYSLIB0003</NoWarn>
</PropertyGroup>
</Project>
```

NOTE

Suppressing warnings in this way only disables the obsoletion warnings you specify. It doesn't disable any other warnings, including obsoletion warnings with different diagnostic IDs.

See also

- [API obsolesions with non-default diagnostic IDs \(.NET 5\)](#)
- [API obsolesions with non-default diagnostic IDs \(.NET 6\)](#)
- [API obsolesions with non-default diagnostic IDs \(.NET 7\)](#)

SYSLIB0001: The UTF-7 encoding is insecure

9/20/2022 • 2 minutes to read • [Edit Online](#)

The UTF-7 encoding is no longer in wide use among applications, and many specs now [forbid its use](#) in interchange. It's also occasionally [used as an attack vector](#) in applications that don't anticipate encountering UTF-7-encoded data. Microsoft warns against use of [System.Text.UTF7Encoding](#) because it doesn't provide error detection.

Consequently, the following APIs are marked obsolete, starting in .NET 5. Use of these APIs generates warning `SYSLIB0001` at compile time.

- [Encoding.UTF7](#) property
- [UTF7Encoding](#) constructors

Workarounds

- If you're using [Encoding.UTF7](#) or [UTF7Encoding](#) within your own protocol or file format:

Switch to using [Encoding.UTF8](#) or [UTF8Encoding](#). UTF-8 is an industry standard and is widely supported across languages, operating systems, and runtimes. Using UTF-8 eases future maintenance of your code and makes it more interoperable with the rest of the ecosystem.

- If you're comparing an [Encoding](#) instance against [Encoding.UTF7](#):

Instead, consider performing a check against the well-known UTF-7 code page, which is `65000`. By comparing against the code page, you avoid the warning and also handle some edge cases, such as if somebody called `new UTF7Encoding()` or subclassed the type.

```
void DoSomething(Encoding enc)
{
    // Don't perform the check this way.
    // It produces a warning and misses some edge cases.
    if (enc == Encoding.UTF7)
    {
        // Encoding is UTF-7.
    }

    // Instead, perform the check this way.
    if (enc != null && enc.CodePage == 65000)
    {
        // Encoding is UTF-7.
    }
}
```

Suppress a warning

If you must use the obsolete APIs, you can suppress the warning in code or in your project file.

To suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the warning.

```
// Disable the warning.  
#pragma warning disable SYSLIB0001  
  
// Code that uses obsolete API.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB0001
```

To suppress all the `SYSLIB0001` warnings in your project, add a `<NoWarn>` property to your project file.

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    ...  
    <NoWarn>$(NoWarn);SYSLIB0001</NoWarn>  
  </PropertyGroup>  
</Project>
```

For more information, see [Suppress warnings](#).

See also

- [UTF-7 code paths are obsolete](#)

SYSLIB0002: PrincipalPermissionAttribute is obsolete

9/20/2022 • 2 minutes to read • [Edit Online](#)

The [PrincipalPermissionAttribute](#) constructor is obsolete and produces compile-time error `SYSLIB0002`, starting in .NET 5. You cannot instantiate this attribute or apply it to a method.

Unlike other obsolescence warnings, you can't suppress the error.

Workarounds

- If you're applying the attribute to an ASP.NET MVC action method:

Consider using ASP.NET's built-in authorization infrastructure. The following code demonstrates how to annotate a controller with an [AuthorizeAttribute](#) attribute. The ASP.NET runtime will authorize the user before performing the action.

```
using Microsoft.AspNetCore.Authorization;

namespace MySampleApp
{
    [Authorize(Roles = "Administrator")]
    public class AdministrationController : Controller
    {
        public ActionResult MyAction()
        {
            // This code won't run unless the current user
            // is in the 'Administrator' role.
        }
    }
}
```

For more information, see [Role-based authorization in ASP.NET Core](#) and [Introduction to authorization in ASP.NET Core](#).

- If you're applying the attribute to library code outside the context of a web app:

Perform the checks manually at the beginning of your method by calling the [IPrincipal.IsInRole\(String\)](#) method.

```
using System.Threading;

void DoSomething()
{
    if (Thread.CurrentPrincipal == null
        || !Thread.CurrentPrincipal.IsInRole("Administrators"))
    {
        throw new Exception("User is anonymous or isn't an admin.");
    }

    // Code that should run only when user is an administrator.
}
```

See also

[PrincipalPermissionAttribute is obsolete as error](#)

SYSLIB0003: Code access security is not supported

9/20/2022 • 2 minutes to read • [Edit Online](#)

[Code access security \(CAS\)](#) is an unsupported, legacy technology. The infrastructure to enable CAS, which exists only in .NET Framework 2.x - 4.x, is deprecated and not receiving servicing or security fixes.

As a result, most code access security (CAS)-related types in .NET are obsolete, starting in .NET 5. This includes CAS attributes, such as [SecurityPermissionAttribute](#), CAS permission objects, such as [SocketPermission](#), [EvidenceBase](#)-derived types, and other supporting APIs. Using these APIs generates warning `SYSLIB0003` at compile time.

The complete list of obsolete CAS APIs is as follows:

- [System.AppDomain.ExecuteAssembly\(String, String\[\], Byte\[\], AssemblyHashAlgorithm\)](#)
- [System.AppDomain.PermissionSet](#)
- [System.Configuration.ConfigurationPermission](#)
- [System.Configuration.ConfigurationPermissionAttribute](#)
- [System.Data.Common.DBDataPermission](#)
- [System.Data.Common.DBDataPermissionAttribute](#)
- [System.Data.Odbc.OdbcPermission](#)
- [System.Data.Odbc.OdbcPermissionAttribute](#)
- [System.Data.OleDb.OleDbPermission](#)
- [System.Data.OleDb.OleDbPermissionAttribute](#)
- [System.Data.OracleClient.OraclePermission](#)
- [System.Data.OracleClient.OraclePermissionAttribute](#)
- [System.Data.SqlClient.SqlClientPermission](#)
- [System.Data.SqlClient.SqlClientPermissionAttribute](#)
- [System.Diagnostics.EventLogPermission](#)
- [System.Diagnostics.EventLogPermissionAttribute](#)
- [System.Diagnostics.PerformanceCounterPermission](#)
- [System.Diagnostics.PerformanceCounterPermissionAttribute](#)
- [System.DirectoryServices.DirectoryServicesPermission](#)
- [System.DirectoryServices.DirectoryServicesPermissionAttribute](#)
- [System.Drawing.Printing.PrintingPermission](#)
- [System.Drawing.Printing.PrintingPermissionAttribute](#)
- [System.Net.DnsPermission](#)
- [System.Net.DnsPermissionAttribute](#)
- [System.Net.Mail.SmtpPermission](#)
- [System.Net.Mail.SmtpPermissionAttribute](#)
- [System.Net.NetworkInformation.NetworkInformationPermission](#)
- [System.Net.NetworkInformation.NetworkInformationPermissionAttribute](#)
- [System.Net.PeerToPeer.Collaboration.PeerCollaborationPermission](#)
- [System.Net.PeerToPeer.Collaboration.PeerCollaborationPermissionAttribute](#)
- [System.Net.PeerToPeer.PnrpPermission](#)
- [System.Net.PeerToPeer.PnrpPermissionAttribute](#)
- [System.Net.SocketPermission](#)

- [System.Net.SocketPermissionAttribute](#)
- [System.Net.WebPermission](#)
- [System.Net.WebPermissionAttribute](#)
- [System.Runtime.InteropServices.AllowReversePInvokeCallsAttribute](#)
- [System.Security.CodeAccessPermission](#)
- [System.Security.HostProtectionException](#)
- [System.Security.IPermission](#)
- [System.Security.IStackWalk](#)
- [System.Security.NamedPermissionSet](#)
- [System.Security.PermissionSet](#)
- [System.Security.Permissions.CodeAccessSecurityAttribute](#)
- [System.Security.Permissions.DataProtectionPermission](#)
- [System.Security.Permissions.DataProtectionPermissionAttribute](#)
- [System.Security.Permissions.DataProtectionPermissionFlags](#)
- [System.Security.Permissions.EnvironmentPermission](#)
- [System.Security.Permissions.EnvironmentPermissionAccess](#)
- [System.Security.Permissions.EnvironmentPermissionAttribute](#)
- [System.Security.Permissions.FileDialogPermission](#)
- [System.Security.Permissions.FileDialogPermissionAccess](#)
- [System.Security.Permissions.FileDialogPermissionAttribute](#)
- [System.Security.Permissions.FileIOPermission](#)
- [System.Security.Permissions.FileIOPermissionAccess](#)
- [System.Security.Permissions.FileIOPermissionAttribute](#)
- [System.Security.Permissions.GacIdentityPermission](#)
- [System.Security.Permissions.GacIdentityPermissionAttribute](#)
- [System.Security.Permissions.HostProtectionAttribute](#)
- [System.Security.Permissions.HostProtectionResource](#)
- [System.Security.Permissions.IUnrestrictedPermission](#)
- [System.Security.Permissions.IsolatedStorageContainment](#)
- [System.Security.Permissions.IsolatedStorageFilePermission](#)
- [System.Security.Permissions.IsolatedStorageFilePermissionAttribute](#)
- [System.Security.Permissions.IsolatedStoragePermission](#)
- [System.Security.Permissions.IsolatedStoragePermissionAttribute](#)
- [System.Security.Permissions.KeyContainerPermission](#)
- [System.Security.Permissions.KeyContainerPermissionAccessEntry](#)
- [System.Security.Permissions.KeyContainerPermissionAccessEntryCollection](#)
- [System.Security.Permissions.KeyContainerPermissionAccessEntryEnumerator](#)
- [System.Security.Permissions.KeyContainerPermissionAttribute](#)
- [System.Security.Permissions.KeyContainerPermissionFlags](#)
- [System.Security.Permissions.MediaPermission](#)
- [System.Security.Permissions.MediaPermissionAttribute](#)
- [System.Security.Permissions.MediaPermissionAudio](#)
- [System.Security.Permissions.MediaPermissionImage](#)
- [System.Security.Permissions.MediaPermissionVideo](#)
- [System.Security.Permissions.PermissionSetAttribute](#)
- [System.Security.Permissions.PermissionState](#)

- [System.Security.Permissions.PrincipalPermission](#)
- [System.Security.Permissions.PrincipalPermissionAttribute](#)
- [System.Security.Permissions.PublisherIdentityPermission](#)
- [System.Security.Permissions.PublisherIdentityPermissionAttribute](#)
- [System.Security.Permissions.ReflectionPermission](#)
- [System.Security.Permissions.ReflectionPermissionAttribute](#)
- [System.Security.Permissions.ReflectionPermissionFlag](#)
- [System.Security.Permissions.RegistryPermission](#)
- [System.Security.Permissions.RegistryPermissionAccess](#)
- [System.Security.Permissions.RegistryPermissionAttribute](#)
- [System.Security.Permissions.ResourcePermissionBase](#)
- [System.Security.Permissions.ResourcePermissionBaseEntry](#)
- [System.Security.Permissions.SecurityAction](#)
- [System.Security.Permissions.SecurityAttribute](#)
- [System.Security.Permissions.SecurityPermission](#)
- [System.Security.Permissions.SecurityPermissionAttribute](#)
- [System.Security.Permissions.SecurityPermissionFlag](#)
- [System.Security.Permissions.SitelIdentityPermission](#)
- [System.Security.Permissions.SitelIdentityPermissionAttribute](#)
- [System.Security.Permissions.StorePermission](#)
- [System.Security.Permissions.StorePermissionAttribute](#)
- [System.Security.Permissions.StorePermissionFlags](#)
- [System.Security.Permissions.StrongNameIdentityPermission](#)
- [System.Security.Permissions.StrongNameIdentityPermissionAttribute](#)
- [System.Security.Permissions.StrongNamePublicKeyBlob](#)
- [System.Security.Permissions.TypeDescriptorPermission](#)
- [System.Security.Permissions.TypeDescriptorPermissionAttribute](#)
- [System.Security.Permissions.TypeDescriptorPermissionFlags](#)
- [System.Security.Permissions.UIPermission](#)
- [System.Security.Permissions.UIPermissionAttribute](#)
- [System.Security.Permissions.UIPermissionClipboard](#)
- [System.Security.Permissions.UIPermissionWindow](#)
- [System.Security.Permissions.UrlIdentityPermission](#)
- [System.Security.Permissions.UrlIdentityPermissionAttribute](#)
- [System.Security.Permissions.WebBrowserPermission](#)
- [System.Security.Permissions.WebBrowserPermissionAttribute](#)
- [System.Security.Permissions.WebBrowserPermissionLevel](#)
- [System.Security.Permissions.ZoneIdentityPermission](#)
- [System.Security.Permissions.ZoneIdentityPermissionAttribute](#)
- [System.Security.Policy.ApplicationTrust.ApplicationTrust\(PermissionSet, IEnumerable<StrongName>\)](#)
- [System.Security.Policy.ApplicationTrust.FullTrustAssemblies](#)
- [System.Security.Policy.FileCodeGroup](#)
- [System.Security.Policy.GacInstalled](#)
- [System.Security.Policy.IIdentityPermissionFactory](#)
- [System.Security.Policy.PolicyLevel.AddNamedPermissionSet\(NamedPermissionSet\)](#)
- [System.Security.Policy.PolicyLevel.ChangeNamedPermissionSet\(String, PermissionSet\)](#)

- [System.Security.Policy.PolicyLevel.GetNamedPermissionSet\(String\)](#)
- [System.Security.Policy.PolicyLevel.RemoveNamedPermissionSet](#)
- [System.Security.Policy.PolicyStatement.PermissionSet](#)
- [System.Security.Policy.PolicyStatement.PolicyStatement](#)
- [System.Security.Policy.Publisher](#)
- [System.Security.Policy.Site](#)
- [System.Security.Policy.StrongName](#)
- [System.Security.Policy.StrongNameMembershipCondition](#)
- [System.Security.Policy.Url](#)
- [System.Security.Policy.Zone](#)
- [System.Security.SecurityContext](#)
- [System.Security.SecurityManager](#)
- [System.ServiceProcess.ServiceControllerPermission](#)
- [System.ServiceProcess.ServiceControllerPermissionAttribute](#)
- [System.Threading.Thread.GetCompressedStack\(\)](#)
- [System.Threading.Thread.SetCompressedStack\(CompressedStack\)](#)
- [System.Transactions.DistributedTransactionPermission](#)
- [System.Transactions.DistributedTransactionPermissionAttribute](#)
- [System.Web.AspNetHostingPermission](#)
- [System.Web.AspNetHostingPermissionAttribute](#)
- [System.Xaml.Permissions.XamlLoadPermission](#)

Workarounds

- If you're asserting any security permission, remove the attribute or call that asserts the permission.

```
// REMOVE the attribute below.
[SecurityPermission(SecurityAction.Assert, ControlThread = true)]
public void DoSomething()
{
}
public void DoAssert()
{
    // REMOVE the line below.
    new SecurityPermission(SecurityPermissionFlag.ControlThread).Assert();
}
```

- If you're denying or restricting (via `PermitOnly`) any permission, contact your security advisor. Because CAS attributes are not honored by the .NET 5+ runtime, your application could have a security hole if it incorrectly relies on the CAS infrastructure to restrict access to these methods.

```
// REVIEW the attribute below; could indicate security vulnerability.
[SecurityPermission(SecurityAction.Deny, ControlThread = true)]
public void DoSomething()
{
}
public void DoPermitOnly()
{
    // REVIEW the line below; could indicate security vulnerability.
    new SecurityPermission(SecurityPermissionFlag.ControlThread).PermitOnly();
}
```

- If you're demanding any permission (except `PrincipalPermission`), remove the demand. All demands will

succeed at run time.

```
// REMOVE the attribute below; it will always succeed.  
[SecurityPermission(SecurityAction.Demand, ControlThread = true)]  
public void DoSomething()  
{  
}  
public void DoDemand()  
{  
    // REMOVE the line below; it will always succeed.  
    new SecurityPermission(SecurityPermissionFlag.ControlThread).Demand();  
}
```

- If you're demanding [PrincipalPermission](#), consult the guidance for [SYSLIB0002: PrincipalPermissionAttribute is obsolete](#). That guidance applies for both [PrincipalPermission](#) and [PrincipalPermissionAttribute](#).

Suppress a warning

If you must use the obsolete APIs, you can suppress the warning in code or in your project file.

To suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the warning.

```
// Disable the warning.  
#pragma warning disable SYSLIB0003  
  
// Code that uses obsolete API.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB0003
```

To suppress all the `SYSLIB0003` warnings in your project, add a `<NoWarn>` property to your project file.

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    ...  
    <NoWarn>$(NoWarn);SYSLIB0003</NoWarn>  
  </PropertyGroup>  
</Project>
```

For more information, see [Suppress warnings](#).

See also

- [SYSLIB0002: PrincipalPermissionAttribute is obsolete](#)

SYSLIB0004: The constrained execution region (CER) feature is not supported

9/20/2022 • 2 minutes to read • [Edit Online](#)

The [Constrained execution regions \(CER\)](#) feature is supported only in .NET Framework. As such, various CER-related APIs are marked obsolete, starting in .NET 5. Using these APIs generates warning `SYSLIB0004` at compile time.

The following CER-related APIs are obsolete:

- [RuntimeHelpers.ExecuteCodeWithGuaranteedCleanup\(RuntimeHelpers+TryCode, RuntimeHelpers+CleanupCode, Object\)](#)
- [RuntimeHelpers.PrepareConstrainedRegions\(\)](#)
- [RuntimeHelpers.PrepareConstrainedRegionsNoOP\(\)](#)
- [RuntimeHelpers.PrepareContractedDelegate\(Delegate\)](#)
- [RuntimeHelpers.ProbeForSufficientStack\(\)](#)
- [System.Runtime.ConstrainedExecution.Cer](#)
- [System.Runtime.ConstrainedExecution.Consistency](#)
- [System.Runtime.ConstrainedExecution.PrePrepareMethodAttribute](#)
- [System.Runtime.ConstrainedExecution.ReliabilityContractAttribute](#)

However, the following CER-related APIs are *not* obsolete:

- [RuntimeHelpers.PrepareDelegate\(Delegate\)](#)
- [RuntimeHelpers.PrepareMethod](#)
- [System.Runtime.ConstrainedExecution.CriticalFinalizerObject](#)

Workarounds

- If you've applied a CER attribute to a method, remove the attribute. These attributes have no effect in .NET 5 and later versions.

```
// REMOVE the attribute below.  
[ReliabilityContract(Consistency.WillNotCorruptState, Cer.Success)]  
public void DoSomething()  
{  
}  
  
// REMOVE the attribute below.  
[PrePrepareMethod]  
public void DoSomething()  
{  
}
```

- If you are calling `RuntimeHelpers.ProbeForSufficientStack` or `RuntimeHelpers.PrepareContractedDelegate`, remove the call. These calls have no effect in .NET 5 and later versions.

```
public void DoSomething()
{
    // REMOVE the call below.
    RuntimeHelpers.ProbeForSufficientStack();

    // (Remainder of your method logic here.)
}
```

- If you are calling `RuntimeHelpers.PrepareConstrainedRegions`, remove the call. This call has no effect in .NET 5 and later versions.

```
public void DoSomething_Old()
{
    // REMOVE the call below.
    RuntimeHelpers.PrepareConstrainedRegions();
    try
    {
        // try code
    }
    finally
    {
        // cleanup code
    }
}

public void DoSomething_Corrected()
{
    // There is no call to PrepareConstrainedRegions. It's a normal try / finally block.

    try
    {
        // try code
    }
    finally
    {
        // cleanup code
    }
}
```

- If you are calling `RuntimeHelpers.ExecuteCodeWithGuaranteedCleanup`, replace the call with a standard `try/catch/finally` block.

```
// The sample below produces warning SYSLIB0004.
public void DoSomething_Old()
{
    RuntimeHelpers.ExecuteCodeWithGuaranteedCleanup(MyTryCode, MyCleanupCode, null);
}
public void MyTryCode(object state) { /* try code */ }
public void MyCleanupCode(object state, bool exceptionThrown) { /* cleanup code */ }

// The corrected sample below does not produce warning SYSLIB0004.
public void DoSomething_Corrected()
{
    try
    {
        // try code
    }
    catch (Exception ex)
    {
        // exception handling code
    }
    finally
    {
        // cleanup code
    }
}
```

Suppress a warning

If you must use the obsolete APIs, you can suppress the warning in code or in your project file.

To suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the warning.

```
// Disable the warning.
#pragma warning disable SYSLIB0004

// Code that uses obsolete API.
// ...

// Re-enable the warning.
#pragma warning restore SYSLIB0004
```

To suppress all the `SYSLIB0004` warnings in your project, add a `<NoWarn>` property to your project file.

```
<Project Sdk="Microsoft.NET.Sdk">
<PropertyGroup>
    ...
    <NoWarn>$(NoWarn);SYSLIB0004</NoWarn>
</PropertyGroup>
</Project>
```

For more information, see [Suppress warnings](#).

See also

- [Constrained execution regions](#)

SYSLIB0005: The global assembly cache (GAC) is not supported

9/20/2022 • 2 minutes to read • [Edit Online](#)

.NET Core and .NET 5 and later versions eliminate the concept of the global assembly cache (GAC) that was present in .NET Framework. To help steer developers away from these APIs, some GAC-related APIs are marked as obsolete, starting in .NET 5. Using these APIs generates warning `SYSLIB0005` at compile time.

The following GAC-related APIs are marked obsolete:

- [Assembly.GlobalAssemblyCache](#)

Libraries and apps should not use the [GlobalAssemblyCache](#) API to make determinations about run-time behavior, as it always returns `false` in .NET Core and .NET 5+.

Workarounds

If your application queries the [GlobalAssemblyCache](#) property, consider removing the call. If you use the [GlobalAssemblyCache](#) value to choose between an "assembly in the GAC"-flow vs. an "assembly not in the GAC"-flow at run time, reconsider whether the flow still makes sense for a .NET 5+ application.

Suppress a warning

If you must use the obsolete APIs, you can suppress the warning in code or in your project file.

To suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the warning.

```
// Disable the warning.  
#pragma warning disable SYSLIB0005  
  
// Code that uses obsolete API.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB0005
```

To suppress all the `SYSLIB0005` warnings in your project, add a `<NoWarn>` property to your project file.

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    ...  
    <NoWarn>$(NoWarn);SYSLIB0005</NoWarn>  
  </PropertyGroup>  
</Project>
```

For more information, see [Suppress warnings](#).

See also

- [Global assembly cache](#)

SYSLIB0006: Thread.Abort is not supported

9/20/2022 • 2 minutes to read • [Edit Online](#)

The following APIs are marked obsolete, starting in .NET 5. Use of these APIs generates warning `SYSLIB0006` at compile time and a [PlatformNotSupportedException](#) at run time.

- [Thread.Abort\(\)](#)
- [Thread.Abort\(Object\)](#)

When you call [Thread.Abort](#) to abort a thread other than the current thread, you don't know what code has executed or failed to execute when the [ThreadAbortException](#) is thrown. You also cannot be certain of the state of your application or any application and user state that it's responsible for preserving. For example, calling [Thread.Abort](#) may prevent the execution of static constructors or the release of managed or unmanaged resources. For this reason, [Thread.Abort](#) always throws a [PlatformNotSupportedException](#) on .NET Core and .NET 5+.

Workarounds

Use a [CancellationToken](#) to abort processing of a unit of work instead of calling [Thread.Abort](#). The following example illustrates the use of [CancellationToken](#).

```
void ProcessPendingWorkItemsNew(CancellationToken cancellationToken)
{
    if (QueryIsMoreWorkPending())
    {
        // If the CancellationToken is marked as "needs to cancel",
        // this will throw the appropriate exception.
        cancellationToken.ThrowIfCancellationRequested();

        WorkItem work = DequeueWorkItem();
        ProcessWorkItem(work);
    }
}
```

Suppress a warning

If you must use the obsolete APIs, you can suppress the warning in code or in your project file.

To suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the warning.

```
// Disable the warning.
#pragma warning disable SYSLIB0006

// Code that uses obsolete API.
// ...

// Re-enable the warning.
#pragma warning restore SYSLIB0006
```

To suppress all the `SYSLIB0006` warnings in your project, add a `<NoWarn>` property to your project file.

```
<Project Sdk="Microsoft.NET.Sdk">
<PropertyGroup>
  ...
  <NoWarn>$(NoWarn);SYSLIB0006</NoWarn>
</PropertyGroup>
</Project>
```

For more information, see [Suppress warnings](#).

See also

- [Thread.Abort is obsolete](#)
- [Cancellation in managed threads](#)

SYSLIB0007: Default implementations of cryptography algorithms not supported

9/20/2022 • 2 minutes to read • [Edit Online](#)

The cryptographic configuration system in .NET Framework doesn't allow for proper cryptographic agility and isn't present in .NET Core and .NET 5+. .NET's backward-compatibility requirements also prohibit the framework from updating certain cryptographic APIs to keep up with advances in cryptography. As a result, the following APIs are marked obsolete, starting in .NET 5. Use of these APIs generates warning `SYSLIB0007` at compile time and a `PlatformNotSupportedException` at run time.

- `System.Security.Cryptography.AsymmetricAlgorithm.Create()`
- `System.Security.Cryptography.HashAlgorithm.Create()`
- `System.Security.Cryptography.HMAC.Create()`
- `System.Security.Cryptography.KeyedHashAlgorithm.Create()`
- `System.Security.Cryptography.SymmetricAlgorithm.Create()`

Workarounds

- The recommended course of action is to replace calls to the now-obsolete APIs with calls to factory methods for specific algorithms, for example, `Aes.Create()`. This gives you full control over which algorithms are instantiated.
- If you need to maintain compatibility with existing payloads generated by .NET Framework apps that use the now-obsolete APIs, use the replacements suggested in the following table. The table provides a mapping from .NET Framework default algorithms to their .NET 5+ equivalents.

.NET FRAMEWORK	.NET CORE / .NET 5+ COMPATIBLE REPLACEMENT	REMARKS
<code>AsymmetricAlgorithm.Create()</code>	<code>RSA.Create()</code>	
<code>HashAlgorithm.Create()</code>	<code>SHA1.Create()</code>	The SHA-1 algorithm is considered broken. Consider using a stronger algorithm if possible. Consult your security advisor for further guidance.
<code>HMAC.Create()</code>	<code>HMACSHA1()</code>	The HMACSHA1 algorithm is discouraged for most modern applications. Consider using a stronger algorithm if possible. Consult your security advisor for further guidance.
<code>KeyedHashAlgorithm.Create()</code>	<code>HMACSHA1()</code>	The HMACSHA1 algorithm is discouraged for most modern applications. Consider using a stronger algorithm if possible. Consult your security advisor for further guidance.

.NET FRAMEWORK	.NET CORE / .NET 5+ COMPATIBLE REPLACEMENT	REMARKS
<code>SymmetricAlgorithm.Create()</code>	<code>Aes.Create()</code>	

Suppress a warning

If you must use the obsolete APIs, you can suppress the warning in code or in your project file.

To suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the warning.

```
// Disable the warning.
#pragma warning disable SYSLIB0007

// Code that uses obsolete API.
// ...

// Re-enable the warning.
#pragma warning restore SYSLIB0007
```

To suppress all the `SYSLIB0007` warnings in your project, add a `<NoWarn>` property to your project file.

```
<Project Sdk="Microsoft.NET.Sdk">
<PropertyGroup>
...
<NoWarn>$({NoWarn});SYSLIB0007</NoWarn>
</PropertyGroup>
</Project>
```

For more information, see [Suppress warnings](#).

See also

- [Instantiating default implementations of cryptographic abstractions is not supported](#)

SYSLIB0008: CreatePdbGenerator is not supported

9/20/2022 • 2 minutes to read • [Edit Online](#)

The [DebugInfoGenerator.CreatePdbGenerator\(\)](#) API is marked obsolete, starting in .NET 5. Using this API generates warning `SYSLIB0008` at compile time and throws a [PlatformNotSupportedException](#) at run time.

Suppress a warning

If you must use the obsolete APIs, you can suppress the warning in code or in your project file.

To suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the warning.

```
// Disable the warning.  
#pragma warning disable SYSLIB0008  
  
// Code that uses obsolete API.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB0008
```

To suppress all the `SYSLIB0008` warnings in your project, add a `<NoWarn>` property to your project file.

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    ...  
    <NoWarn>$(NoWarn);SYSLIB0008</NoWarn>  
  </PropertyGroup>  
</Project>
```

For more information, see [Suppress warnings](#).

SYSLIB0009: The AuthenticationManager Authenticate and PreAuthenticate methods are not supported

9/20/2022 • 2 minutes to read • [Edit Online](#)

The following APIs are marked obsolete, starting in .NET 5. Use of these APIs generates warning `SYSLIB0009` at compile time and throws a `PlatformNotSupportedException` at run time.

- [AuthenticationManager.Authenticate](#)
- [AuthenticationManager.PreAuthenticate](#)

Workarounds

Implement [IAuthenticationModule](#), which has methods that were previously called by `AuthenticationManager.Authenticate`.

Suppress a warning

If you must use the obsolete APIs, you can suppress the warning in code or in your project file.

To suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the warning.

```
// Disable the warning.  
#pragma warning disable SYSLIB0009  
  
// Code that uses obsolete API.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB0009
```

To suppress all the `SYSLIB0009` warnings in your project, add a `<NoWarn>` property to your project file.

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    ...  
    <NoWarn>$(NoWarn);SYSLIB0009</NoWarn>  
  </PropertyGroup>  
</Project>
```

For more information, see [Suppress warnings](#).

SYSLIB0010: Unsupported remoting APIs

9/20/2022 • 2 minutes to read • [Edit Online](#)

.NET remoting is a legacy technology, and the infrastructure exists only in .NET Framework. The following remoting-related APIs are marked as obsolete, starting in .NET 5. Using them in code generates warning `SYSLIB0010` at compile time and throws a `PlatformNotSupportedException` at run time.

- `MarshalByRefObject.GetLifetimeService()`
- `MarshalByRefObject.InitializeLifetimeService()`

Workarounds

Consider using WCF or HTTP-based REST services to communicate with objects in other applications or across machines. For more information, see [.NET Framework technologies unavailable on .NET Core](#).

Suppress a warning

If you must use the obsolete APIs, you can suppress the warning in code or in your project file.

To suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the warning.

```
// Disable the warning.  
#pragma warning disable SYSLIB0010  
  
// Code that uses obsolete API.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB0010
```

To suppress all the `SYSLIB0010` warnings in your project, add a `<NoWarn>` property to your project file.

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    ...  
    <NoWarn>$(NoWarn);SYSLIB0010</NoWarn>  
  </PropertyGroup>  
</Project>
```

For more information, see [Suppress warnings](#).

See also

- [.NET remoting](#)

SYSLIB0011: BinaryFormatter serialization is obsolete

9/20/2022 • 2 minutes to read • [Edit Online](#)

Due to [security vulnerabilities](#) in [BinaryFormatter](#), the following APIs are marked as obsolete, starting in .NET 5. Using them in code generates warning `SYSLIB0011` at compile time.

- [System.Exception.SerializeObjectState](#)
- [BinaryFormatter.Serialize](#)
- [BinaryFormatter.Deserialize](#)
- [Formatter.Serialize\(Stream, Object\)](#)
- [Formatter.Deserialize\(Stream\)](#)
- [IFormatter.Serialize\(Stream, Object\)](#)
- [IFormatter.Deserialize\(Stream\)](#)

Workarounds

Consider using [JsonSerializer](#) or [XmlSerializer](#) instead of [BinaryFormatter](#).

For more information about recommended actions, see [Resolving BinaryFormatter obsolescence and disablement errors](#).

Suppress a warning

If you must use the obsolete APIs, you can suppress the warning in code or in your project file.

To suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the warning.

```
// Disable the warning.  
#pragma warning disable SYSLIB0011  
  
// Code that uses obsolete API.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB0011
```

To suppress all the `SYSLIB0011` warnings in your project, add a `<NoWarn>` property to your project file.

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    ...  
    <NoWarn>$(NoWarn);SYSLIB0011</NoWarn>  
  </PropertyGroup>  
</Project>
```

For more information, see [Suppress warnings](#).

See also

- [Resolving BinaryFormatter obsolescence and disablement errors](#)

- `BinaryFormatter` serialization methods are obsolete and prohibited in ASP.NET apps

SYSLIB0012: Assembly.CodeBase and Assembly.EscapedCodeBase are obsolete

9/20/2022 • 2 minutes to read • [Edit Online](#)

The following APIs are marked as obsolete, starting in .NET 5. Using them in code generates warning `SYSLIB0012` at compile time.

- [Assembly.CodeBase](#)
- [Assembly.EscapedCodeBase](#)

Workarounds

Use [Assembly.Location](#) instead.

Suppress a warning

If you must use the obsolete APIs, you can suppress the warning in code or in your project file.

To suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the warning.

```
// Disable the warning.  
#pragma warning disable SYSLIB0012  
  
// Code that uses obsolete API.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB0012
```

To suppress all the `SYSLIB0012` warnings in your project, add a `<NoWarn>` property to your project file.

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    ...  
    <NoWarn>$(NoWarn);SYSLIB0012</NoWarn>  
  </PropertyGroup>  
</Project>
```

For more information, see [Suppress warnings](#).

SYSLIB0013: EscapeUriString is obsolete

9/20/2022 • 2 minutes to read • [Edit Online](#)

The [Uri.EscapeUriString\(String\)](#) API is marked as obsolete, starting in .NET 6. Using it in code generates warning `SYSLIB0013` at compile time.

`Uri.EscapeUriString(String)` can corrupt the Uri string in some cases.

For more information, see <https://github.com/dotnet/runtime/issues/31387>.

Workarounds

Use [Uri.EscapeDataString\(String\)](#) for query string components instead.

Suppress a warning

If you must use the obsolete APIs, you can suppress the warning in code or in your project file.

To suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the warning.

```
// Disable the warning.  
#pragma warning disable SYSLIB0013  
  
// Code that uses obsolete API.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB0013
```

To suppress all the `SYSLIB0013` warnings in your project, add a `<NoWarn>` property to your project file.

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    ...  
    <NoWarn>$(NoWarn);SYSLIB0013</NoWarn>  
  </PropertyGroup>  
</Project>
```

For more information, see [Suppress warnings](#).

SYSLIB0014: WebRequest, HttpWebRequest, ServicePoint, WebClient are obsolete

9/20/2022 • 2 minutes to read • [Edit Online](#)

The following APIs are marked as obsolete, starting in .NET 6. Using them in code generates warning `SYSLIB0014` at compile time.

- [WebRequest\(\)](#)
- [System.Net.WebRequest.Create](#)
- [System.Net.WebRequest.CreateHttp](#)
- [System.Net.WebRequest.CreateDefault\(Uri\)](#)
- [HttpWebRequest\(SerializationInfo, StreamingContext\)](#)
- [System.Net.ServicePointManager.FindServicePoint](#)
- [WebClient\(\)](#)

Workarounds

Use [HttpClient](#) instead.

Suppress a warning

If you must use the obsolete APIs, you can suppress the warning in code or in your project file.

To suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the warning.

```
// Disable the warning.  
#pragma warning disable SYSLIB0014  
  
// Code that uses obsolete API.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB0014
```

To suppress all the `SYSLIB0014` warnings in your project, add a `<NoWarn>` property to your project file.

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    ...  
    <NoWarn>$(NoWarn);SYSLIB0014</NoWarn>  
  </PropertyGroup>  
</Project>
```

For more information, see [Suppress warnings](#).

See also

- [WebRequest, WebClient, and ServicePoint are obsolete](#)

SYSLIB0015: DisablePrivateReflectionAttribute is obsolete

9/20/2022 • 2 minutes to read • [Edit Online](#)

Starting in .NET 6, the [System.Runtime.CompilerServices.DisablePrivateReflectionAttribute](#) type is marked as obsolete. This attribute has no effect in .NET Core 2.1 and later apps. Using it in code generates warning `SYSLIB0015` at compile time for .NET 6 and later apps.

For more information, see <https://github.com/dotnet/runtime/issues/11811>.

Workarounds

None.

Suppress a warning

If you must use the obsolete APIs, you can suppress the warning in code or in your project file.

To suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the warning.

```
// Disable the warning.  
#pragma warning disable SYSLIB0015  
  
// Code that uses obsolete API.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB0015
```

To suppress all the `SYSLIB0015` warnings in your project, add a `<NoWarn>` property to your project file.

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    ...  
    <NoWarn>$(NoWarn);SYSLIB0015</NoWarn>  
  </PropertyGroup>  
</Project>
```

For more information, see [Suppress warnings](#).

SYSLIB0016: GetContextInfo() is obsolete

9/20/2022 • 2 minutes to read • [Edit Online](#)

The [Graphics.GetContextInfo\(\)](#) method that takes no arguments is marked as obsolete, starting in .NET 6. Using it in code generates warning `SYSLIB0016` at compile time.

For more information, see <https://github.com/dotnet/runtime/issues/47880>.

Workarounds

For better performance and fewer allocations, use the [Graphics.GetContextInfo](#) overloads that accept arguments:

- [System.Drawing.Graphics.GetContextInfo\(PointF\)](#)
- [System.Drawing.Graphics.GetContextInfo\(PointF, Region\)](#)

Suppress a warning

If you must use the obsolete APIs, you can suppress the warning in code or in your project file.

To suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the warning.

```
// Disable the warning.  
#pragma warning disable SYSLIB0016  
  
// Code that uses obsolete API.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB0016
```

To suppress all the `SYSLIB0016` warnings in your project, add a `<NoWarn>` property to your project file.

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    ...  
    <NoWarn>$({NoWarn});SYSLIB0016</NoWarn>  
  </PropertyGroup>  
</Project>
```

For more information, see [Suppress warnings](#).

SYSLIB0017: Strong-name signing is not supported and throws PlatformNotSupportedException

9/20/2022 • 2 minutes to read • [Edit Online](#)

The following APIs are marked as obsolete, starting in .NET 6. Using them in code generates warning `SYSLIB0017` at compile time. These APIs throw a [PlatformNotSupportedException](#) at run time.

- [AssemblyName.KeyPair](#)
- [StrongNameKeyPair](#)

For more information, see <https://github.com/dotnet/runtime/issues/50529>.

Workarounds

None.

Suppress a warning

If you must use the obsolete APIs, you can suppress the warning in code or in your project file.

To suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the warning.

```
// Disable the warning.  
#pragma warning disable SYSLIB0017  
  
// Code that uses obsolete API.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB0017
```

To suppress all the `SYSLIB0017` warnings in your project, add a `<NoWarn>` property to your project file.

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    ...  
    <NoWarn>$({NoWarn});SYSLIB0017</NoWarn>  
  </PropertyGroup>  
</Project>
```

For more information, see [Suppress warnings](#).

SYSLIB0018: Reflection-only loading is not supported and throws PlatformNotSupportedException

9/20/2022 • 2 minutes to read • [Edit Online](#)

The following methods are marked as obsolete, starting in .NET 6. Calling them in code generates warning `SYSLIB0018` at compile time. These methods throw a [PlatformNotSupportedException](#) at run time.

- [Assembly.ReflectionOnlyLoad](#)
- [Assembly.ReflectionOnlyLoadFrom\(String\)](#)
- [Type.ReflectionOnlyGetType\(String, Boolean, Boolean\)](#)

Workarounds

Reflection-only loading is replaced by the metadata load-context in .NET Core and .NET 5+. For more information, see [How to: Inspect assembly contents using MetadataLoadContext](#).

Suppress a warning

If you must use the obsolete APIs, you can suppress the warning in code or in your project file.

To suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the warning.

```
// Disable the warning.  
#pragma warning disable SYSLIB0018  
  
// Code that uses obsolete API.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB0018
```

To suppress all the `SYSLIB0018` warnings in your project, add a `<NoWarn>` property to your project file.

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    ...  
    <NoWarn>$(<NoWarn>) ; SYSLIB0018</NoWarn>  
  </PropertyGroup>  
</Project>
```

For more information, see [Suppress warnings](#).

SYSLIB0019: Some RuntimeEnvironment APIs are obsolete

9/20/2022 • 2 minutes to read • [Edit Online](#)

The following APIs are marked as obsolete, starting in .NET 6. Using them in code generates warning `SYSLIB0019` at compile time.

- `RuntimeEnvironment.SystemConfigurationFile` property
- `RuntimeEnvironment.GetRuntimeInterfaceAsIntPtr(Guid, Guid)` method
- `RuntimeEnvironment.GetRuntimeInterfaceAsObject(Guid, Guid)` method

These APIs always throw a `PlatformNotSupportedException` at run time.

Workarounds

None.

Suppress a warning

If you must use the obsolete APIs, you can suppress the warning in code or in your project file.

To suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the warning.

```
// Disable the warning.  
#pragma warning disable SYSLIB0019  
  
// Code that uses obsolete API.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB0019
```

To suppress all the `SYSLIB0019` warnings in your project, add a `<NoWarn>` property to your project file.

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    ...  
    <NoWarn>$(NoWarn);SYSLIB0019</NoWarn>  
  </PropertyGroup>  
</Project>
```

For more information, see [Suppress warnings](#).

SYSLIB0020: IgnoreNullValues is obsolete

9/20/2022 • 2 minutes to read • [Edit Online](#)

The `JsonSerializerOptions.IgnoreNullValues` property is marked as obsolete, starting in .NET 6. Using it in code generates warning `SYSLIB0020` at compile time.

Workarounds

To ignore null values when serializing, set `DefaultIgnoreCondition` to `JsonIgnoreCondition.WhenWritingNull`. For more information, see <https://github.com/dotnet/runtime/issues/39152>.

Suppress a warning

If you must use the obsolete APIs, you can suppress the warning in code or in your project file.

To suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the warning.

```
// Disable the warning.  
#pragma warning disable SYSLIB0020  
  
// Code that uses obsolete API.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB0020
```

To suppress all the `SYSLIB0020` warnings in your project, add a `<NoWarn>` property to your project file.

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    ...  
    <NoWarn>$(NoWarn);SYSLIB0020</NoWarn>  
  </PropertyGroup>  
</Project>
```

For more information, see [Suppress warnings](#).

SYSLIB0021: Derived cryptographic types are obsolete

9/20/2022 • 2 minutes to read • [Edit Online](#)

The following derived cryptographic types are marked as obsolete, starting in .NET 6. Using them in code generates warning `SYSLIB0021` at compile time.

- [System.Security.Cryptography.AesCryptoServiceProvider](#)
- [System.Security.Cryptography.AesManaged](#)
- [System.Security.Cryptography.DESCryptoServiceProvider](#)
- [System.Security.Cryptography.MD5CryptoServiceProvider](#)
- [System.Security.Cryptography.RC2CryptoServiceProvider](#)
- [System.Security.Cryptography.SHA1CryptoServiceProvider](#)
- [System.Security.Cryptography.SHA1Managed](#)
- [System.Security.Cryptography.SHA256Managed](#)
- [System.Security.Cryptography.SHA256CryptoServiceProvider](#)
- [System.Security.Cryptography.SHA384Managed](#)
- [System.Security.Cryptography.SHA384CryptoServiceProvider](#)
- [System.Security.Cryptography.SHA512Managed](#)
- [System.Security.Cryptography.SHA512CryptoServiceProvider](#)
- [System.Security.Cryptography.TripleDESCryptoServiceProvider](#)

Workarounds

Use the `Create` method on the base type instead. For example, use [TripleDES.Create](#) instead of [TripleDESCryptoServiceProvider](#).

Suppress a warning

If you must use the obsolete APIs, you can suppress the warning in code or in your project file.

To suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the warning.

```
// Disable the warning.  
#pragma warning disable SYSLIB0021  
  
// Code that uses obsolete API.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB0021
```

To suppress all the `SYSLIB0021` warnings in your project, add a `<NoWarn>` property to your project file.

```
<Project Sdk="Microsoft.NET.Sdk">
<PropertyGroup>
...
<NoWarn>$(NoWarn);SYSLIB0021</NoWarn>
</PropertyGroup>
</Project>
```

For more information, see [Suppress warnings](#).

SYSLIB0022: The Rijndael and RijndaelManaged types are obsolete

9/20/2022 • 2 minutes to read • [Edit Online](#)

The [Rijndael](#) and [RijndaelManaged](#) types are marked as obsolete, starting in .NET 6. Using them in code generates warning `SYSLIB0022` at compile time.

Workarounds

Use [System.Security.Cryptography.Aes](#) instead.

Suppress a warning

If you must use the obsolete APIs, you can suppress the warning in code or in your project file.

To suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the warning.

```
// Disable the warning.  
#pragma warning disable SYSLIB0022  
  
// Code that uses obsolete API.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB0022
```

To suppress all the `SYSLIB0022` warnings in your project, add a `<NoWarn>` property to your project file.

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    ...  
    <NoWarn>$(NoWarn);SYSLIB0022</NoWarn>  
  </PropertyGroup>  
</Project>
```

For more information, see [Suppress warnings](#).

SYSLIB0023: RNGCryptoServiceProvider is obsolete

9/20/2022 • 2 minutes to read • [Edit Online](#)

`RNGCryptoServiceProvider` is marked as obsolete, starting in .NET 6. Using it in code generates warning `SYSLIB0023` at compile time.

Workarounds

To generate a random number, use one of the `RandomNumberGenerator` methods instead, for example, `RandomNumberGenerator.GetInt32(Int32)`.

Suppress a warning

If you must use the obsolete APIs, you can suppress the warning in code or in your project file.

To suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the warning.

```
// Disable the warning.  
#pragma warning disable SYSLIB0023  
  
// Code that uses obsolete API.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB0023
```

To suppress all the `SYSLIB0023` warnings in your project, add a `<NoWarn>` property to your project file.

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    ...  
    <NoWarn>$(NoWarn);SYSLIB0023</NoWarn>  
  </PropertyGroup>  
</Project>
```

For more information, see [Suppress warnings](#).

SYSLIB0024: Creating and unloading AppDomains is not supported and throws an exception

9/20/2022 • 2 minutes to read • [Edit Online](#)

The `AppDomain.CreateDomain(String)` and `AppDomain.Unload(AppDomain)` methods are marked as obsolete, starting in .NET 6. Using them in code generates warning `SYSLIB0024` at compile time and throws an exception at run time.

Workarounds

None.

Suppress a warning

If you must use the obsolete APIs, you can suppress the warning in code or in your project file.

To suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the warning.

```
// Disable the warning.  
#pragma warning disable SYSLIB0024  
  
// Code that uses obsolete API.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB0024
```

To suppress all the `SYSLIB0024` warnings in your project, add a `<NoWarn>` property to your project file.

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    ...  
    <NoWarn>$({NoWarn});SYSLIB0024</NoWarn>  
  </PropertyGroup>  
</Project>
```

For more information, see [Suppress warnings](#).

SYSLIB0025: SuppressIldasmAttribute is obsolete

9/20/2022 • 2 minutes to read • [Edit Online](#)

The [SuppressIldasmAttribute](#) type is marked as obsolete, starting in .NET 6. Using it in code generates warning `SYSLIB0025` at compile time. [IL Disassembler \(ildasm.exe\)](#) no longer supports this attribute.

Workarounds

None.

Suppress a warning

If you must use the obsolete APIs, you can suppress the warning in code or in your project file.

To suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the warning.

```
// Disable the warning.  
#pragma warning disable SYSLIB0025  
  
// Code that uses obsolete API.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB0025
```

To suppress all the `SYSLIB0025` warnings in your project, add a `<NoWarn>` property to your project file.

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    ...  
    <NoWarn>$(NoWarn);SYSLIB0025</NoWarn>  
  </PropertyGroup>  
</Project>
```

For more information, see [Suppress warnings](#).

SYSLIB0026: X509Certificate and X509Certificate2 are immutable

9/20/2022 • 2 minutes to read • [Edit Online](#)

The following mutable x509 certificate APIs are marked as obsolete, starting in .NET 6. Using these APIs in code generates warning `SYSLIB0026` at compile time.

- [X509Certificate\(\)](#)
- [X509Certificate.Import](#)
- [X509Certificate2\(\)](#)
- [X509Certificate2.Import](#)

Workarounds

Create a new instance of `X509Certificate` and `X509Certificate2` using a constructor overload that accepts the certificate as input. For example:

```
// Change this:  
cert.Import("/path/to/certificate.crt");  
  
// To this:  
cert.Dispose();  
cert = new X509Certificate2("/path/to/certificate.crt");
```

Suppress a warning

If you must use the obsolete APIs, you can suppress the warning in code or in your project file.

To suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the warning.

```
// Disable the warning.  
#pragma warning disable SYSLIB0026  
  
// Code that uses obsolete API.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB0026
```

To suppress all the `SYSLIB0026` warnings in your project, add a `<NoWarn>` property to your project file.

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    ...  
    <NoWarn>$(NoWarn);SYSLIB0026</NoWarn>  
  </PropertyGroup>  
</Project>
```

For more information, see [Suppress warnings](#).

SYSLIB0027: PublicKey.Key is obsolete

9/20/2022 • 2 minutes to read • [Edit Online](#)

The [PublicKey.Key](#) property is marked as obsolete, starting in .NET 6. Using this API in code generates warning `SYSLIB0027` at compile time.

Workarounds

Use the appropriate method to get the public key, such as [GetRSAPublicKey\(\)](#).

Suppress a warning

If you must use the obsolete APIs, you can suppress the warning in code or in your project file.

To suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the warning.

```
// Disable the warning.  
#pragma warning disable SYSLIB0027  
  
// Code that uses obsolete API.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB0027
```

To suppress all the `SYSLIB0027` warnings in your project, add a `<NoWarn>` property to your project file.

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    ...  
    <NoWarn>$(<NoWarn>) ; SYSLIB0027</NoWarn>  
  </PropertyGroup>  
</Project>
```

For more information, see [Suppress warnings](#).

SYSLIB0028: X509Certificate2.PrivateKey is obsolete

9/20/2022 • 2 minutes to read • [Edit Online](#)

The [X509Certificate2.PrivateKey](#) property is marked as obsolete, starting in .NET 6. Using this API in code generates warning `SYSLIB0028` at compile time.

Workarounds

Use the appropriate method to get the private key, such as [RSACertificateExtensions.GetRSAPrivateKey\(X509Certificate2\)](#), or use the [X509Certificate2.CopyWithPrivateKey\(ECDiffieHellman\)](#) method to create a new instance with a private key.

Suppress a warning

If you must use the obsolete APIs, you can suppress the warning in code or in your project file.

To suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the warning.

```
// Disable the warning.  
#pragma warning disable SYSLIB0028  
  
// Code that uses obsolete API.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB0028
```

To suppress all the `SYSLIB0028` warnings in your project, add a `<NoWarn>` property to your project file.

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    ...  
    <NoWarn>$(NoWarn);SYSLIB0028</NoWarn>  
  </PropertyGroup>  
</Project>
```

For more information, see [Suppress warnings](#).

SYSLIB0029: ProduceLegacyHmacValues is obsolete

9/20/2022 • 2 minutes to read • [Edit Online](#)

The following properties are marked as obsolete, starting in .NET 6:

- [HMACSHA384.ProduceLegacyHmacValues](#)
- [HMACSHA512.ProduceLegacyHmacValues](#)

Using these APIs in code generates warning `SYSLIB0029` at compile time. Producing legacy HMAC values is no longer supported.

Workarounds

None.

Suppress a warning

If you must use the obsolete APIs, you can suppress the warning in code or in your project file.

To suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the warning.

```
// Disable the warning.  
#pragma warning disable SYSLIB0029  
  
// Code that uses obsolete API.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB0029
```

To suppress all the `SYSLIB0029` warnings in your project, add a `<NoWarn>` property to your project file.

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    ...  
    <NoWarn>$(NoWarn);SYSLIB0029</NoWarn>  
  </PropertyGroup>  
</Project>
```

For more information, see [Suppress warnings](#).

SYSLIB0030: HMACSHA1 always uses the algorithm implementation provided by the platform

9/20/2022 • 2 minutes to read • [Edit Online](#)

The [HMACSHA1\(Byte\[\], Boolean\)](#) constructor is marked as obsolete, starting in .NET 6. Using this API in code generates warning `SYSLIB0030` at compile time.

Workarounds

Use a constructor without the `useManagedSha1` parameter.

Suppress a warning

If you must use the obsolete APIs, you can suppress the warning in code or in your project file.

To suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the warning.

```
// Disable the warning.  
#pragma warning disable SYSLIB0030  
  
// Code that uses obsolete API.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB0030
```

To suppress all the `SYSLIB0030` warnings in your project, add a `<NoWarn>` property to your project file.

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    ...  
    <NoWarn>$(NoWarn);SYSLIB0030</NoWarn>  
  </PropertyGroup>  
</Project>
```

For more information, see [Suppress warnings](#).

SYSLIB0031: EncodeOID is obsolete

9/20/2022 • 2 minutes to read • [Edit Online](#)

The [CryptoConfig.EncodeOID\(String\)](#) method is marked as obsolete, starting in .NET 6. Using this API in code generates warning `SYSLIB0031` at compile time and throws a [PlatformNotSupportedException](#) exception at run time.

Workarounds

Use the ASN.1 functionality provided in [System.Formats.Asn1](#).

Suppress a warning

If you must use the obsolete APIs, you can suppress the warning in code or in your project file.

To suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the warning.

```
// Disable the warning.  
#pragma warning disable SYSLIB0031  
  
// Code that uses obsolete API.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB0031
```

To suppress all the `SYSLIB0031` warnings in your project, add a `<NoWarn>` property to your project file.

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    ...  
    <NoWarn>$(NoWarn);SYSLIB0031</NoWarn>  
  </PropertyGroup>  
</Project>
```

For more information, see [Suppress warnings](#).

SYSLIB0032: Recovery from corrupted process state exceptions is not supported

9/20/2022 • 2 minutes to read • [Edit Online](#)

Recovery from corrupted process state exceptions is not supported, and the [HandleProcessCorruptedStateExceptionsAttribute](#) type is marked as obsolete, starting in .NET 6. Using this API in code generates warning `SYSLIB0032` at compile time.

Suppress a warning

If you must use the obsolete APIs, you can suppress the warning in code or in your project file.

To suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the warning.

```
// Disable the warning.  
#pragma warning disable SYSLIB0032  
  
// Code that uses obsolete API.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB0032
```

To suppress all the `SYSLIB0032` warnings in your project, add a `<NoWarn>` property to your project file.

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    ...  
    <NoWarn>$(NoWarn);SYSLIB0032</NoWarn>  
  </PropertyGroup>  
</Project>
```

For more information, see [Suppress warnings](#).

SYSLIB0033: Rfc2898DeriveBytes.CryptDeriveKey is obsolete

9/20/2022 • 2 minutes to read • [Edit Online](#)

The `Rfc2898DeriveBytes.CryptDeriveKey(String, String, Int32, Byte[])` method is marked as obsolete, starting in .NET 6. Using this API in code generates warning `SYSLIB0033` at compile time.

Workaround

Use `PasswordDeriveBytes.CryptDeriveKey(String, String, Int32, Byte[])` instead.

Suppress a warning

If you must use the obsolete APIs, you can suppress the warning in code or in your project file.

To suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the warning.

```
// Disable the warning.  
#pragma warning disable SYSLIB0033  
  
// Code that uses obsolete API.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB0033
```

To suppress all the `SYSLIB0033` warnings in your project, add a `<NoWarn>` property to your project file.

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    ...  
    <NoWarn>$(NoWarn);SYSLIB0033</NoWarn>  
  </PropertyGroup>  
</Project>
```

For more information, see [Suppress warnings](#).

SYSLIB0034: CmsSigner(CspParameters) constructor is obsolete

9/20/2022 • 2 minutes to read • [Edit Online](#)

The [CmsSigner\(CspParameters\)](#) constructor is marked as obsolete, starting in .NET 6. Using this API in code generates warning `SYSLIB0034` at compile time.

Workaround

Use an alternative constructor.

Suppress a warning

If you must use the obsolete APIs, you can suppress the warning in code or in your project file.

To suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the warning.

```
// Disable the warning.  
#pragma warning disable SYSLIB0034  
  
// Code that uses obsolete API.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB0034
```

To suppress all the `SYSLIB0034` warnings in your project, add a `<NoWarn>` property to your project file.

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    ...  
    <NoWarn>$(NoWarn);SYSLIB0034</NoWarn>  
  </PropertyGroup>  
</Project>
```

For more information, see [Suppress warnings](#).

SYSLIB0035: ComputeCounterSignature without specifying a CmsSigner is obsolete

9/20/2022 • 2 minutes to read • [Edit Online](#)

The [SignerInfo.ComputeCounterSignature\(\)](#) method is marked as obsolete, starting in .NET 6. Using this API in code generates warning `SYSLIB0035` at compile time.

Workaround

Use the overload that accepts a [CmsSigner](#), that is, [SignerInfo.ComputeCounterSignature\(CmsSigner\)](#).

Suppress a warning

If you must use the obsolete APIs, you can suppress the warning in code or in your project file.

To suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the warning.

```
// Disable the warning.  
#pragma warning disable SYSLIB0035  
  
// Code that uses obsolete API.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB0035
```

To suppress all the `SYSLIB0035` warnings in your project, add a `<NoWarn>` property to your project file.

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    ...  
    <NoWarn>$(NoWarn);SYSLIB0035</NoWarn>  
  </PropertyGroup>  
</Project>
```

For more information, see [Suppress warnings](#).

SYSLIB0036: Regex.CompileToAssembly is obsolete

9/20/2022 • 2 minutes to read • [Edit Online](#)

The [Regex.CompileToAssembly](#) method is marked as obsolete, starting in .NET 7. Using this API in code generates warning `SYSLIB0036` at compile time.

In .NET 5, .NET 6, and all versions of .NET Core, [Regex.CompileToAssembly](#) throws a [PlatformNotSupportedException](#). In .NET Framework, [Regex.CompileToAssembly](#) allows a regular expression instance to be compiled into an assembly.

Workaround

Use the `RegexGeneratorAttribute` feature, which invokes a regular expression source generator. At compile time, the source generator produces an API specific to a regular expression pattern and its options.

Suppress a warning

If you must use the obsolete APIs, you can suppress the warning in code or in your project file.

To suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the warning.

```
// Disable the warning.  
#pragma warning disable SYSLIB0036  
  
// Code that uses obsolete API.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB0036
```

To suppress all the `SYSLIB0036` warnings in your project, add a `<NoWarn>` property to your project file.

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    ...  
    <NoWarn>$(NoWarn);SYSLIB0036</NoWarn>  
  </PropertyGroup>  
</Project>
```

For more information, see [Suppress warnings](#).

SYSLIB0037: AssemblyName members HashAlgorithm, ProcessorArchitecture, and VersionCompatibility are obsolete

9/20/2022 • 2 minutes to read • [Edit Online](#)

The following [System.Reflection.AssemblyName](#) properties are marked as obsolete, starting in .NET 7. Using these APIs in code generates warning `SYSLIB0037` at compile time.

- [HashAlgorithm](#)
- [ProcessorArchitecture](#)
- [VersionCompatibility](#)

These properties are not a proper part of an [AssemblyName](#) instance. They don't roundtrip through [AssemblyName](#) string representation, and they are ignored by the assembly loader in .NET Core.

Workaround

Don't use these members in scenarios where it was expected for the values to be round-tripped through the string representation of the [AssemblyName](#).

Suppress a warning

If you must use the obsolete APIs, you can suppress the warning in code or in your project file.

To suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the warning.

```
// Disable the warning.  
#pragma warning disable SYSLIB0037  
  
// Code that uses obsolete API.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB0037
```

To suppress all the `SYSLIB0037` warnings in your project, add a `<NoWarn>` property to your project file.

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    ...  
    <NoWarn>$({NoWarn});SYSLIB0037</NoWarn>  
  </PropertyGroup>  
</Project>
```

For more information, see [Suppress warnings](#).

SYSLIB0038: SerializationFormat.Binary is obsolete

9/20/2022 • 2 minutes to read • [Edit Online](#)

`SerializationFormat.Binary` is marked as obsolete, starting in .NET 7. Using this API in code generates warning `SYSLIB0038` at compile time.

Workaround

If your code uses `SerializationFormat.Binary`, switch to using `SerializationFormat.Xml` or use another method of serialization.

Otherwise, you can set the `Switch.System.Data.AllowUnsafeSerializationFormatBinary` `AppContext` switch. This switch lets you opt in to allowing the use of `SerializationFormat.Binary`, so that code can work as before. However, this switch will be removed in .NET 8. For information about setting the switch, see [AppContext for library consumers](#).

Suppress a warning

If you must use the obsolete APIs, you can suppress the warning in code or in your project file.

To suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the warning.

```
// Disable the warning.  
#pragma warning disable SYSLIB0038  
  
// Code that uses obsolete API.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB0038
```

To suppress all the `SYSLIB0038` warnings in your project, add a `<NoWarn>` property to your project file.

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    ...  
    <NoWarn>$(<NoWarn>) ; SYSLIB0038</NoWarn>  
  </PropertyGroup>  
</Project>
```

For more information, see [Suppress warnings](#).

See also

- [SerializationFormat.Binary is obsolete](#)

SYSLIB0039: SslProtocols.Tls and SslProtocols.Tls11 are obsolete

9/20/2022 • 2 minutes to read • [Edit Online](#)

`SslProtocols.Tls` and `SslProtocols.Tls11` are marked as obsolete, starting in .NET 7. Using these enumeration fields in code generates warning `SYSLIB0039` at compile time.

Workaround

Use a higher TLS protocol version, or use `SslProtocols.None` to defer to system defaults.

Suppress a warning

If you must use the obsolete APIs, you can suppress the warning in code or in your project file.

To suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the warning.

```
// Disable the warning.  
#pragma warning disable SYSLIB0039  
  
// Code that uses obsolete API.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB0039
```

To suppress all the `SYSLIB0039` warnings in your project, add a `<NoWarn>` property to your project file.

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    ...  
    <NoWarn>$(NoWarn);SYSLIB0039</NoWarn>  
  </PropertyGroup>  
</Project>
```

For more information, see [Suppress warnings](#).

SYSLIB0040: EncryptionPolicy.NoEncryption and EncryptionPolicy.AllowNoEncryption are obsolete

9/20/2022 • 2 minutes to read • [Edit Online](#)

[EncryptionPolicy.NoEncryption](#) and [EncryptionPolicy.AllowNoEncryption](#) are marked as obsolete, starting in .NET 7. Using these enumeration fields in code generates warning `SYSLIB0040` at compile time.

Older SSL/TLS versions permitted no encryption, and while it may be useful for debugging, it shouldn't be used in production. In addition, NULL encryption is no longer available as of TLS 1.3.

Workaround

N/A

Suppress a warning

If you must use the obsolete APIs, you can suppress the warning in code or in your project file.

To suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the warning.

```
// Disable the warning.  
#pragma warning disable SYSLIB0040  
  
// Code that uses obsolete API.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB0040
```

To suppress all the `SYSLIB0040` warnings in your project, add a `<NoWarn>` property to your project file.

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    ...  
    <NoWarn>$(NoWarn);SYSLIB0040</NoWarn>  
  </PropertyGroup>  
</Project>
```

For more information, see [Suppress warnings](#).

SYSLIB0041: Some Rfc2898DeriveBytes constructors are obsolete

9/20/2022 • 2 minutes to read • [Edit Online](#)

The following [Rfc2898DeriveBytes](#) constructors are obsolete, starting in .NET 7. Using them in code generates warning `SYSLIB0041` at compile time.

- [Rfc2898DeriveBytes\(String, Byte\[\]\)](#)
- [Rfc2898DeriveBytes\(String, Int32\)](#)
- [Rfc2898DeriveBytes\(Byte\[\], Byte\[\], Int32\)](#)
- [Rfc2898DeriveBytes\(String, Byte\[\], Int32\)](#)
- [Rfc2898DeriveBytes\(String, Int32, Int32\)](#)

These overloads default the hash algorithm or number of iterations, and the defaults are no longer considered secure. These are all of the constructors that were available in .NET 4.7.1 and earlier versions. Going forward, you should only use the newer constructors.

Workaround

Use a different constructor overload where you can explicitly specify the iteration count (the default is 1000) and hash algorithm name (the default is [HashAlgorithmName.SHA1](#)).

If you're using the default iteration count or default hash algorithm, consider moving to more secure values—that is, a larger iteration count or a newer hash algorithm.

Suppress a warning

If you must use the obsolete APIs, you can suppress the warning in code or in your project file.

To suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the warning.

```
// Disable the warning.  
#pragma warning disable SYSLIB0041  
  
// Code that uses obsolete API.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB0041
```

To suppress all the `SYSLIB0041` warnings in your project, add a `<NoWarn>` property to your project file.

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    ...  
    <NoWarn>$(NoWarn);SYSLIB0041</NoWarn>  
  </PropertyGroup>  
</Project>
```

For more information, see [Suppress warnings](#).

SYSLIB0042: FromXmlString and ToXmlString on ECC types are obsolete

9/20/2022 • 2 minutes to read • [Edit Online](#)

The `FromXmlString` and `ToXmlString` methods that are on elliptic curve cryptography (ECC) types are obsolete, starting in .NET 7. Using them in code generates warning `SYSLIB0042` at compile time. They were never implemented and always threw a `PlatformNotSupportedException` exception. The obsoletion affects the following methods:

- `ECDiffieHellmanCng.FromXmlString(String, ECKeyXmlFormat)`
- `ECDiffieHellmanCng.ToXmlString(ECKeyXmlFormat)`
- `ECDiffieHellmanCngPublicKey.FromXmlString(String)`
- `ECDiffieHellmanCngPublicKey.ToString()`
- `ECDiffieHellmanPublicKey.ToString()`
- `ECDsaCng.FromXmlString(String, ECKeyXmlFormat)`
- `ECDsaCng.ToString(ECKeyXmlFormat)`

Workaround

Use a standard data format for exchanging elliptic curve (EC) keys.

Instead of `ToXmlString`, use `ExportSubjectPublicKeyInfo` or `ExportPkcs8PrivateKey` depending on whether you want the public or private key.

Instead of `FromXmlString`, use `ImportSubjectPublicKeyInfo` or `ImportPkcs8PrivateKey` depending on whether you want to import a public or private key.

Suppress a warning

If you must use the obsolete APIs, you can suppress the warning in code or in your project file.

To suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the warning.

```
// Disable the warning.  
#pragma warning disable SYSLIB0042  
  
// Code that uses obsolete API.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB0042
```

To suppress all the `SYSLIB0042` warnings in your project, add a `<NoWarn>` property to your project file.

```
<Project Sdk="Microsoft.NET.Sdk">
<PropertyGroup>
...
<NoWarn>$(NoWarn);SYSLIB0042</NoWarn>
</PropertyGroup>
</Project>
```

For more information, see [Suppress warnings](#).

SYSLIB0043: ECDiffieHellmanPublicKey.ToArray is obsolete

9/20/2022 • 2 minutes to read • [Edit Online](#)

The following methods are obsolete, starting in .NET 7. Using them in code generates warning `SYSLIB0043` at compile time.

- [ECDiffieHellmanPublicKey.ToArray\(\)](#)
- [ECDiffieHellmanPublicKey\(Byte\[\]\)](#) (constructor)

The [ECDiffieHellmanPublicKey.ToArray\(\)](#) method does not have an implied file format. Also, for the built-in implementations, it throws [PlatformNotSupportedException](#) on all non-Windows operating systems. Since [ECDiffieHellmanPublicKey](#) also has a standard format export (via the [ExportSubjectPublicKeyInfo\(\)](#) method), the older member has been obsoleted.

Workaround

If you're exporting the public key value, use the [ExportSubjectPublicKeyInfo\(\)](#) method instead.

For new derived types (or existing derived types that don't currently call the [ECDiffieHellmanPublicKey\(Byte\[\]\)](#) constructor), don't call the protected [ECDiffieHellmanPublicKey\(Byte\[\]\)](#) constructor, and either override [ToArray\(\)](#) to throw an exception, or accept the default behavior of returning an empty array.

For existing derived types that already call the protected [ECDiffieHellmanPublicKey\(Byte\[\]\)](#) constructor, continue calling the constructor and suppress the `SYSLIB0043` warning.

Suppress a warning

If you must use the obsolete APIs, you can suppress the warning in code or in your project file.

To suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the warning.

```
// Disable the warning.  
#pragma warning disable SYSLIB0043  
  
// Code that uses obsolete API.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB0043
```

To suppress all the `SYSLIB0043` warnings in your project, add a `<NoWarn>` property to your project file.

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    ...  
    <NoWarn>$({NoWarn});SYSLIB0043</NoWarn>  
  </PropertyGroup>  
</Project>
```

For more information, see [Suppress warnings](#).

SYSLIB0047: XmlSecureResolver is obsolete

9/20/2022 • 2 minutes to read • [Edit Online](#)

The `System.Xml.XmlSecureResolver` type is obsolete, starting in .NET 7. Using it in code generates warning `SYSLIB0047` at compile time.

Workaround

Consider using the static property `XmlResolver.ThrowingResolver` instead. This property provides an `XmlResolver` instance that forbids resolution of external XML resources.

```
using System.Xml;

XmlResolver resolver = XmlResolver.ThrowingResolver;
```

Suppress a warning

If you must use the obsolete APIs, you can suppress the warning in code or in your project file.

To suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the warning.

```
// Disable the warning.
#pragma warning disable SYSLIB0047

// Code that uses obsolete API.
// ...

// Re-enable the warning.
#pragma warning restore SYSLIB0047
```

To suppress all the `SYSLIB0047` warnings in your project, add a `<NoWarn>` property to your project file.

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    ...
    <NoWarn>$({NoWarn});SYSLIB0047</NoWarn>
  </PropertyGroup>
</Project>
```

For more information, see [Suppress warnings](#).

See also

- [XmlSecureResolver is obsolete](#)

Source-generator diagnostics in .NET 6+

9/20/2022 • 2 minutes to read • [Edit Online](#)

If your .NET 6+ project references a package that enables source generation of code, for example, a logging solution, then the analyzers that are specific to source generation will run at compile time. This article lists the compiler diagnostics related to source-generated code.

If you encounter one of these build warnings or errors, follow the specific guidance provided for the diagnostic ID listed in the [Reference](#) section. Warnings can also be suppressed using the specific `SYSLIB1XXX` diagnostic ID value. For more information, see [Suppress warnings](#).

Analyzer warnings

The diagnostic ID values reserved for source-generated code analyzer warnings are `SYSLIB1001` through `SYSLIB1999`.

Reference

The following table provides an index to the `SYSLIB1XXX` diagnostics in .NET 6 and later versions.

DIAGNOSTIC ID	DESCRIPTION
<code>SYSLIB1001</code>	Logging method names cannot start with <code>_</code>
<code>SYSLIB1002</code>	Don't include log level parameters as templates in the logging message
<code>SYSLIB1003</code>	<code>InvalidLoggingMethodNameTitle</code>
<code>SYSLIB1005</code>	Could not find a required type definition
<code>SYSLIB1006</code>	Multiple logging methods cannot use the same event ID within a class
<code>SYSLIB1007</code>	Logging methods must return <code>void</code>
<code>SYSLIB1008</code>	One of the arguments to a logging method must implement the <code>Microsoft.Extensions.Logging.ILogger</code> interface
<code>SYSLIB1009</code>	Logging methods must be <code>static</code>
<code>SYSLIB1010</code>	Logging methods must be <code>partial</code>
<code>SYSLIB1011</code>	Logging methods cannot be generic
<code>SYSLIB1012</code>	Redundant qualifier in logging message
<code>SYSLIB1013</code>	Don't include exception parameters as templates in the logging message

DIAGNOSTIC ID	DESCRIPTION
SYSLIB1014	Logging template has no corresponding method argument
SYSLIB1015	Argument is not referenced from the logging message
SYSLIB1016	Logging methods cannot have a body
SYSLIB1017	A LogLevel value must be supplied in the <code>LoggerMessage</code> attribute or as a parameter to the logging method
SYSLIB1018	Don't include logger parameters as templates in the logging message
SYSLIB1019	Couldn't find a field of type <code>Microsoft.Extensions.Logging.ILogger</code>
SYSLIB1020	Found multiple fields of type <code>Microsoft.Extensions.Logging.ILogger</code>
SYSLIB1021	Multiple message-template item names differ only by case
SYSLIB1022	Can't have malformed format strings (for example, dangling curly braces)
SYSLIB1023	Generating more than six arguments is not supported
SYSLIB1030	The <code>System.Text.Json</code> source generator did not generate serialization metadata for type
SYSLIB1031	The <code>System.Text.Json</code> source generator encountered a duplicate <code>JsonTypeInfo</code> property name
SYSLIB1032	The <code>System.Text.Json</code> source generator encountered a context class that is not partial
SYSLIB1033	The <code>System.Text.Json</code> source generator encountered a type that has multiple <code>[JsonConstructor]</code> annotations
SYSLIB1035	The <code>System.Text.Json</code> source generator encountered a type that has multiple <code>[JsonExtensionData]</code> annotations
SYSLIB1036	The <code>System.Text.Json</code> source generator encountered an invalid <code>[JsonExtensionData]</code> annotation
SYSLIB1037	The <code>System.Text.Json</code> source generator encountered a type with init-only properties for which deserialization is not supported
SYSLIB1038	The <code>System.Text.Json</code> source generator encountered a property annotated with <code>[JsonIgnore]</code> that has inaccessible accessors

Suppress warnings

It's recommended that you use one of the workarounds when possible. However, if you cannot change your code, you can suppress the warning through a `#pragma` directive or a `<NoWarn>` project setting. If the `SYSLIB1XXX` source generator diagnostic doesn't surface as an error, you can suppress the warning in code or in your project file.

To suppress the warnings in code:

```
// Disable the warning.  
#pragma warning disable SYSLIB1006  
  
// Code that generates compiler diagnostic.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB1006
```

To suppress the warnings in a project file:

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    <TargetFramework>net6.0</TargetFramework>  
    <!-- Nowarn below suppresses SYSLIB1002 project-wide -->  
    <NoWarn>$(NoWarn);SYSLIB1002</NoWarn>  
    <!-- To suppress multiple warnings, you can use multiple Nowarn elements -->  
    <NoWarn>$(NoWarn);SYSLIB1002</NoWarn>  
    <NoWarn>$(NoWarn);SYSLIB1006</NoWarn>  
    <!-- Alternatively, you can suppress multiple warnings by using a semicolon-delimited list -->  
    <NoWarn>$(NoWarn);SYSLIB1002;SYSLIB1006;SYSLIB1007</NoWarn>  
  </PropertyGroup>  
</Project>
```

SYSLIB1001: Logging method names can't start with an underscore

9/20/2022 • 2 minutes to read • [Edit Online](#)

The name of a method annotated with the `LoggerMessageAttribute` starts with an underscore character. This is not allowed, as it may result in conflicting symbol names with respect to the automatically generated code.

Workarounds

Choose a different method name that doesn't start with an underscore.

Suppress warnings

It's recommended that you use one of the workarounds when possible. However, if you cannot change your code, you can suppress the warning through a `#pragma` directive or a `<NoWarn>` project setting. If the `SYSLIB1XXX` source generator diagnostic doesn't surface as an error, you can suppress the warning in code or in your project file.

To suppress the warnings in code:

```
// Disable the warning.  
#pragma warning disable SYSLIB1006  
  
// Code that generates compiler diagnostic.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB1006
```

To suppress the warnings in a project file:

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    <TargetFramework>net6.0</TargetFramework>  
    <!-- Nowarn below suppresses SYSLIB1002 project-wide -->  
    <NoWarn>$(NoWarn);SYSLIB1002</NoWarn>  
    <!-- To suppress multiple warnings, you can use multiple Nowarn elements -->  
    <NoWarn>$(NoWarn);SYSLIB1002</NoWarn>  
    <NoWarn>$(NoWarn);SYSLIB1006</NoWarn>  
    <!-- Alternatively, you can suppress multiple warnings by using a semicolon-delimited list -->  
    <NoWarn>$(NoWarn);SYSLIB1002;SYSLIB1006;SYSLIB1007</NoWarn>  
  </PropertyGroup>  
</Project>
```

SYSLIB1002: Don't include log level parameters as templates in the logging message

9/20/2022 • 2 minutes to read • [Edit Online](#)

The first log-level argument to a logging method is referenced as a template in the logging message. This is not necessary, since the first log level is passed down to the logging infrastructure explicitly. It doesn't need to be repeated in the logging message.

Workarounds

Remove the template that references the log-level argument from the logging message.

Suppress warnings

It's recommended that you use one of the workarounds when possible. However, if you cannot change your code, you can suppress the warning through a `#pragma` directive or a `<NoWarn>` project setting. If the `SYSLIB1XXX` source generator diagnostic doesn't surface as an error, you can suppress the warning in code or in your project file.

To suppress the warnings in code:

```
// Disable the warning.  
#pragma warning disable SYSLIB1006  
  
// Code that generates compiler diagnostic.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB1006
```

To suppress the warnings in a project file:

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    <TargetFramework>net6.0</TargetFramework>  
    <!-- Nowarn below suppresses SYSLIB1002 project-wide -->  
    <NoWarn>$(NoWarn);SYSLIB1002</NoWarn>  
    <!-- To suppress multiple warnings, you can use multiple Nowarn elements -->  
    <NoWarn>$(NoWarn);SYSLIB1002</NoWarn>  
    <NoWarn>$(NoWarn);SYSLIB1006</NoWarn>  
    <!-- Alternatively, you can suppress multiple warnings by using a semicolon-delimited list -->  
    <NoWarn>$(NoWarn);SYSLIB1002;SYSLIB1006;SYSLIB1007</NoWarn>  
  </PropertyGroup>  
</Project>
```

SYSLIB1003: Logging method parameter names can't start with an underscore

9/20/2022 • 2 minutes to read • [Edit Online](#)

The name of a parameter of a method annotated with the `LoggerMessageAttribute` starts with an underscore character. This is not allowed as it may result in conflicting symbol names with respect to the automatically generated code.

Workarounds

Choose a different parameter name that doesn't start with an underscore.

Suppress warnings

It's recommended that you use one of the workarounds when possible. However, if you cannot change your code, you can suppress the warning through a `#pragma` directive or a `<NoWarn>` project setting. If the `SYSLIB1XXX` source generator diagnostic doesn't surface as an error, you can suppress the warning in code or in your project file.

To suppress the warnings in code:

```
// Disable the warning.  
#pragma warning disable SYSLIB1006  
  
// Code that generates compiler diagnostic.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB1006
```

To suppress the warnings in a project file:

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    <TargetFramework>net6.0</TargetFramework>  
    <!-- Nowarn below suppresses SYSLIB1002 project-wide -->  
    <NoWarn>$(NoWarn);SYSLIB1002</NoWarn>  
    <!-- To suppress multiple warnings, you can use multiple Nowarn elements -->  
    <NoWarn>$(NoWarn);SYSLIB1002</NoWarn>  
    <NoWarn>$(NoWarn);SYSLIB1006</NoWarn>  
    <!-- Alternatively, you can suppress multiple warnings by using a semicolon-delimited list -->  
    <NoWarn>$(NoWarn);SYSLIB1002;SYSLIB1006;SYSLIB1007</NoWarn>  
  </PropertyGroup>  
</Project>
```

SYSLIB1005: Could not find a required type definition

9/20/2022 • 2 minutes to read • [Edit Online](#)

The code generator was unable to find a necessary reference to a well-known type.

Workarounds

This error is highly unlikely to occur. If it does occur, you can consider adding a `<PackageReference>` to your project file to include the required type definition.

Suppress warnings

It's recommended that you use one of the workarounds when possible. However, if you cannot change your code, you can suppress the warning through a `#pragma` directive or a `<NoWarn>` project setting. If the `SYSLIB1XXX` source generator diagnostic doesn't surface as an error, you can suppress the warning in code or in your project file.

To suppress the warnings in code:

```
// Disable the warning.  
#pragma warning disable SYSLIB1006  
  
// Code that generates compiler diagnostic.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB1006
```

To suppress the warnings in a project file:

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    <TargetFramework>net6.0</TargetFramework>  
    <!-- Nowarn below suppresses SYSLIB1002 project-wide -->  
    <NoWarn>$(NoWarn);SYSLIB1002</NoWarn>  
    <!-- To suppress multiple warnings, you can use multiple Nowarn elements -->  
    <NoWarn>$(NoWarn);SYSLIB1002</NoWarn>  
    <NoWarn>$(NoWarn);SYSLIB1006</NoWarn>  
    <!-- Alternatively, you can suppress multiple warnings by using a semicolon-delimited list -->  
    <NoWarn>$(NoWarn);SYSLIB1002;SYSLIB1006;SYSLIB1007</NoWarn>  
  </PropertyGroup>  
</Project>
```

SYSLIB1006: Multiple logging methods cannot use the same event ID

9/20/2022 • 2 minutes to read • [Edit Online](#)

Multiple methods annotated with the `LoggerMessageAttribute` are using the same event ID value. Event ID values must be unique within the scope of each assembly.

Workarounds

Review the event ID values used for all logging methods in the assembly and ensure they are unique.

Suppress warnings

It's recommended that you use one of the workarounds when possible. However, if you cannot change your code, you can suppress the warning through a `#pragma` directive or a `<NoWarn>` project setting. If the `SYSLIB1XXX` source generator diagnostic doesn't surface as an error, you can suppress the warning in code or in your project file.

To suppress the warnings in code:

```
// Disable the warning.  
#pragma warning disable SYSLIB1006  
  
// Code that generates compiler diagnostic.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB1006
```

To suppress the warnings in a project file:

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    <TargetFramework>net6.0</TargetFramework>  
    <!-- Nowarn below suppresses SYSLIB1002 project-wide -->  
    <NoWarn>$(NoWarn);SYSLIB1002</NoWarn>  
    <!-- To suppress multiple warnings, you can use multiple Nowarn elements -->  
    <NoWarn>$(NoWarn);SYSLIB1002</NoWarn>  
    <NoWarn>$(NoWarn);SYSLIB1006</NoWarn>  
    <!-- Alternatively, you can suppress multiple warnings by using a semicolon-delimited list -->  
    <NoWarn>$(NoWarn);SYSLIB1002;SYSLIB1006;SYSLIB1007</NoWarn>  
  </PropertyGroup>  
</Project>
```

SYSLIB1007: Logging methods must return void

9/20/2022 • 2 minutes to read • [Edit Online](#)

A method annotated with the `LoggerMessageAttribute` attribute returns a value.

Workarounds

All logging methods must return void.

Suppress warnings

It's recommended that you use one of the workarounds when possible. However, if you cannot change your code, you can suppress the warning through a `#pragma` directive or a `<NoWarn>` project setting. If the `SYSLIB1XXX` source generator diagnostic doesn't surface as an error, you can suppress the warning in code or in your project file.

To suppress the warnings in code:

```
// Disable the warning.  
#pragma warning disable SYSLIB1006  
  
// Code that generates compiler diagnostic.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB1006
```

To suppress the warnings in a project file:

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    <TargetFramework>net6.0</TargetFramework>  
    <!-- Nowarn below suppresses SYSLIB1002 project-wide -->  
    <NoWarn>$(NoWarn);SYSLIB1002</NoWarn>  
    <!-- To suppress multiple warnings, you can use multiple Nowarn elements -->  
    <NoWarn>$(NoWarn);SYSLIB1002</NoWarn>  
    <NoWarn>$(NoWarn);SYSLIB1006</NoWarn>  
    <!-- Alternatively, you can suppress multiple warnings by using a semicolon-delimited list -->  
    <NoWarn>$(NoWarn);SYSLIB1002;SYSLIB1006;SYSLIB1007</NoWarn>  
  </PropertyGroup>  
</Project>
```

SYSLIB1008: One of the arguments to a logging method must implement the `ILogger` interface

9/20/2022 • 2 minutes to read • [Edit Online](#)

One of the parameters of a method annotated with the `LoggerMessageAttribute` must be of the type `ILogger` or a type that implements `ILogger`.

Workarounds

Ensure that a parameter of all logging methods is of type `ILogger` or of a type that implements `ILogger`.

Suppress warnings

It's recommended that you use one of the workarounds when possible. However, if you cannot change your code, you can suppress the warning through a `#pragma` directive or a `<NoWarn>` project setting. If the `SYSLIB1XXX` source generator diagnostic doesn't surface as an error, you can suppress the warning in code or in your project file.

To suppress the warnings in code:

```
// Disable the warning.  
#pragma warning disable SYSLIB1006  
  
// Code that generates compiler diagnostic.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB1006
```

To suppress the warnings in a project file:

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    <TargetFramework>net6.0</TargetFramework>  
    <!-- NoWarn below suppresses SYSLIB1002 project-wide -->  
    <NoWarn>$(NoWarn);SYSLIB1002</NoWarn>  
    <!-- To suppress multiple warnings, you can use multiple NoWarn elements -->  
    <NoWarn>$(NoWarn);SYSLIB1002</NoWarn>  
    <NoWarn>$(NoWarn);SYSLIB1006</NoWarn>  
    <!-- Alternatively, you can suppress multiple warnings by using a semicolon-delimited list -->  
    <NoWarn>$(NoWarn);SYSLIB1002;SYSLIB1006;SYSLIB1007</NoWarn>  
  </PropertyGroup>  
</Project>
```

SYSLIB1009: Logging methods must be static

9/20/2022 • 2 minutes to read • [Edit Online](#)

A method annotated with the `LoggerMessageAttribute` is not static.

Workarounds

All logging methods must be declared as static.

Suppress warnings

It's recommended that you use one of the workarounds when possible. However, if you cannot change your code, you can suppress the warning through a `#pragma` directive or a `<NoWarn>` project setting. If the `SYSLIB1XXX` source generator diagnostic doesn't surface as an error, you can suppress the warning in code or in your project file.

To suppress the warnings in code:

```
// Disable the warning.  
#pragma warning disable SYSLIB1006  
  
// Code that generates compiler diagnostic.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB1006
```

To suppress the warnings in a project file:

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    <TargetFramework>net6.0</TargetFramework>  
    <!-- Nowarn below suppresses SYSLIB1002 project-wide -->  
    <NoWarn>$(NoWarn);SYSLIB1002</NoWarn>  
    <!-- To suppress multiple warnings, you can use multiple Nowarn elements -->  
    <NoWarn>$(NoWarn);SYSLIB1002</NoWarn>  
    <NoWarn>$(NoWarn);SYSLIB1006</NoWarn>  
    <!-- Alternatively, you can suppress multiple warnings by using a semicolon-delimited list -->  
    <NoWarn>$(NoWarn);SYSLIB1002;SYSLIB1006;SYSLIB1007</NoWarn>  
  </PropertyGroup>  
</Project>
```

SYSLIB1010: Logging methods must be partial

9/20/2022 • 2 minutes to read • [Edit Online](#)

A method annotated with the `LoggerMessageAttribute` is not marked as partial.

Workarounds

All logging methods must be declared partial.

Suppress warnings

It's recommended that you use one of the workarounds when possible. However, if you cannot change your code, you can suppress the warning through a `#pragma` directive or a `<NoWarn>` project setting. If the `SYSLIB1XXX` source generator diagnostic doesn't surface as an error, you can suppress the warning in code or in your project file.

To suppress the warnings in code:

```
// Disable the warning.  
#pragma warning disable SYSLIB1006  
  
// Code that generates compiler diagnostic.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB1006
```

To suppress the warnings in a project file:

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    <TargetFramework>net6.0</TargetFramework>  
    <!-- Nowarn below suppresses SYSLIB1002 project-wide -->  
    <NoWarn>$(NoWarn);SYSLIB1002</NoWarn>  
    <!-- To suppress multiple warnings, you can use multiple Nowarn elements -->  
    <NoWarn>$(NoWarn);SYSLIB1002</NoWarn>  
    <NoWarn>$(NoWarn);SYSLIB1006</NoWarn>  
    <!-- Alternatively, you can suppress multiple warnings by using a semicolon-delimited list -->  
    <NoWarn>$(NoWarn);SYSLIB1002;SYSLIB1006;SYSLIB1007</NoWarn>  
  </PropertyGroup>  
</Project>
```

SYSLIB1011: Logging methods cannot be generic

9/20/2022 • 2 minutes to read • [Edit Online](#)

A method annotated with the `LoggerMessageAttribute` contains parameters with generic types.

Workarounds

Logging methods cannot have any generically typed parameters. Use fully resolved types instead.

Suppress warnings

It's recommended that you use one of the workarounds when possible. However, if you cannot change your code, you can suppress the warning through a `#pragma` directive or a `<NoWarn>` project setting. If the `SYSLIB1XXX` source generator diagnostic doesn't surface as an error, you can suppress the warning in code or in your project file.

To suppress the warnings in code:

```
// Disable the warning.  
#pragma warning disable SYSLIB1006  
  
// Code that generates compiler diagnostic.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB1006
```

To suppress the warnings in a project file:

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    <TargetFramework>net6.0</TargetFramework>  
    <!-- Nowarn below suppresses SYSLIB1002 project-wide -->  
    <NoWarn>$(NoWarn);SYSLIB1002</NoWarn>  
    <!-- To suppress multiple warnings, you can use multiple Nowarn elements -->  
    <NoWarn>$(NoWarn);SYSLIB1002</NoWarn>  
    <NoWarn>$(NoWarn);SYSLIB1006</NoWarn>  
    <!-- Alternatively, you can suppress multiple warnings by using a semicolon-delimited list -->  
    <NoWarn>$(NoWarn);SYSLIB1002;SYSLIB1006;SYSLIB1007</NoWarn>  
  </PropertyGroup>  
</Project>
```

SYSLIB1012: Redundant qualifier in logging message

9/20/2022 • 2 minutes to read • [Edit Online](#)

The message string of a `LoggerMessageAttribute` attribute contains a prefix such as `INFO:` or `ERROR:`, which is redundant because each logging message has a corresponding log level.

Workarounds

Remove the prefix from the message string.

Suppress warnings

It's recommended that you use one of the workarounds when possible. However, if you cannot change your code, you can suppress the warning through a `#pragma` directive or a `<NoWarn>` project setting. If the `SYSLIB1XXX` source generator diagnostic doesn't surface as an error, you can suppress the warning in code or in your project file.

To suppress the warnings in code:

```
// Disable the warning.  
#pragma warning disable SYSLIB1006  
  
// Code that generates compiler diagnostic.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB1006
```

To suppress the warnings in a project file:

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    <TargetFramework>net6.0</TargetFramework>  
    <!-- Nowarn below suppresses SYSLIB1002 project-wide -->  
    <NoWarn>$(NoWarn);SYSLIB1002</NoWarn>  
    <!-- To suppress multiple warnings, you can use multiple Nowarn elements -->  
    <NoWarn>$(NoWarn);SYSLIB1002</NoWarn>  
    <NoWarn>$(NoWarn);SYSLIB1006</NoWarn>  
    <!-- Alternatively, you can suppress multiple warnings by using a semicolon-delimited list -->  
    <NoWarn>$(NoWarn);SYSLIB1002;SYSLIB1006;SYSLIB1007</NoWarn>  
  </PropertyGroup>  
</Project>
```

SYSLIB1013: Don't include exception parameters as templates in the logging message

9/20/2022 • 2 minutes to read • [Edit Online](#)

The first exception argument to a logging method is referenced as a template in the logging message. This isn't necessary, because the first exception is passed down to the logging infrastructure explicitly. It doesn't need to be repeated in the logging message.

Workarounds

Remove the template that references the exception argument from the logging message.

Suppress warnings

It's recommended that you use one of the workarounds when possible. However, if you cannot change your code, you can suppress the warning through a `#pragma` directive or a `<NoWarn>` project setting. If the `SYSLIB1XXX` source generator diagnostic doesn't surface as an error, you can suppress the warning in code or in your project file.

To suppress the warnings in code:

```
// Disable the warning.  
#pragma warning disable SYSLIB1006  
  
// Code that generates compiler diagnostic.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB1006
```

To suppress the warnings in a project file:

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    <TargetFramework>net6.0</TargetFramework>  
    <!-- Nowarn below suppresses SYSLIB1002 project-wide -->  
    <NoWarn>$(NoWarn);SYSLIB1002</NoWarn>  
    <!-- To suppress multiple warnings, you can use multiple Nowarn elements -->  
    <NoWarn>$(NoWarn);SYSLIB1002</NoWarn>  
    <NoWarn>$(NoWarn);SYSLIB1006</NoWarn>  
    <!-- Alternatively, you can suppress multiple warnings by using a semicolon-delimited list -->  
    <NoWarn>$(NoWarn);SYSLIB1002;SYSLIB1006;SYSLIB1007</NoWarn>  
  </PropertyGroup>  
</Project>
```

SYSLIB1014: Logging template has no corresponding method argument

9/20/2022 • 2 minutes to read • [Edit Online](#)

A template in the logging message doesn't have a matching parameter in the logging method definition.

Workarounds

Ensure that all templates in logging messages have corresponding parameters in the logging method definition.

Suppress warnings

It's recommended that you use one of the workarounds when possible. However, if you cannot change your code, you can suppress the warning through a `#pragma` directive or a `<NoWarn>` project setting. If the `SYSLIB1XXX` source generator diagnostic doesn't surface as an error, you can suppress the warning in code or in your project file.

To suppress the warnings in code:

```
// Disable the warning.  
#pragma warning disable SYSLIB1006  
  
// Code that generates compiler diagnostic.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB1006
```

To suppress the warnings in a project file:

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    <TargetFramework>net6.0</TargetFramework>  
    <!-- Nowarn below suppresses SYSLIB1002 project-wide -->  
    <NoWarn>$(NoWarn);SYSLIB1002</NoWarn>  
    <!-- To suppress multiple warnings, you can use multiple Nowarn elements -->  
    <NoWarn>$(NoWarn);SYSLIB1002</NoWarn>  
    <NoWarn>$(NoWarn);SYSLIB1006</NoWarn>  
    <!-- Alternatively, you can suppress multiple warnings by using a semicolon-delimited list -->  
    <NoWarn>$(NoWarn);SYSLIB1002;SYSLIB1006;SYSLIB1007</NoWarn>  
  </PropertyGroup>  
</Project>
```

SYSLIB1015: Argument is not referenced from the logging message

9/20/2022 • 2 minutes to read • [Edit Online](#)

An argument in the logging method doesn't have a corresponding template in the logging message.

Workarounds

Ensure that all arguments to the logging method have corresponding templates in the associated logging message.

SUPPRESS WARNINGS

It's recommended that you use one of the workarounds when possible. However, if you cannot change your code, you can suppress the warning through a `#pragma` directive or a `<NoWarn>` project setting. If the `SYSLIB1XXX` source generator diagnostic doesn't surface as an error, you can suppress the warning in code or in your project file.

To suppress the warnings in code:

```
// Disable the warning.  
#pragma warning disable SYSLIB1006  
  
// Code that generates compiler diagnostic.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB1006
```

To suppress the warnings in a project file:

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    <TargetFramework>net6.0</TargetFramework>  
    <!-- Nowarn below suppresses SYSLIB1002 project-wide -->  
    <NoWarn>$(NoWarn);SYSLIB1002</NoWarn>  
    <!-- To suppress multiple warnings, you can use multiple Nowarn elements -->  
    <NoWarn>$(NoWarn);SYSLIB1002</NoWarn>  
    <NoWarn>$(NoWarn);SYSLIB1006</NoWarn>  
    <!-- Alternatively, you can suppress multiple warnings by using a semicolon-delimited list -->  
    <NoWarn>$(NoWarn);SYSLIB1002;SYSLIB1006;SYSLIB1007</NoWarn>  
  </PropertyGroup>  
</Project>
```

SYSLIB1016: Logging methods cannot have a body

9/20/2022 • 2 minutes to read • [Edit Online](#)

The `LoggerMessageAttribute` attribute was applied to a method that has a method body.

Workarounds

Remove the method body.

Suppress warnings

It's recommended that you use one of the workarounds when possible. However, if you cannot change your code, you can suppress the warning through a `#pragma` directive or a `<NoWarn>` project setting. If the `SYSLIB1XXX` source generator diagnostic doesn't surface as an error, you can suppress the warning in code or in your project file.

To suppress the warnings in code:

```
// Disable the warning.  
#pragma warning disable SYSLIB1006  
  
// Code that generates compiler diagnostic.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB1006
```

To suppress the warnings in a project file:

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    <TargetFramework>net6.0</TargetFramework>  
    <!-- Nowarn below suppresses SYSLIB1002 project-wide -->  
    <NoWarn>$(NoWarn);SYSLIB1002</NoWarn>  
    <!-- To suppress multiple warnings, you can use multiple Nowarn elements -->  
    <NoWarn>$(NoWarn);SYSLIB1002</NoWarn>  
    <NoWarn>$(NoWarn);SYSLIB1006</NoWarn>  
    <!-- Alternatively, you can suppress multiple warnings by using a semicolon-delimited list -->  
    <NoWarn>$(NoWarn);SYSLIB1002;SYSLIB1006;SYSLIB1007</NoWarn>  
  </PropertyGroup>  
</Project>
```

SYSLIB1017: A `LogLevel` value must be supplied in the `LoggerMessage` attribute or as a parameter to the logging method

9/20/2022 • 2 minutes to read • [Edit Online](#)

The `LoggerMessageAttribute` attribute was applied to a method without a `LogLevel` value specified. When you do this, one argument to the logging method must be of that type so that the `LogLevel` value ends up being specified explicitly when calling the logging method.

Workarounds

Either specify a `LogLevel` value in the `LoggerMessage` attribute, or make one of the arguments of the logging method a `LogLevel` value.

Suppress warnings

It's recommended that you use one of the workarounds when possible. However, if you cannot change your code, you can suppress the warning through a `#pragma` directive or a `<NoWarn>` project setting. If the `SYSLIB1XXX` source generator diagnostic doesn't surface as an error, you can suppress the warning in code or in your project file.

To suppress the warnings in code:

```
// Disable the warning.  
#pragma warning disable SYSLIB1006  
  
// Code that generates compiler diagnostic.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB1006
```

To suppress the warnings in a project file:

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    <TargetFramework>net6.0</TargetFramework>  
    <!-- NoWarn below suppresses SYSLIB1002 project-wide -->  
    <NoWarn>$(NoWarn);SYSLIB1002</NoWarn>  
    <!-- To suppress multiple warnings, you can use multiple NoWarn elements -->  
    <NoWarn>$(NoWarn);SYSLIB1002</NoWarn>  
    <NoWarn>$(NoWarn);SYSLIB1006</NoWarn>  
    <!-- Alternatively, you can suppress multiple warnings by using a semicolon-delimited list -->  
    <NoWarn>$(NoWarn);SYSLIB1002;SYSLIB1006;SYSLIB1007</NoWarn>  
  </PropertyGroup>  
</Project>
```

SYSLIB1018: Don't include logger parameters as templates in the logging message

9/20/2022 • 2 minutes to read • [Edit Online](#)

The first logger argument to a logging method is referenced as a template in the logging message. The first logger is passed down to the logging infrastructure explicitly, so it doesn't need to be repeated in the logging message.

Workarounds

Remove the template that references the logger argument from the logging message.

Suppress warnings

It's recommended that you use one of the workarounds when possible. However, if you cannot change your code, you can suppress the warning through a `#pragma` directive or a `<NoWarn>` project setting. If the `SYSLIB1XXX` source generator diagnostic doesn't surface as an error, you can suppress the warning in code or in your project file.

To suppress the warnings in code:

```
// Disable the warning.  
#pragma warning disable SYSLIB1006  
  
// Code that generates compiler diagnostic.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB1006
```

To suppress the warnings in a project file:

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    <TargetFramework>net6.0</TargetFramework>  
    <!-- Nowarn below suppresses SYSLIB1002 project-wide -->  
    <NoWarn>$(NoWarn);SYSLIB1002</NoWarn>  
    <!-- To suppress multiple warnings, you can use multiple Nowarn elements -->  
    <NoWarn>$(NoWarn);SYSLIB1002</NoWarn>  
    <NoWarn>$(NoWarn);SYSLIB1006</NoWarn>  
    <!-- Alternatively, you can suppress multiple warnings by using a semicolon-delimited list -->  
    <NoWarn>$(NoWarn);SYSLIB1002;SYSLIB1006;SYSLIB1007</NoWarn>  
  </PropertyGroup>  
</Project>
```

SYSLIB1019: Couldn't find a field of type `ILogger`

9/20/2022 • 2 minutes to read • [Edit Online](#)

When a logging method definition doesn't explicitly include a parameter of type `ILogger`, then the type containing the logging method must have one and only one field of type `ILogger`. The `ILogger` will be used as the target for log messages.

Workarounds

Ensure the type containing the logging method includes a field of type `ILogger` or include a parameter of type `ILogger` in the logging method signature.

Suppress warnings

It's recommended that you use one of the workarounds when possible. However, if you cannot change your code, you can suppress the warning through a `#pragma` directive or a `<NoWarn>` project setting. If the `SYSLIB1XXX` source generator diagnostic doesn't surface as an error, you can suppress the warning in code or in your project file.

To suppress the warnings in code:

```
// Disable the warning.  
#pragma warning disable SYSLIB1006  
  
// Code that generates compiler diagnostic.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB1006
```

To suppress the warnings in a project file:

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    <TargetFramework>net6.0</TargetFramework>  
    <!-- NoWarn below suppresses SYSLIB1002 project-wide -->  
    <NoWarn>$(NoWarn);SYSLIB1002</NoWarn>  
    <!-- To suppress multiple warnings, you can use multiple NoWarn elements -->  
    <NoWarn>$(NoWarn);SYSLIB1002</NoWarn>  
    <NoWarn>$(NoWarn);SYSLIB1006</NoWarn>  
    <!-- Alternatively, you can suppress multiple warnings by using a semicolon-delimited list -->  
    <NoWarn>$(NoWarn);SYSLIB1002;SYSLIB1006;SYSLIB1007</NoWarn>  
  </PropertyGroup>  
</Project>
```

SYSLIB1020: Found multiple fields of type `ILogger`

9/20/2022 • 2 minutes to read • [Edit Online](#)

When a logging method definition doesn't explicitly include a parameter of type `ILogger`, then the type containing the logging method must have one and only one field of type `ILogger`, which will be used as the target for log messages.

Workarounds

Ensure the type containing the logging method includes only a single field of type `ILogger`.

Suppress warnings

It's recommended that you use one of the workarounds when possible. However, if you cannot change your code, you can suppress the warning through a `#pragma` directive or a `<NoWarn>` project setting. If the `SYSLIB1XXX` source generator diagnostic doesn't surface as an error, you can suppress the warning in code or in your project file.

To suppress the warnings in code:

```
// Disable the warning.  
#pragma warning disable SYSLIB1006  
  
// Code that generates compiler diagnostic.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB1006
```

To suppress the warnings in a project file:

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    <TargetFramework>net6.0</TargetFramework>  
    <!-- NoWarn below suppresses SYSLIB1002 project-wide -->  
    <NoWarn>$(NoWarn);SYSLIB1002</NoWarn>  
    <!-- To suppress multiple warnings, you can use multiple Nowarn elements -->  
    <NoWarn>$(NoWarn);SYSLIB1002</NoWarn>  
    <NoWarn>$(NoWarn);SYSLIB1006</NoWarn>  
    <!-- Alternatively, you can suppress multiple warnings by using a semicolon-delimited list -->  
    <NoWarn>$(NoWarn);SYSLIB1002;SYSLIB1006;SYSLIB1007</NoWarn>  
  </PropertyGroup>  
</Project>
```

SYSLIB1021: Multiple message-template item names differ only by case

9/20/2022 • 2 minutes to read • [Edit Online](#)

A method annotated with the `LoggerMessageAttribute` attribute has multiple template item names that differ only by case.

Workarounds

Make sure the message-template item names aren't repeated with different casing in the method annotated with `LoggerMessageAttribute`.

Suppress warnings

It's recommended that you use one of the workarounds when possible. However, if you cannot change your code, you can suppress the warning through a `#pragma` directive or a `<NoWarn>` project setting. If the `SYSLIB1XXX` source generator diagnostic doesn't surface as an error, you can suppress the warning in code or in your project file.

To suppress the warnings in code:

```
// Disable the warning.  
#pragma warning disable SYSLIB1006  
  
// Code that generates compiler diagnostic.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB1006
```

To suppress the warnings in a project file:

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    <TargetFramework>net6.0</TargetFramework>  
    <!-- Nowarn below suppresses SYSLIB1002 project-wide -->  
    <NoWarn>$(NoWarn);SYSLIB1002</NoWarn>  
    <!-- To suppress multiple warnings, you can use multiple Nowarn elements -->  
    <NoWarn>$(NoWarn);SYSLIB1002</NoWarn>  
    <NoWarn>$(NoWarn);SYSLIB1006</NoWarn>  
    <!-- Alternatively, you can suppress multiple warnings by using a semicolon-delimited list -->  
    <NoWarn>$(NoWarn);SYSLIB1002;SYSLIB1006;SYSLIB1007</NoWarn>  
  </PropertyGroup>  
</Project>
```

SYSLIB1022: Can't have malformed format strings

9/20/2022 • 2 minutes to read • [Edit Online](#)

A method annotated with the `LoggerMessageAttribute` attribute has a message template that's formatted incorrectly. For example, the template has an unmatched curly brace (`}`).

Workarounds

Makes sure curly braces are used appropriately in the message template.

Suppress warnings

It's recommended that you use one of the workarounds when possible. However, if you cannot change your code, you can suppress the warning through a `#pragma` directive or a `<NoWarn>` project setting. If the `SYSLIB1XXX` source generator diagnostic doesn't surface as an error, you can suppress the warning in code or in your project file.

To suppress the warnings in code:

```
// Disable the warning.  
#pragma warning disable SYSLIB1006  
  
// Code that generates compiler diagnostic.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB1006
```

To suppress the warnings in a project file:

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    <TargetFramework>net6.0</TargetFramework>  
    <!-- Nowarn below suppresses SYSLIB1002 project-wide -->  
    <NoWarn>$(NoWarn);SYSLIB1002</NoWarn>  
    <!-- To suppress multiple warnings, you can use multiple Nowarn elements -->  
    <NoWarn>$(NoWarn);SYSLIB1002</NoWarn>  
    <NoWarn>$(NoWarn);SYSLIB1006</NoWarn>  
    <!-- Alternatively, you can suppress multiple warnings by using a semicolon-delimited list -->  
    <NoWarn>$(NoWarn);SYSLIB1002;SYSLIB1006;SYSLIB1007</NoWarn>  
  </PropertyGroup>  
</Project>
```

SYSLIB1023: Generating more than six arguments is not supported

9/20/2022 • 2 minutes to read • [Edit Online](#)

The `LoggerMessageAttribute` is limited to the number of parameters supported by `LoggerMessage.Define`, which is six.

Workarounds

Rather than using `LoggerMessageAttribute`, consider implementing a custom attribute using a struct.

SUPPRESS WARNINGS

It's recommended that you use one of the workarounds when possible. However, if you cannot change your code, you can suppress the warning through a `#pragma` directive or a `<NoWarn>` project setting. If the `SYSLIB1XXX` source generator diagnostic doesn't surface as an error, you can suppress the warning in code or in your project file.

To suppress the warnings in code:

```
// Disable the warning.  
#pragma warning disable SYSLIB1006  
  
// Code that generates compiler diagnostic.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB1006
```

To suppress the warnings in a project file:

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    <TargetFramework>net6.0</TargetFramework>  
    <!-- NoWarn below suppresses SYSLIB1002 project-wide -->  
    <NoWarn>$(NoWarn);SYSLIB1002</NoWarn>  
    <!-- To suppress multiple warnings, you can use multiple NoWarn elements -->  
    <NoWarn>$(NoWarn);SYSLIB1002</NoWarn>  
    <NoWarn>$(NoWarn);SYSLIB1006</NoWarn>  
    <!-- Alternatively, you can suppress multiple warnings by using a semicolon-delimited list -->  
    <NoWarn>$(NoWarn);SYSLIB1002;SYSLIB1006;SYSLIB1007</NoWarn>  
  </PropertyGroup>  
</Project>
```

SYSLIB1030: `System.Text.Json` source generator did not generate output for type

9/20/2022 • 2 minutes to read • [Edit Online](#)

The `System.Text.Json` source generator did not generate output for a given type in the input object graph. This typically means that the type is not supported by `JsonSerializer`, for example multi-dimensional arrays like `int[,]`.

Workarounds

Register a [custom converter](#) for the type.

Suppress warnings

It's recommended that you use one of the workarounds when possible. However, if you cannot change your code, you can suppress the warning through a `#pragma` directive or a `<NoWarn>` project setting. If the `SYSLIB1XXX` source generator diagnostic doesn't surface as an error, you can suppress the warning in code or in your project file.

To suppress the warnings in code:

```
// Disable the warning.  
#pragma warning disable SYSLIB1006  
  
// Code that generates compiler diagnostic.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB1006
```

To suppress the warnings in a project file:

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    <TargetFramework>net6.0</TargetFramework>  
    <!-- NoWarn below suppresses SYSLIB1002 project-wide -->  
    <NoWarn>$(NoWarn);SYSLIB1002</NoWarn>  
    <!-- To suppress multiple warnings, you can use multiple NoWarn elements -->  
    <NoWarn>$(NoWarn);SYSLIB1002</NoWarn>  
    <NoWarn>$(NoWarn);SYSLIB1006</NoWarn>  
    <!-- Alternatively, you can suppress multiple warnings by using a semicolon-delimited list -->  
    <NoWarn>$(NoWarn);SYSLIB1002;SYSLIB1006;SYSLIB1007</NoWarn>  
  </PropertyGroup>  
</Project>
```

SYSLIB1031: `System.Text.Json` source generator encountered a duplicate type info property name

9/20/2022 • 2 minutes to read • [Edit Online](#)

The `System.Text.Json` source generator encountered a duplicate type info property name to be generated on the specified partial context type.

Workarounds

Specify a new type info property name for the duplicate instance using

`System.Text.Json.Serialization.JsonSerializableAttribute.TypeInfoPropertyName`.

Suppress warnings

It's recommended that you use one of the workarounds when possible. However, if you cannot change your code, you can suppress the warning through a `#pragma` directive or a `<NoWarn>` project setting. If the `SYSLIB1XXX` source generator diagnostic doesn't surface as an error, you can suppress the warning in code or in your project file.

To suppress the warnings in code:

```
// Disable the warning.  
#pragma warning disable SYSLIB1006  
  
// Code that generates compiler diagnostic.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB1006
```

To suppress the warnings in a project file:

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    <TargetFramework>net6.0</TargetFramework>  
    <!-- NoWarn below suppresses SYSLIB1002 project-wide -->  
    <NoWarn>$(NoWarn);SYSLIB1002</NoWarn>  
    <!-- To suppress multiple warnings, you can use multiple NoWarn elements -->  
    <NoWarn>$(NoWarn);SYSLIB1002</NoWarn>  
    <NoWarn>$(NoWarn);SYSLIB1006</NoWarn>  
    <!-- Alternatively, you can suppress multiple warnings by using a semicolon-delimited list -->  
    <NoWarn>$(NoWarn);SYSLIB1002;SYSLIB1006;SYSLIB1007</NoWarn>  
  </PropertyGroup>  
</Project>
```

SYSLIB1032: Context classes to be augmented by the `System.Text.Json` source generator must be declared as partial

9/20/2022 • 2 minutes to read • [Edit Online](#)

The `System.Text.Json` source generator encountered a context type included for source generation that is not partial, or whose containing type(s) is not partial.

Workarounds

Make the context type and all containing types partial.

SUPPRESS WARNINGS

It's recommended that you use one of the workarounds when possible. However, if you cannot change your code, you can suppress the warning through a `#pragma` directive or a `<NoWarn>` project setting. If the `SYSLIB1XXX` source generator diagnostic doesn't surface as an error, you can suppress the warning in code or in your project file.

To suppress the warnings in code:

```
// Disable the warning.  
#pragma warning disable SYSLIB1006  
  
// Code that generates compiler diagnostic.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB1006
```

To suppress the warnings in a project file:

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    <TargetFramework>net6.0</TargetFramework>  
    <!-- NoWarn below suppresses SYSLIB1002 project-wide -->  
    <NoWarn>$(NoWarn);SYSLIB1002</NoWarn>  
    <!-- To suppress multiple warnings, you can use multiple NoWarn elements -->  
    <NoWarn>$(NoWarn);SYSLIB1002</NoWarn>  
    <NoWarn>$(NoWarn);SYSLIB1006</NoWarn>  
    <!-- Alternatively, you can suppress multiple warnings by using a semicolon-delimited list -->  
    <NoWarn>$(NoWarn);SYSLIB1002;SYSLIB1006;SYSLIB1007</NoWarn>  
  </PropertyGroup>  
</Project>
```

SYSLIB1033: `System.Text.Json` source generator encountered a type with multiple `[JsonConstructor]` annotations

9/20/2022 • 2 minutes to read • [Edit Online](#)

The `System.Text.Json` source generator encountered a serializable type with multiple `[JsonConstructor]` annotations.

Workarounds

Remove duplicate `[JsonConstructor]` annotations.

Suppress warnings

It's recommended that you use one of the workarounds when possible. However, if you cannot change your code, you can suppress the warning through a `#pragma` directive or a `<NoWarn>` project setting. If the `SYSLIB1XXX` source generator diagnostic doesn't surface as an error, you can suppress the warning in code or in your project file.

To suppress the warnings in code:

```
// Disable the warning.  
#pragma warning disable SYSLIB1006  
  
// Code that generates compiler diagnostic.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB1006
```

To suppress the warnings in a project file:

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    <TargetFramework>net6.0</TargetFramework>  
    <!-- NoWarn below suppresses SYSLIB1002 project-wide -->  
    <NoWarn>$(NoWarn);SYSLIB1002</NoWarn>  
    <!-- To suppress multiple warnings, you can use multiple NoWarn elements -->  
    <NoWarn>$(NoWarn);SYSLIB1002</NoWarn>  
    <NoWarn>$(NoWarn);SYSLIB1006</NoWarn>  
    <!-- Alternatively, you can suppress multiple warnings by using a semicolon-delimited list -->  
    <NoWarn>$(NoWarn);SYSLIB1002;SYSLIB1006;SYSLIB1007</NoWarn>  
  </PropertyGroup>  
</Project>
```

SYSLIB1035: `System.Text.Json` source generator encountered a type with multiple `[JsonExtensionData]` annotations

9/20/2022 • 2 minutes to read • [Edit Online](#)

The `System.Text.Json` source generator encountered a serializable type with multiple `[JsonExtensionData]` annotations.

Workarounds

Remove duplicate `[JsonExtensionData]` annotations.

Suppress warnings

It's recommended that you use one of the workarounds when possible. However, if you cannot change your code, you can suppress the warning through a `#pragma` directive or a `<NoWarn>` project setting. If the `SYSLIB1XXX` source generator diagnostic doesn't surface as an error, you can suppress the warning in code or in your project file.

To suppress the warnings in code:

```
// Disable the warning.  
#pragma warning disable SYSLIB1006  
  
// Code that generates compiler diagnostic.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB1006
```

To suppress the warnings in a project file:

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    <TargetFramework>net6.0</TargetFramework>  
    <!-- NoWarn below suppresses SYSLIB1002 project-wide -->  
    <NoWarn>$(NoWarn);SYSLIB1002</NoWarn>  
    <!-- To suppress multiple warnings, you can use multiple NoWarn elements -->  
    <NoWarn>$(NoWarn);SYSLIB1002</NoWarn>  
    <NoWarn>$(NoWarn);SYSLIB1006</NoWarn>  
    <!-- Alternatively, you can suppress multiple warnings by using a semicolon-delimited list -->  
    <NoWarn>$(NoWarn);SYSLIB1002;SYSLIB1006;SYSLIB1007</NoWarn>  
  </PropertyGroup>  
</Project>
```

SYSLIB1036: `System.Text.Json` source generator encountered an invalid `[JsonExtensionData]` annotation

9/20/2022 • 2 minutes to read • [Edit Online](#)

The `System.Text.Json` source generator encountered a property or field annotated with `[JsonExtensionData]` but whose data type does not implement `IDictionary<string, JsonElement>`, `IDictionary<string, object>`, `IDictionary<string, JsonNode>`, or `JsonNode`.

Workarounds

Ensure that the data type for any property or field that is annotated with `[JsonExtensionData]` implements `IDictionary<string, JsonElement>`, `IDictionary<string, object>`, `IDictionary<string, JsonNode>`, or `JsonNode`.

Suppress warnings

It's recommended that you use one of the workarounds when possible. However, if you cannot change your code, you can suppress the warning through a `#pragma` directive or a `<NoWarn>` project setting. If the `SYSLIB1XXX` source generator diagnostic doesn't surface as an error, you can suppress the warning in code or in your project file.

To suppress the warnings in code:

```
// Disable the warning.  
#pragma warning disable SYSLIB1006  
  
// Code that generates compiler diagnostic.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB1006
```

To suppress the warnings in a project file:

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    <TargetFramework>net6.0</TargetFramework>  
    <!-- NoWarn below suppresses SYSLIB1002 project-wide -->  
    <NoWarn>$(NoWarn);SYSLIB1002</NoWarn>  
    <!-- To suppress multiple warnings, you can use multiple NoWarn elements -->  
    <NoWarn>$(NoWarn);SYSLIB1002</NoWarn>  
    <NoWarn>$(NoWarn);SYSLIB1006</NoWarn>  
    <!-- Alternatively, you can suppress multiple warnings by using a semicolon-delimited list -->  
    <NoWarn>$(NoWarn);SYSLIB1002;SYSLIB1006;SYSLIB1007</NoWarn>  
  </PropertyGroup>  
</Project>
```

SYSLIB1037: `System.Text.Json` source generator encountered a type with init-only properties which are not supported for deserialization

9/20/2022 • 2 minutes to read • [Edit Online](#)

The `System.Text.Json` source generator encountered a type with init-only properties, such as a record type. These properties are currently not supported by the source generator for deserialization.

Workarounds

If deserialization of init-only properties is required, use the reflection-based `JsonSerializer` implementation.

SUPPRESS WARNINGS

It's recommended that you use one of the workarounds when possible. However, if you cannot change your code, you can suppress the warning through a `#pragma` directive or a `<NoWarn>` project setting. If the `SYSLIB1006` source generator diagnostic doesn't surface as an error, you can suppress the warning in code or in your project file.

To suppress the warnings in code:

```
// Disable the warning.  
#pragma warning disable SYSLIB1006  
  
// Code that generates compiler diagnostic.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB1006
```

To suppress the warnings in a project file:

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    <TargetFramework>net6.0</TargetFramework>  
    <!-- Nowarn below suppresses SYSLIB1002 project-wide -->  
    <NoWarn>$(NoWarn);SYSLIB1002</NoWarn>  
    <!-- To suppress multiple warnings, you can use multiple Nowarn elements -->  
    <NoWarn>$(NoWarn);SYSLIB1002</NoWarn>  
    <NoWarn>$(NoWarn);SYSLIB1006</NoWarn>  
    <!-- Alternatively, you can suppress multiple warnings by using a semicolon-delimited list -->  
    <NoWarn>$(NoWarn);SYSLIB1002;SYSLIB1006;SYSLIB1007</NoWarn>  
  </PropertyGroup>  
</Project>
```

SYSLIB1038: `System.Text.Json` source generator encountered a property annotated with `[JsonInclude]` but with inaccessible accessors

9/20/2022 • 2 minutes to read • [Edit Online](#)

The `System.Text.Json` source generator encountered a property that is annotated with `[JsonInclude]` but has accessors that are inaccessible to the source generator. The property must have `public` or `internal` accessors.

Workarounds

If serialization or deserialization of properties with accessors that are not `public` or `internal` is required, use the [reflection-based `JsonSerializer` implementation](#).

Suppress warnings

It's recommended that you use one of the workarounds when possible. However, if you cannot change your code, you can suppress the warning through a `#pragma` directive or a `<NoWarn>` project setting. If the `SYSLIB1XXX` source generator diagnostic doesn't surface as an error, you can suppress the warning in code or in your project file.

To suppress the warnings in code:

```
// Disable the warning.  
#pragma warning disable SYSLIB1006  
  
// Code that generates compiler diagnostic.  
// ...  
  
// Re-enable the warning.  
#pragma warning restore SYSLIB1006
```

To suppress the warnings in a project file:

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    <TargetFramework>net6.0</TargetFramework>  
    <!-- NoWarn below suppresses SYSLIB1002 project-wide -->  
    <NoWarn>$(NoWarn);SYSLIB1002</NoWarn>  
    <!-- To suppress multiple warnings, you can use multiple NoWarn elements -->  
    <NoWarn>$(NoWarn);SYSLIB1002</NoWarn>  
    <NoWarn>$(NoWarn);SYSLIB1006</NoWarn>  
    <!-- Alternatively, you can suppress multiple warnings by using a semicolon-delimited list -->  
    <NoWarn>$(NoWarn);SYSLIB1002;SYSLIB1006;SYSLIB1007</NoWarn>  
  </PropertyGroup>  
</Project>
```

.NET project SDKs

9/20/2022 • 9 minutes to read • [Edit Online](#)

.NET Core and .NET 5 and later projects are associated with a software development kit (SDK). Each *project SDK* is a set of MSBuild [targets](#) and associated [tasks](#) that are responsible for compiling, packing, and publishing code. A project that references a project SDK is sometimes referred to as an *SDK-style project*.

Available SDKs

The following SDKs are available:

ID	DESCRIPTION	REPO
<code>Microsoft.NET.Sdk</code>	The .NET SDK	https://github.com/dotnet/sdk
<code>Microsoft.NET.Sdk.Web</code>	The .NET Web SDK	https://github.com/dotnet/sdk
<code>Microsoft.NET.Sdk.BlazorWebAssembly</code>	The .NET Blazor WebAssembly SDK	
<code>Microsoft.NET.Sdk.Razor</code>	The .NET Razor SDK	
<code>Microsoft.NET.Sdk.Worker</code>	The .NET Worker Service SDK	
<code>Microsoft.NET.Sdk.WindowsDesktop</code>	The .NET Desktop SDK, which includes Windows Forms (WinForms) and Windows Presentation Foundation (WPF).*	https://github.com/dotnet/winforms and https://github.com/dotnet/wpf

The .NET SDK is the base SDK for .NET. The other SDKs reference the .NET SDK, and projects that are associated with the other SDKs have all the .NET SDK properties available to them. The Web SDK, for example, depends on both the .NET SDK and the Razor SDK.

You can also author your own SDK that can be distributed via NuGet.

* Starting in .NET 5, Windows Forms and Windows Presentation Foundation (WPF) projects should specify the .NET SDK (`Microsoft.NET.Sdk`) instead of `Microsoft.NET.Sdk.WindowsDesktop`. For these projects, setting `TargetFramework` to `net5.0-windows` and `UseWPF` or `UseWindowsForms` to `true` will automatically import the Windows desktop SDK. If your project targets .NET 5 or later and specifies the `Microsoft.NET.Sdk.WindowsDesktop` SDK, you'll get build warning NETSDK1137.

Project files

.NET projects are based on the [MSBuild](#) format. Project files, which have extensions like `.csproj` for C# projects and `.fsproj` for F# projects, are in XML format. The root element of an MSBuild project file is the [Project](#) element. The `Project` element has an optional `Sdk` attribute that specifies which SDK (and version) to use. To use the .NET tools and build your code, set the `Sdk` attribute to one of the IDs in the [Available SDKs](#) table.

```
<Project Sdk="Microsoft.NET.Sdk">
  ...
</Project>
```

To specify an SDK that comes from NuGet, include the version at the end of the name, or specify the name and version in the `global.json` file.

```
<Project Sdk="MSBuild.Sdk.Extras/2.0.54">
  ...
</Project>
```

Another way to specify the SDK is with the top-level `Sdk` element:

```
<Project>
  <Sdk Name="Microsoft.NET.Sdk" />
  ...
</Project>
```

Referencing an SDK in one of these ways greatly simplifies project files for .NET. While evaluating the project, MSBuild adds implicit imports for `Sdk.props` at the top of the project file and `Sdk.targets` at the bottom.

```
<Project>
  <!-- Implicit top import -->
  <Import Project="Sdk.props" Sdk="Microsoft.NET.Sdk" />
  ...
  <!-- Implicit bottom import -->
  <Import Project="Sdk.targets" Sdk="Microsoft.NET.Sdk" />
</Project>
```

TIP

On a Windows machine, the `Sdk.props` and `Sdk.targets` files can be found in the `%ProgramFiles%\dotnet\sdk\{version}\Sdks\Microsoft.NET.Sdk\Sdk` folder.

Preprocess the project file

You can see the fully expanded project as MSBuild sees it after the SDK and its targets are included by using the `dotnet msbuild -preprocess` command. The `preprocess` switch of the `dotnet msbuild` command shows which files are imported, their sources, and their contributions to the build without actually building the project.

If the project has multiple target frameworks, focus the results of the command on only one framework by specifying it as an MSBuild property. For example:

```
dotnet msbuild -property:TargetFramework=netcoreapp2.0 -preprocess:output.xml
```

Default includes and excludes

The default includes and excludes for `Compile` items, `embedded resources`, and `None` items are defined in the SDK. Unlike non-SDK .NET Framework projects, you don't need to specify these items in your project file, because the defaults cover most common use cases. This behavior makes the project file smaller and easier to understand and edit by hand, if needed.

The following table shows which elements and which `globs` are included and excluded in the .NET SDK:

ELEMENT	INCLUDE GLOB	EXCLUDE GLOB	REMOVE GLOB
Compile	<code>**/*.cs</code> (or other language extensions)	<code>**/*.user; **/*.*proj; **/*.sln; **/*.vsscc</code>	N/A

ELEMENT	INCLUDE GLOB	EXCLUDE GLOB	REMOVE GLOB
EmbeddedResource	**/*.resx	**/*.user; **/*.proj; **/*.sln; **/*.vsscc	N/A
None	**/*	**/*.user; **/*.proj; **/*.sln; **/*.vsscc	**/*.cs; **/*.resx

NOTE

The `./bin` and `./obj` folders, which are represented by the `$(BaseOutputPath)` and `$(BaseIntermediateOutputPath)` MSBuild properties, are excluded from the globs by default. Excludes are represented by the [DefaultItemExcludes](#) property.

The .NET Desktop SDK has more includes and excludes for WPF. For more information, see [WPF default includes and excludes](#).

Build errors

If you explicitly define any of these items in your project file, you're likely to get a "NETSDK1022" build error similar to the following:

Duplicate 'Compile' items were included. The .NET SDK includes 'Compile' items from your project directory by default. You can either remove these items from your project file, or set the 'EnableDefaultCompileItems' property to 'false' if you want to explicitly include them in your project file.

Duplicate 'EmbeddedResource' items were included. The .NET SDK includes 'EmbeddedResource' items from your project directory by default. You can either remove these items from your project file, or set the 'EnableDefaultEmbeddedResourceItems' property to 'false' if you want to explicitly include them in your project file.

To resolve the errors, do one of the following:

- Remove the explicit `Compile`, `EmbeddedResource`, or `None` items that match the implicit ones listed on the previous table.
- Set the [EnableDefaultItems](#) property to `false` to disable all implicit file inclusion:

```
<PropertyGroup>
  <EnableDefaultItems>false</EnableDefaultItems>
</PropertyGroup>
```

If you want to specify files to be published with your app, you can still use the known MSBuild mechanisms for that, for example, the `Content` element.

- Selectively disable only `Compile`, `EmbeddedResource`, or `None` globs by setting the `EnableDefaultCompileItems`, `EnableDefaultEmbeddedResourceItems`, or `EnableDefaultNoneItems` property to `false`:

```
<PropertyGroup>
  <EnableDefaultCompileItems>false</EnableDefaultCompileItems>
  <EnableDefaultEmbeddedResourceItems>false</EnableDefaultEmbeddedResourceItems>
  <EnableDefaultNoneItems>false</EnableDefaultNoneItems>
</PropertyGroup>
```

If you only disable `Compile` globs, Solution Explorer in Visual Studio still shows *.cs items as part of the project, included as `None` items. To disable the implicit `None` glob, set `EnableDefaultNoneItems` to `false` too.

Implicit using directives

Starting in .NET 6, implicit `global using` directives are added to new C# projects. This means that you can use types defined in these namespaces without having to specify their fully qualified name or manually add a `using` directive. The *implicit* aspect refers to the fact that the `global using` directives are added to a generated file in the project's *obj* directory.

Implicit `global using` directives are added for projects that use one of the following SDKs:

- Microsoft.NET.Sdk
- Microsoft.NET.Sdk.Web
- Microsoft.NET.Sdk.Worker
- Microsoft.NET.Sdk.WindowsDesktop

A `global using` directive is added for each namespaces in a set of default namespaces that's based on the project's SDK. These default namespaces are shown in the following table.

SDK	DEFAULT NAMESPACES
Microsoft.NET.Sdk	<code>System</code> <code>System.Collections.Generic</code> <code>System.IO</code> <code>System.Linq</code> <code>System.Net.Http</code> <code>System.Threading</code> <code>System.Threading.Tasks</code>
Microsoft.NET.Sdk.Web	<code>System.Net.Http.Json</code> <code>Microsoft.AspNetCore.Builder</code> <code>Microsoft.AspNetCore.Hosting</code> <code>Microsoft.AspNetCore.Http</code> <code>Microsoft.AspNetCore.Routing</code> <code>Microsoft.Extensions.Configuration</code> <code>Microsoft.Extensions.DependencyInjection</code> <code>Microsoft.Extensions.Hosting</code> <code>Microsoft.Extensions.Logging</code>
Microsoft.NET.Sdk.Worker	<code>Microsoft.Extensions.Configuration</code> <code>Microsoft.Extensions.DependencyInjection</code> <code>Microsoft.Extensions.Hosting</code> <code>Microsoft.Extensions.Logging</code>
Microsoft.NET.Sdk.WindowsDesktop (Windows Forms)	Microsoft.NET.Sdk namespaces <code>System.Drawing</code> <code>System.Windows.Forms</code>
Microsoft.NET.Sdk.WindowsDesktop (WPF)	Microsoft.NET.Sdk namespaces Removed <code>System.IO</code> Removed <code>System.Net.Http</code>

If you want to disable this feature, or if you want to enable implicit `global using` directives in an existing C# project, you can do so via the [ImplicitUsings](#) MSBuild property.

You can specify additional implicit `global using` directives by adding `Using` items (or `Import` items for Visual

Basic projects) to your project file, for example:

```
<ItemGroup>
  <Using Include="System.IO.Pipes" />
</ItemGroup>
```

Implicit package references

When targeting .NET Core 1.0 - 2.2 or .NET Standard 1.0 - 2.0, the .NET SDK adds implicit references to certain *metapackages*. A metapackage is a framework-based package that consists only of dependencies on other packages. Metapackages are implicitly referenced based on the target framework(s) specified in the [TargetFramework](#) or [TargetFrameworks](#) property of your project file.

```
<PropertyGroup>
  <TargetFramework>netcoreapp2.1</TargetFramework>
</PropertyGroup>
```

```
<PropertyGroup>
  <TargetFrameworks>netcoreapp2.1;net462</TargetFrameworks>
</PropertyGroup>
```

If needed, you can disable implicit package references using the [DisableImplicitFrameworkReferences](#) property, and add explicit references to just the frameworks or packages you need.

Recommendations:

- When targeting .NET Framework, .NET Core 1.0 - 2.2, or .NET Standard 1.0 - 2.0, don't add an explicit reference to the `Microsoft.NETCore.App` or `NETStandard.Library` metapackages via a `<PackageReference>` item in your project file. For .NET Core 1.0 - 2.2 and .NET Standard 1.0 - 2.0 projects, these metapackages are implicitly referenced. For .NET Framework projects, if any version of `NETStandard.Library` is needed when using a .NET Standard-based NuGet package, NuGet automatically installs that version.
- If you need a specific version of the runtime when targeting .NET Core 1.0 - 2.2, use the `<RuntimeFrameworkVersion>` property in your project (for example, `1.0.4`) instead of referencing the metapackage. For example, you might need a specific patch version of 1.0.0 LTS runtime if you're using [self-contained deployments](#).
- If you need a specific version of the `NETStandard.Library` metapackage when targeting .NET Standard 1.0 - 2.0, you can use the `<NetStandardImplicitPackageVersion>` property and set the version you need.

Build events

In SDK-style projects, use an MSBuild target named `PreBuild` or `PostBuild` and set the `BeforeTargets` property for `PreBuild` or the `AfterTargets` property for `PostBuild`.

```
<Target Name="PreBuild" BeforeTargets="PreBuildEvent">
  <Exec Command="$(ProjectDir)PreBuildEvent.bat" $(ProjectDir)..\"/>
</Target>

<Target Name="PostBuild" AfterTargets="PostBuildEvent">
  <Exec Command="echo Output written to $(TargetDir)" />
</Target>
```

NOTE

- You can use any name for the MSBuild targets. However, the Visual Studio IDE recognizes `PreBuild` and `PostBuild` targets, so by using those names, you can edit the commands in the IDE.
- The properties `PreBuildEvent` and `PostBuildEvent` are not recommended in SDK-style projects, because macros such as `$(ProjectDir)` aren't resolved. For example, the following code is not supported:

```
<PropertyGroup>
  <PreBuildEvent>"$(ProjectDir)PreBuildEvent.bat" "$(ProjectDir)..\" "$(ProjectDir)" "$(TargetDir)"
</PreBuildEvent>
</PropertyGroup>
```

Customize the build

There are various ways to [customize a build](#). You may want to override a property by passing it as an argument to an `msbuild` or `dotnet` command. You can also add the property to the project file or to a `Directory.Build.props` file. For a list of useful properties for .NET projects, see [MSBuild reference for .NET SDK projects](#).

Custom targets

.NET projects can package custom MSBuild targets and properties for use by projects that consume the package. Use this type of extensibility when you want to:

- Extend the build process.
- Access artifacts of the build process, such as generated files.
- Inspect the configuration under which the build is invoked.

You add custom build targets or properties by placing files in the form `<package_id>.targets` or `<package_id>.props` (for example, `Contoso.Utility.UsefulStuff.targets`) in the `build` folder of the project.

The following XML is a snippet from a `.csproj` file that instructs the `dotnet pack` command what to package. The `<ItemGroup Label="dotnet pack instructions">` element places the targets files into the `build` folder inside the package. The `<Target Name="CollectRuntimeOutputs" BeforeTargets="_GetPackageFiles">` element places the assemblies and `.json` files into the `build` folder.

```
<Project Sdk="Microsoft.NET.Sdk">

  ...
  <ItemGroup Label="dotnet pack instructions">
    <Content Include="build\*.targets">
      <Pack>true</Pack>
      <PackagePath>build\</PackagePath>
    </Content>
  </ItemGroup>
  <Target Name="CollectRuntimeOutputs" BeforeTargets="_GetPackageFiles">
    <!-- Collect these items inside a target that runs after build but before packaging. -->
    <ItemGroup>
      <Content Include="$(OutputPath)\*.dll;$(OutputPath)\*.json">
        <Pack>true</Pack>
        <PackagePath>build\</PackagePath>
      </Content>
    </ItemGroup>
  </Target>
  ...
</Project>
```

To consume a custom target in your project, add a `PackageReference` element that points to the package and its

version. Unlike the tools, the custom targets package is included in the consuming project's dependency closure.

You can configure how to use the custom target. Since it's an MSBuild target, it can depend on a given target, run after another target, or be manually invoked by using the `dotnet msbuild -t:<target-name>` command. However, to provide a better user experience, you can combine per-project tools and custom targets. In this scenario, the per-project tool accepts whatever parameters are needed and translates that into the required `dotnet msbuild` invocation that executes the target. You can see a sample of this kind of synergy on the [MVP Summit 2016 Hackathon samples](#) repo in the `dotnet-packer` project.

See also

- [Install .NET Core](#)
- [How to use MSBuild project SDKs](#)
- [Package custom MSBuild targets and props with NuGet](#)

MSBuild reference for .NET SDK projects

9/20/2022 • 35 minutes to read • [Edit Online](#)

This page is a reference for the MSBuild properties and items that you can use to configure .NET projects.

NOTE

This page is a work in progress and does not list all of the useful MSBuild properties for the .NET SDK. For a list of common MSBuild properties, see [Common MSBuild properties](#).

Framework properties

The following MSBuild properties are documented in this section:

- [TargetFramework](#)
- [TargetFrameworks](#)
- [NetStandardImplicitPackageVersion](#)

TargetFramework

The `TargetFramework` property specifies the target framework version for the app. For a list of valid target framework monikers, see [Target frameworks in SDK-style projects](#).

```
<PropertyGroup>
  <TargetFramework>netcoreapp3.1</TargetFramework>
</PropertyGroup>
```

For more information, see [Target frameworks in SDK-style projects](#).

TargetFrameworks

Use the `TargetFrameworks` property when you want your app to target multiple platforms. For a list of valid target framework monikers, see [Target frameworks in SDK-style projects](#).

NOTE

This property is ignored if `TargetFramework` (singular) is specified.

```
<PropertyGroup>
  <TargetFrameworks>netcoreapp3.1;net462</TargetFrameworks>
</PropertyGroup>
```

For more information, see [Target frameworks in SDK-style projects](#).

NetStandardImplicitPackageVersion

NOTE

This property only applies to projects using `netstandard1.x`. It doesn't apply to projects that use `netstandard2.x`.

Use the `NetStandardImplicitPackageVersion` property when you want to specify a framework version that's lower than the metapackage version. The project file in the following example targets `netstandard1.3` but uses the

1.6.0 version of `NETStandard.Library`.

```
<PropertyGroup>
  <TargetFramework>netstandard1.3</TargetFramework>
  <NetStandardImplicitPackageVersion>1.6.0</NetStandardImplicitPackageVersion>
</PropertyGroup>
```

Assembly attribute properties

- [GenerateAssemblyInfo](#)
- [GeneratedAssemblyInfoFile](#)

GenerateAssemblyInfo

The `GenerateAssemblyInfo` property controls `AssemblyInfo` attribute generation for the project. The default value is `true`. Use `false` to disable generation of the file:

```
<PropertyGroup>
  <GenerateAssemblyInfo>false</GenerateAssemblyInfo>
</PropertyGroup>
```

The `GeneratedAssemblyInfoFile` setting controls the name of the generated file.

When the `GenerateAssemblyInfo` value is `true`, [package-related project properties](#) are transformed into assembly attributes. The following table lists the project properties that generate the attributes. It also lists the properties that you can use to disable that generation on a per-attribute basis, for example:

```
<PropertyGroup>
  <GenerateNeutralResourcesLanguageAttribute>false</GenerateNeutralResourcesLanguageAttribute>
</PropertyGroup>
```

MSBUILD PROPERTY	ASSEMBLY ATTRIBUTE	PROPERTY TO DISABLE ATTRIBUTE GENERATION
<code>Company</code>	<code>AssemblyCompanyAttribute</code>	<code>GenerateAssemblyCompanyAttribute</code>
<code>Configuration</code>	<code>AssemblyConfigurationAttribute</code>	<code>GenerateAssemblyConfigurationAttribute</code>
<code>Copyright</code>	<code>AssemblyCopyrightAttribute</code>	<code>GenerateAssemblyCopyrightAttribute</code>
<code>Description</code>	<code>AssemblyDescriptionAttribute</code>	<code>GenerateAssemblyDescriptionAttribute</code>
<code>FileVersion</code>	<code>AssemblyFileVersionAttribute</code>	<code>GenerateAssemblyFileVersionAttribute</code>
<code>InformationalVersion</code>	<code>AssemblyInformationalVersionAttribute</code>	<code>GenerateAssemblyInformationalVersionAttribute</code>
<code>Product</code>	<code>AssemblyProductAttribute</code>	<code>GenerateAssemblyProductAttribute</code>
<code>AssemblyTitle</code>	<code>AssemblyTitleAttribute</code>	<code>GenerateAssemblyTitleAttribute</code>
<code> AssemblyVersion</code>	<code>AssemblyVersionAttribute</code>	<code>GenerateAssemblyVersionAttribute</code>
<code>NeutralLanguage</code>	<code>NeutralResourcesLanguageAttribute</code>	<code>GenerateNeutralResourcesLanguageAttribute</code>

Notes about these settings:

- `AssemblyVersion` and `FileVersion` default to the value of `$(Version)` without the suffix. For example, if `$(Version)` is `1.2.3-beta.4`, then the value would be `1.2.3`.
- `InformationalVersion` defaults to the value of `$(Version)`.
- If the `$(SourceRevisionId)` property is present, it's appended to `InformationalVersion`. You can disable this behavior using `IncludeSourceRevisionInInformationalVersion`.
- `Copyright` and `Description` properties are also used for NuGet metadata.
- `Configuration`, which defaults to `Debug`, is shared with all MSBuild targets. You can set it via the `--configuration` option of `dotnet` commands, for example, `dotnet pack`.
- Some of the properties are used when creating a NuGet package. For more information, see [Package properties](#).

Migrating from .NET Framework

.NET Framework project templates create a code file with these assembly info attributes set. The file is typically located at `.\Properties\AssemblyInfo.cs` or `.\Properties\AssemblyInfo.vb`. SDK-style projects generate this file for you based on the project settings. **You can't have both**. When porting your code to .NET 5 (or .NET Core 3.1) or later, do one of the following:

- Disable the generation of the temporary code file that contains the assembly info attributes by setting `GenerateAssemblyInfo` to `false` in your project file. This enables you to keep your `AssemblyInfo` file.
- Migrate the settings in the `AssemblyInfo` file to the project file, and then delete the `AssemblyInfo` file.

GeneratedAssemblyInfoFile

The `GeneratedAssemblyInfoFile` property defines the relative or absolute path of the generated assembly info file. Defaults to a file named `[project-name].AssemblyInfo.[cs/vb]` in the `$(IntermediateOutputPath)` (usually the `obj`) directory.

```
<PropertyGroup>
  <GeneratedAssemblyInfoFile>assemblyinfo.cs</GeneratedAssemblyInfoFile>
</PropertyGroup>
```

Package properties

- [Descriptive properties](#)
- [PackRelease](#)

Descriptive properties

You can specify properties such as `PackageId`, `PackageVersion`, `PackageIcon`, `Title`, and `Description` to describe the package that gets created from your project. For information about these and other properties, see [pack target](#).

```
<PropertyGroup>
  ...
  <PackageId>ClassLibDotNetStandard</PackageId>
  <Version>1.0.0</Version>
  <Authors>John Doe</Authors>
  <Company>Contoso</Company>
</PropertyGroup>
```

PackRelease

The `PackRelease` property is similar to the `PublishRelease` property, except that it changes the default behavior of `dotnet pack`.

```
<PropertyGroup>
  <PackRelease>true</PackRelease>
</PropertyGroup>
```

NOTE

To use `PackRelease` in a project that's part of a Visual Studio solution, you must set the environment variable `DOTNET_CLI_ENABLE_PACK_RELEASE_FOR_SOLUTIONS` to `true` (or any other value). Setting this variable will increase the time required to pack solutions that have many projects.

Publish-related properties

The following MSBuild properties are documented in this section:

- [AppendRuntimeIdentifierToOutputPath](#)
- [AppendTargetFrameworkToOutputPath](#)
- [CopyLocalLockFileAssemblies](#)
- [EnablePackageValidation](#)
- [ErrorOnDuplicatePublishOutputFiles](#)
- [GenerateRuntimeConfigDevFile](#)
- [GenerateRuntimeConfigurationFiles](#)
- [GenerateSatelliteAssembliesForCore](#)
- [IsPublishable](#)
- [PreserveCompilationContext](#)
- [PreserveCompilationReferences](#)
- [ProduceReferenceAssemblyInOutDir](#)
- [PublishRelease](#)
- [RollForward](#)
- [RuntimeFrameworkVersion](#)
- [RuntimeIdentifier](#)
- [RuntimeIdentifiers](#)
- [SatelliteResourceLanguages](#)
- [UseAppHost](#)

AppendTargetFrameworkToOutputPath

The `AppendTargetFrameworkToOutputPath` property controls whether the [target framework moniker \(TFM\)](#) is appended to the output path (which is defined by [OutputPath](#)). The .NET SDK automatically appends the target framework and, if present, the runtime identifier to the output path. Setting `AppendTargetFrameworkToOutputPath` to `false` prevents the TFM from being appended to the output path. However, without the TFM in the output path, multiple build artifacts may overwrite each other.

For example, for a .NET 5 app, the output path changes from `bin\Debug\net5.0` to `bin\Debug` with the following setting:

```
<PropertyGroup>
  <AppendTargetFrameworkToOutputPath>false</AppendTargetFrameworkToOutputPath>
</PropertyGroup>
```

AppendRuntimeIdentifierToOutputPath

The `AppendRuntimeIdentifierToOutputPath` property controls whether the [runtime identifier \(RID\)](#) is appended to the output path. The .NET SDK automatically appends the target framework and, if present, the runtime identifier

to the output path. Setting `AppendRuntimeIdentifierToOutputPath` to `false` prevents the RID from being appended to the output path.

For example, for a .NET 5 app and an RID of `win10-x64`, the output path changes from `bin\Debug\net5.0\win10-x64` to `bin\Debug\net5.0` with the following setting:

```
<PropertyGroup>
  <AppendRuntimeIdentifierToOutputPath>false</AppendRuntimeIdentifierToOutputPath>
</PropertyGroup>
```

CopyLocalLockFileAssemblies

The `CopyLocalLockFileAssemblies` property is useful for plugin projects that have dependencies on other libraries. If you set this property to `true`, any NuGet package dependencies are copied to the output directory. That means you can use the output of `dotnet build` to run your plugin on any machine.

```
<PropertyGroup>
  <CopyLocalLockFileAssemblies>true</CopyLocalLockFileAssemblies>
</PropertyGroup>
```

TIP

Alternatively, you can use `dotnet publish` to publish the class library. For more information, see [dotnet publish](#).

ErrorOnDuplicatePublishOutputFiles

The `ErrorOnDuplicatePublishOutputFiles` property relates to whether the SDK generates error NETSDK1148 when MSBuild detects duplicate files in the publish output, but can't determine which files to remove. Set the `ErrorOnDuplicatePublishOutputFiles` property to `false` if you don't want the error to be generated.

```
<PropertyGroup>
  <ErrorOnDuplicatePublishOutputFiles>false</ErrorOnDuplicatePublishOutputFiles>
</PropertyGroup>
```

This property was introduced in .NET 6.

EnablePackageValidation

The `EnablePackageValidation` property enables a series of validations on the package after the `pack` task. For more information, see [package validation](#).

```
<PropertyGroup>
  <EnablePackageValidation>true</EnablePackageValidation>
</PropertyGroup>
```

This property was introduced in .NET 6.

GenerateRuntimeConfigDevFile

Starting with the .NET 6 SDK, the `[Appname].runtimesettings.dev.json` file is [no longer generated by default](#) at compile time. If you still want this file to be generated, set the `GenerateRuntimeConfigDevFile` property to `true`.

```
<PropertyGroup>
  <GenerateRuntimeConfigDevFile>true</GenerateRuntimeConfigDevFile>
</PropertyGroup>
```

GenerateRuntimeConfigurationFiles

The `GenerateRuntimeConfigurationFiles` property controls whether runtime configuration options are copied from the `runtimeconfig.template.json` file to the `[appname].runtimeconfig.json` file. For apps that require a `runtimeconfig.json` file, that is, those whose `OutputType` is `Exe`, this property defaults to `true`.

```
<PropertyGroup>
  <GenerateRuntimeConfigurationFiles>true</GenerateRuntimeConfigurationFiles>
</PropertyGroup>
```

GenerateSatelliteAssembliesForCore

The `GenerateSatelliteAssembliesForCore` property controls whether satellite assemblies are generated using `csc.exe` or [Al.exe \(Assembly Linker\)](#) in .NET Framework projects. (.NET Core and .NET 5+ projects always use `csc.exe` to generate satellite assemblies.) For .NET Framework projects, satellite assemblies are created by `al.exe`, by default. By setting the `GenerateSatelliteAssembliesForCore` property to `true`, satellite assemblies are created by `csc.exe` instead. Using `csc.exe` can be advantageous in the following situations:

- You want to use the C# compiler [deterministic](#) option.
- You're limited by the fact that `al.exe` has no support for public signing and handles [AssemblyInformationalVersionAttribute](#) poorly.

```
<PropertyGroup>
  <GenerateSatelliteAssembliesForCore>true</GenerateSatelliteAssembliesForCore>
</PropertyGroup>
```

IsPublishable

The `IsPublishable` property allows the `Publish` target to run. This property only affects processes that use `*.proj` files and the `Publish` target, such as the `dotnet publish` command. It does not affect publishing in Visual Studio, which uses the `PublishOnly` target. The default value is `true`.

This property is useful if you run `dotnet publish` on a solution file, as it allows automatic selection of projects that should be published.

```
<PropertyGroup>
  <IsPublishable>false</IsPublishable>
</PropertyGroup>
```

PreserveCompilationContext

The `PreserveCompilationContext` property allows a built or published application to compile more code at run time using the same settings that were used at build time. The assemblies referenced at build time will be copied into the `ref` subdirectory of the output directory. The names of the reference assemblies are stored in the application's `.deps.json` file along with the options passed to the compiler. You can retrieve this information using the [DependencyContext.CompileLibraries](#) and [DependencyContext.CompilationOptions](#) properties.

This functionality is mostly used internally by ASP.NET Core MVC and Razor pages to support run-time compilation of Razor files.

```
<PropertyGroup>
  <PreserveCompilationContext>true</PreserveCompilationContext>
</PropertyGroup>
```

PreserveCompilationReferences

The `PreserveCompilationReferences` property is similar to the `PreserveCompilationContext` property, except that it only copies the referenced assemblies to the publish directory, and not the `.deps.json` file.

```
<PropertyGroup>
  <PreserveCompilationReferences>true</PreserveCompilationReferences>
</PropertyGroup>
```

For more information, see [Razor SDK properties](#).

ProduceReferenceAssemblyInOutDir

In .NET 5 and earlier versions, reference assemblies are always written to the `OutDir` directory. In .NET 6 and later versions, you can use the `ProduceReferenceAssemblyInOutDir` property to control whether reference assemblies are written to the `OutDir` directory. The default value is `false`, and reference assemblies are only written to the `IntermediateOutputPath` directory. Set the value to `true` to write reference assemblies to the `OutDir` directory.

```
<PropertyGroup>
  <ProduceReferenceAssemblyInOutDir>true</ProduceReferenceAssemblyInOutDir>
</PropertyGroup>
```

For more information, see [Write reference assemblies to intermediate output](#).

PublishRelease

The `PublishRelease` property informs `dotnet publish` to use the `Release` configuration by default instead of the `Debug` configuration.

```
<PropertyGroup>
  <PublishRelease>true</PublishRelease>
</PropertyGroup>
```

NOTE

- This property does not affect the behavior of `dotnet build /t:Publish` and changes the configuration only when publishing via the .NET CLI.
- To use `PublishRelease` in a project that's part of a Visual Studio solution, you must set the environment variable `DOTNET_CLI_ENABLE_PUBLISH_RELEASE_FOR_SOLUTIONS` to `true` (or any other value). This will increase the time required to publish solutions that have many projects. When publishing a solution with this variable enabled, the executable project's `PublishRelease` value takes precedence and flows the new default configuration to any other projects in the solution. If a solution contains multiple executable or top-level projects with differing values of `PublishRelease`, the solution won't successfully publish.

RollForward

The `RollForward` property controls how the application chooses a runtime when multiple runtime versions are available. This value is output to the `.runtimeconfig.json` as the `rollForward` setting.

```
<PropertyGroup>
  <RollForward>LatestMinor</RollForward>
</PropertyGroup>
```

Set `RollForward` to one of the following values:

VALUE	DESCRIPTION
-------	-------------

VALUE	DESCRIPTION
Minor	Default if not specified. Roll-forward to the lowest higher minor version, if requested minor version is missing. If the requested minor version is present, then the <code>LatestPatch</code> policy is used.
Major	Roll-forward to the next available higher major version, and lowest minor version, if requested major version is missing. If the requested major version is present, then the <code>Minor</code> policy is used.
LatestPatch	Roll-forward to the highest patch version. This value disables minor version roll-forward.
LatestMinor	Roll-forward to highest minor version, even if requested minor version is present.
LatestMajor	Roll-forward to highest major and highest minor version, even if requested major is present.
Disable	Don't roll-forward, only bind to the specified version. This policy isn't recommended for general use since it disables the ability to roll-forward to the latest patches. This value is only recommended for testing.

For more information, see [Control roll-forward behavior](#).

RuntimeFrameworkVersion

The `RuntimeFrameworkVersion` property specifies the version of the runtime to use when publishing. Specify a runtime version:

```
<PropertyGroup>
  <RuntimeFrameworkVersion>5.0.7</RuntimeFrameworkVersion>
</PropertyGroup>
```

When publishing a framework-dependent application, this value specifies the *minimum* version required. When publishing a self-contained application, this value specifies the *exact* version required.

RuntimeIdentifier

The `RuntimeIdentifier` property lets you specify a single [runtime identifier \(RID\)](#) for the project. The RID enables publishing a self-contained deployment.

```
<PropertyGroup>
  <RuntimeIdentifier>ubuntu.16.04-x64</RuntimeIdentifier>
</PropertyGroup>
```

RuntimeIdentifiers

The `RuntimeIdentifiers` property lets you specify a semicolon-delimited list of [runtime identifiers \(IDs\)](#) for the project. Use this property if you need to publish for multiple runtimes. `RuntimeIdentifiers` is used at restore time to ensure the right assets are in the graph.

TIP

`RuntimeIdentifier` (singular) can provide faster builds when only a single runtime is required.

```
<PropertyGroup>
  <RuntimeIdentifiers>win10-x64;osx.10.11-x64;ubuntu.16.04-x64</RuntimeIdentifiers>
</PropertyGroup>
```

SatelliteResourceLanguages

The `SatelliteResourceLanguages` property lets you specify which languages you want to preserve satellite resource assemblies for during build and publish. Many NuGet packages include localized resource satellite assemblies in the main package. For projects that reference these NuGet packages that don't require localized resources, the localized assemblies can unnecessarily inflate the build and publish output size. By adding the `SatelliteResourceLanguages` property to your project file, only localized assemblies for the languages you specify will be included in the build and publish output. For example, in the following project file, only English (US) and German (Germany) resource satellite assemblies will be retained.

```
<PropertyGroup>
  <SatelliteResourceLanguages>en-US;de-DE</SatelliteResourceLanguages>
</PropertyGroup>
```

NOTE

- You must specify this property in the project that references the NuGet package with localized resource satellite assemblies.
- To specify multiple languages as an argument to `dotnet publish`, you must add *three pairs* of quotation marks around the language identifiers. For example:

```
dotnet msbuild multi.msbuildproj -p:SatelliteResourceLanguages=""de;en""
```

UseAppHost

The `UseAppHost` property controls whether or not a native executable is created for a deployment. A native executable is required for self-contained deployments.

In .NET Core 3.0 and later versions, a framework-dependent executable is created by default. Set the `UseAppHost` property to `false` to disable generation of the executable.

```
<PropertyGroup>
  <UseAppHost>false</UseAppHost>
</PropertyGroup>
```

For more information about deployment, see [.NET application deployment](#).

Trim-related properties

Numerous MSBuild properties are available to fine tune trimming, which is a feature that trims unused code from self-contained deployments. These options are discussed in detail at [Trimming options](#). The following table provides a quick reference.

PROPERTY	VALUES	DESCRIPTION
<code>PublishTrimmed</code>	<code>true</code> or <code>false</code>	Controls whether trimming is enabled during publish.
<code>TrimMode</code>	<code>full</code> or <code>partial</code>	Default is <code>full</code> . Controls the trimming granularity.

PROPERTY	VALUES	DESCRIPTION
<code>SuppressTrimAnalysisWarnings</code>	<code>true</code> or <code>false</code>	Controls whether trim analysis warnings are produced.
<code>EnableTrimAnalyzer</code>	<code>true</code> or <code>false</code>	Controls whether a subset of trim analysis warnings are produced. You can enable analysis even if <code>PublishTrimmed</code> is set to <code>false</code> .
<code>ILLinkTreatWarningsAsErrors</code>	<code>true</code> or <code>false</code>	Controls whether trim warnings are treated as errors. For example, you may want to set this property to <code>false</code> when <code>TreatWarningsAsErrors</code> is set to <code>true</code> .
<code>TrimmerSingleWarn</code>	<code>true</code> or <code>false</code>	Controls whether a single warning per assembly is shown or all warnings.
<code>TrimmerRemoveSymbols</code>	<code>true</code> or <code>false</code>	Controls whether all symbols are removed from a trimmed application.

Compilation-related properties

The following MSBuild properties are documented in this section:

- [DisableImplicitFrameworkDefines](#)
- [DocumentationFile](#)
- [EmbeddedResourceUseDependentUponConvention](#)
- [EnablePreviewFeatures](#)
- [GenerateDocumentationFile](#)
- [GenerateRequiresPreviewFeaturesAttribute](#)
- [OptimizeImplicitlyTriggeredBuild](#)

C# compiler options, such as `LangVersion` and `Nullable`, can also be specified as MSBuild properties in your project file. For more information, see [C# compiler options](#).

DisableImplicitFrameworkDefines

The `DisableImplicitFrameworkDefines` property controls whether or not the SDK generates preprocessor symbols for the target framework and platform for the .NET project. When this property is set to `false` or is unset (which is the default value) preprocessor symbols are generated for:

- Framework without version (`NETFRAMEWORK`, `NETSTANDARD`, `NET`)
- Framework with version (`NET48`, `NETSTANDARD2_0`, `NET6_0`)
- Framework with version minimum bound (`NET48_OR_GREATER`, `NETSTANDARD2_0_OR_GREATER`, `NET6_0_OR_GREATER`)

For more information on target framework monikers and these implicit preprocessor symbols, see [Target frameworks](#).

Additionally, if you specify an operating system-specific target framework in the project (for example `net6.0-android`), the following preprocessor symbols are generated:

- Platform without version (`ANDROID`, `IOS`, `WINDOWS`)
- Platform with version (`IOS15_1`)

- Platform with version minimum bound (`IOS15_1_OR_GREATER`)

For more information on operating system-specific target framework monikers, see [OS-specific TFMs](#).

Finally, if your target framework implies support for older target frameworks, preprocessor symbols for those older frameworks are emitted. For example, `net6.0` implies support for `net5.0` and so on all the way back to `.netcoreapp1.0`. So for each of these target frameworks, the *Framework with version minimum bound* symbol will be defined.

DocumentationFile

The `DocumentationFile` property lets you specify a file name for the XML file that contains the documentation for your library. For IntelliSense to function correctly with your documentation, the file name must be the same as the assembly name and must be in the same directory as the assembly. If you don't specify this property but you do set `GenerateDocumentationFile` to `true`, the name of the documentation file defaults to the name of your assembly but with an `.xml` file extension. For this reason, it's often easier to omit this property and use the `GenerateDocumentationFile` property instead.

If you specify this property but you set `GenerateDocumentationFile` to `false`, the compiler *does not* generate a documentation file. If you specify this property and omit the `GenerateDocumentationFile` property, the compiler *does* generate a documentation file.

```
<PropertyGroup>
  <DocumentationFile>path/to/file.xml</DocumentationFile>
</PropertyGroup>
```

EmbeddedResourceUseDependentUponConvention

The `EmbeddedResourceUseDependentUponConvention` property defines whether resource manifest file names are generated from type information in source files that are co-located with resource files. For example, if `Form1.resx` is in the same folder as `Form1.cs`, and `EmbeddedResourceUseDependentUponConvention` is set to `true`, the generated `.resources` file takes its name from the first type that's defined in `Form1.cs`. For example, if `MyNamespace.Form1` is the first type defined in `Form1.cs`, the generated file name is `MyNamespace.Form1.resources`.

NOTE

If `LogicalName`, `ManifestResourceName`, or `DependentUpon` metadata is specified for an `EmbeddedResource` item, the generated manifest file name for that resource file is based on that metadata instead.

By default, in a new .NET project, this property is set to `true`. If set to `false`, and no `LogicalName`, `ManifestResourceName`, or `DependentUpon` metadata is specified for the `EmbeddedResource` item in the project file, the resource manifest file name is based off the root namespace for the project and the relative file path to the `.resx` file. For more information, see [How resource manifest files are named](#).

```
<PropertyGroup>
  <EmbeddedResourceUseDependentUponConvention>true</EmbeddedResourceUseDependentUponConvention>
</PropertyGroup>
```

EnablePreviewFeatures

The `EnablePreviewFeatures` property defines whether your project depends on any APIs or assemblies that are decorated with the `RequiresPreviewFeaturesAttribute` attribute. This attribute is used to signify that an API or assembly uses features that are considered to be in *preview* for the SDK version you're using. Preview features are not supported and may be removed in a future version. To enable the use of preview features, set the property to `True`.

```
<PropertyGroup>
  <EnablePreviewFeatures>True</EnablePreviewFeatures>
</PropertyGroup>
```

When a project contains this property set to `True`, the following assembly-level attribute is added to the `AssemblyInfo.cs` file:

```
[assembly: RequiresPreviewFeatures]
```

An analyzer warns if this attribute is present on dependencies for projects where `EnablePreviewFeatures` is not set to `True`.

Library authors who intend to ship preview assemblies should set this property to `True`. If an assembly needs to ship with a mixture of preview and non-preview APIs, see the [GenerateRequiresPreviewFeaturesAttribute](#) section below.

GenerateDocumentationFile

The `GenerateDocumentationFile` property controls whether the compiler generates an XML documentation file for your library. If you set this property to `true` and you don't specify a file name via the [DocumentationFile property](#), the generated XML file is placed in the same output directory as your assembly and has the same file name (but with an `.xml` extension).

```
<PropertyGroup>
  <GenerateDocumentationFile>true</GenerateDocumentationFile>
</PropertyGroup>
```

For more information about generating documentation from code comments, see [XML documentation comments \(C#\)](#), [Document your code with XML \(Visual Basic\)](#), or [Document your code with XML \(F#\)](#).

GenerateRequiresPreviewFeaturesAttribute

The `GenerateRequiresPreviewFeaturesAttribute` property is closely related to the `EnablePreviewFeatures` property. If your library uses preview features but you don't want the entire assembly to be marked with the `RequiresPreviewFeaturesAttribute` attribute, which would require any consumers to [enable preview features](#), set this property to `False`.

```
<PropertyGroup>
  <EnablePreviewFeatures>True</EnablePreviewFeatures>
  <GenerateRequiresPreviewFeaturesAttribute>False</GenerateRequiresPreviewFeaturesAttribute>
</PropertyGroup>
```

IMPORTANT

If you set the `GenerateRequiresPreviewFeaturesAttribute` property to `False`, you must be certain to decorate all public APIs that rely on preview features with `RequiresPreviewFeaturesAttribute`.

OptimizeImplicitlyTriggeredBuild

To speed up the build time, builds that are implicitly triggered by Visual Studio skip code analysis, including nullable analysis. Visual Studio triggers an implicit build when you run tests, for example. However, implicit builds are optimized only when `TreatWarningsAsErrors` is not `true`. If you have `TreatWarningsAsErrors` set to `true` but you still want implicitly triggered builds to be optimized, you can set

`OptimizeImplicitlyTriggeredBuild` to `True`. To turn off build optimization for implicitly triggered builds, set `OptimizeImplicitlyTriggeredBuild` to `False`.

```
<PropertyGroup>
  <OptimizeImplicitlyTriggeredBuild>True</OptimizeImplicitlyTriggeredBuild>
</PropertyGroup>
```

Default item inclusion properties

The following MSBuild properties are documented in this section:

- [DefaultItemExcludesInProjectFolder](#)
- [DefaultItemExcludes](#)
- [EnableDefaultCompileItems](#)
- [EnableDefaultEmbeddedResourceItems](#)
- [EnableDefaultItems](#)
- [EnableDefaultNoneItems](#)

For more information, see [Default includes and excludes](#).

DefaultItemExcludes

Use the `DefaultItemExcludes` property to define glob patterns for files and folders that should be excluded from the include, exclude, and remove globs. By default, the `./bin` and `./obj` folders are excluded from the glob patterns.

```
<PropertyGroup>
  <DefaultItemExcludes>$({DefaultItemExcludes});**/*.*myextension</DefaultItemExcludes>
</PropertyGroup>
```

DefaultItemExcludesInProjectFolder

Use the `DefaultItemExcludesInProjectFolder` property to define glob patterns for files and folders in the project folder that should be excluded from the include, exclude, and remove globs. By default, folders that start with a period (`.`), such as `.git` and `.vs`, are excluded from the glob patterns.

This property is very similar to the `DefaultItemExcludes` property, except that it only considers files and folders in the project folder. When a glob pattern would unintentionally match items outside the project folder with a relative path, use the `DefaultItemExcludesInProjectFolder` property instead of the `DefaultItemExcludes` property.

```
<PropertyGroup>
  <DefaultItemExcludesInProjectFolder>$({DefaultItemExcludesInProjectFolder});**/myprefix/**</DefaultItemExcludesInProjectFolder>
</PropertyGroup>
```

EnableDefaultItems

The `EnableDefaultItems` property controls whether compile items, embedded resource items, and `None` items are implicitly included in the project. The default value is `true`. Set the `EnableDefaultItems` property to `false` to disable all implicit file inclusion.

```
<PropertyGroup>
  <EnableDefaultItems>false</EnableDefaultItems>
</PropertyGroup>
```

EnableDefaultCompileItems

The `EnableDefaultCompileItems` property controls whether compile items are implicitly included in the project. The default value is `true`. Set the `EnableDefaultCompileItems` property to `false` to disable implicit inclusion of

*.cs and other language-extension files.

```
<PropertyGroup>
  <EnableDefaultCompileItems>false</EnableDefaultCompileItems>
</PropertyGroup>
```

EnableDefaultEmbeddedResourceItems

The `EnableDefaultEmbeddedResourceItems` property controls whether embedded resource items are implicitly included in the project. The default value is `true`. Set the `EnableDefaultEmbeddedResourceItems` property to `false` to disable implicit inclusion of embedded resource files.

```
<PropertyGroup>
  <EnableDefaultEmbeddedResourceItems>false</EnableDefaultEmbeddedResourceItems>
</PropertyGroup>
```

EnableDefaultNoneItems

The `EnableDefaultNoneItems` property controls whether `None` items (files that have no role in the build process) are implicitly included in the project. The default value is `true`. Set the `EnableDefaultNoneItems` property to `false` to disable implicit inclusion of `None` items.

```
<PropertyGroup>
  <EnableDefaultNoneItems>false</EnableDefaultNoneItems>
</PropertyGroup>
```

Code analysis properties

The following MSBuild properties are documented in this section:

- [AnalysisLevel](#)
- [AnalysisLevel<Category>](#)
- [AnalysisMode](#)
- [AnalysisMode<Category>](#)
- [CodeAnalysisTreatWarningsAsErrors](#)
- [EnableNETAnalyzers](#)
- [EnforceCodeStyleInBuild](#)
- [_SkipUpgradeNetAnalyzersNuGetWarning](#)

AnalysisLevel

The `AnalysisLevel` property lets you specify a set of code analyzers to run according to a .NET release. Each .NET release, starting in .NET 5, has a set of code analysis rules. Of that set, the rules that are enabled by default for that release will analyze your code. For example, if you upgrade to .NET 6 but don't want the default set of code analysis rules to change, set `AnalysisLevel` to `5`.

```
<PropertyGroup>
  <AnalysisLevel>preview</AnalysisLevel>
</PropertyGroup>
```

Optionally, starting in .NET 6, you can specify a compound value for this property that also specifies how aggressively to enable rules. Compound values take the form `<version>-<mode>`, where the `<mode>` value is one of the [AnalysisMode](#) values. The following example uses the preview version of code analyzers, and enables the recommended set of rules.

```
<PropertyGroup>
  <AnalysisLevel>preview-recommended</AnalysisLevel>
</PropertyGroup>
```

NOTE

If you set `AnalysisLevel` to `5-<mode>` or `5.0-<mode>` and then install the .NET 6 SDK and recompile your project, you may see unexpected new build warnings. For more information, see [dotnet/roslyn-analyzers#5679](#).

Default value:

- If your project targets .NET 5 or later, or if you've added the `AnalysisMode` property, the default value is `latest`.
- Otherwise, this property is omitted unless you explicitly add it to the project file.

The following table shows the values you can specify.

VALUE	MEANING
<code>latest</code>	The latest code analyzers that have been released are used. This is the default.
<code>latest-<mode></code>	The latest code analyzers that have been released are used. The <code><mode></code> value determines which rules are enabled.
<code>preview</code>	The latest code analyzers are used, even if they are in preview.
<code>preview-<mode></code>	The latest code analyzers are used, even if they are in preview. The <code><mode></code> value determines which rules are enabled.
<code>6.0</code>	The set of rules that was available for the .NET 6 release is used, even if newer rules are available.
<code>6.0-<mode></code>	The set of rules that was available for the .NET 6 release is used, even if newer rules are available. The <code><mode></code> value determines which rules are enabled.
<code>6</code>	The set of rules that was available for the .NET 6 release is used, even if newer rules are available.
<code>6-<mode></code>	The set of rules that was available for the .NET 6 release is used, even if newer rules are available. The <code><mode></code> value determines which rules are enabled.
<code>5.0</code>	The set of rules that was available for the .NET 5 release is used, even if newer rules are available.
<code>5.0-<mode></code>	The set of rules that was available for the .NET 5 release is used, even if newer rules are available. The <code><mode></code> value determines which rules are enabled.
<code>5</code>	The set of rules that was available for the .NET 5 release is used, even if newer rules are available.

VALUE	MEANING
5- <code><mode></code>	The set of rules that was available for the .NET 5 release is used, even if newer rules are available. The <code><mode></code> value determines which rules are enabled.

NOTE

- In .NET 5 and earlier versions, this property only affects [code-quality \(CAXXXX\) rules](#). Starting in .NET 6, if you set `EnforceCodeStyleInBuild` to `true`, this property affects [code-style \(IDEXXXX\) rules](#) too.
- If you set a compound value for `AnalysisLevel`, you don't need to specify an `AnalysisMode`. However, if you do, `AnalysisLevel` takes precedence over `AnalysisMode`.
- This property has no effect on code analysis in projects that don't reference a [project SDK](#), for example, legacy .NET Framework projects that reference the `Microsoft.CodeAnalysis.NetAnalyzers` NuGet package.

AnalysisLevel<Category>

Introduced in .NET 6, this property is the same as [AnalysisLevel](#), except that it only applies to a specific [category of code-analysis rules](#). This property allows you to use a different version of code analyzers for a specific category, or to enable or disable rules at a different level to the other rule categories. If you omit this property for a particular category of rules, it defaults to the [AnalysisLevel](#) value. The available values are the same as those for [AnalysisLevel](#).

```
<PropertyGroup>
  <AnalysisLevelSecurity>preview</AnalysisLevelSecurity>
</PropertyGroup>
```

```
<PropertyGroup>
  <AnalysisLevelSecurity>preview-recommended</AnalysisLevelSecurity>
</PropertyGroup>
```

The following table lists the property name for each rule category.

PROPERTY NAME	RULE CATEGORY
<code><AnalysisLevelDesign></code>	Design rules
<code><AnalysisLevelDocumentation></code>	Documentation rules
<code><AnalysisLevelGlobalization></code>	Globalization rules
<code><AnalysisLevelInteroperability></code>	Portability and interoperability rules
<code><AnalysisLevelMaintainability></code>	Maintainability rules
<code><AnalysisLevelNaming></code>	Naming rules
<code><AnalysisLevelPerformance></code>	Performance rules
<code><AnalysisLevelSingleFile></code>	Single-file application rules
<code><AnalysisLevelReliability></code>	Reliability rules

PROPERTY NAME	RULE CATEGORY
<AnalysisLevelSecurity>	Security rules
<AnalysisLevelStyle>	Code-style (IDEXXX) rules
<AnalysisLevelUsage>	Usage rules

AnalysisMode

Starting with .NET 5, the .NET SDK ships with all of the "CA" code quality rules. By default, only [some rules are enabled](#) as build warnings in each .NET release. The `AnalysisMode` property lets you customize the set of rules that are enabled by default. You can either switch to a more aggressive analysis mode where you can opt out of rules individually, or a more conservative analysis mode where you can opt in to specific rules. For example, if you want to enable all rules as build warnings, set the value to `All`.

```
<PropertyGroup>
  <AnalysisMode>All</AnalysisMode>
</PropertyGroup>
```

The following table shows the available option values in .NET 5 and .NET 6. They're listed in increasing order of the number of rules they enable.

.NET 6+ VALUE	.NET 5 VALUE	MEANING
<code>None</code>	<code>AllDisabledByDefault</code>	All rules are disabled. You can selectively opt in to individual rules to enable them.
<code>Default</code>	<code>Default</code>	Default mode, where certain rules are enabled as build warnings, certain rules are enabled as Visual Studio IDE suggestions, and the remainder are disabled.
<code>Minimum</code>	N/A	More aggressive mode than the <code>Default</code> mode. Certain suggestions that are highly recommended for build enforcement are enabled as build warnings.
<code>Recommended</code>	N/A	More aggressive mode than the <code>Minimum</code> mode, where more rules are enabled as build warnings.
<code>All</code>	<code>AllEnabledByDefault</code>	All rules are enabled as build warnings. You can selectively opt out of individual rules to disable them.

NOTE

- In .NET 5, this property only affects [code-quality \(CAXXXX\) rules](#). Starting in .NET 6, if you set `EnforceCodeStyleInBuild` to `true`, this property affects [code-style \(IDEXXXX\) rules](#) too.
- If you use a compound value for `AnalysisLevel`, for example, `<AnalysisLevel>5-recommended</AnalysisLevel>`, you can omit this property entirely. However, if you specify both properties, `AnalysisLevel` takes precedence over `AnalysisMode`.
- If `AnalysisMode` is set to `AllEnabledByDefault` and `AnalysisLevel` is set to `5` or `5.0`, and then you install the .NET 6 SDK and recompile your project, you may see unexpected new build warnings. For more information, see [dotnet/roslyn-analyzers#5679](#).
- This property has no effect on code analysis in projects that don't reference a [project SDK](#), for example, legacy .NET Framework projects that reference the `Microsoft.CodeAnalysis.NetAnalyzers` NuGet package.

AnalysisMode <Category>

Introduced in .NET 6, this property is the same as [AnalysisMode](#), except that it only applies to a specific [category of code-analysis rules](#). This property allows you to enable or disable rules at a different level to the other rule categories. If you omit this property for a particular category of rules, it defaults to the [AnalysisMode](#) value. The available values are the same as those for [AnalysisMode](#).

```
<PropertyGroup>
  <AnalysisModeSecurity>All</AnalysisModeSecurity>
</PropertyGroup>
```

The following table lists the property name for each rule category.

PROPERTY NAME	RULE CATEGORY
<code><AnalysisModeDesign></code>	Design rules
<code><AnalysisModeDocumentation></code>	Documentation rules
<code><AnalysisModeGlobalization></code>	Globalization rules
<code><AnalysisModeInteroperability></code>	Portability and interoperability rules
<code><AnalysisModeMaintainability></code>	Maintainability rules
<code><AnalysisModeNaming></code>	Naming rules
<code><AnalysisModePerformance></code>	Performance rules
<code><AnalysisModeSingleFile></code>	Single-file application rules
<code><AnalysisModeReliability></code>	Reliability rules
<code><AnalysisModeSecurity></code>	Security rules
<code><AnalysisModeStyle></code>	Code-style (IDEXXXX) rules
<code><AnalysisModeUsage></code>	Usage rules

CodeAnalysisTreatWarningsAsErrors

The `CodeAnalysisTreatWarningsAsErrors` property lets you configure whether code quality analysis warnings (CAxxx) should be treated as warnings and break the build. If you use the `-warnaserror` flag when you build your projects, [.NET code quality analysis](#) warnings are also treated as errors. If you do not want code quality analysis warnings to be treated as errors, you can set the `CodeAnalysisTreatWarningsAsErrors` MSBuild property to `false` in your project file.

```
<PropertyGroup>
  <CodeAnalysisTreatWarningsAsErrors>false</CodeAnalysisTreatWarningsAsErrors>
</PropertyGroup>
```

EnableNETAnalyzers

[.NET code quality analysis](#) is enabled, by default, for projects that target .NET 5 or a later version. If you're developing using the .NET 5+ SDK, you can enable .NET code analysis for SDK-style projects that target earlier versions of .NET by setting the `EnableNETAnalyzers` property to `true`. To disable code analysis in any project, set this property to `false`.

```
<PropertyGroup>
  <EnableNETAnalyzers>true</EnableNETAnalyzers>
</PropertyGroup>
```

NOTE

This property applies specifically to the built-in analyzers in the .NET 5+ SDK. It should not be used when you install a NuGet code analysis package.

EnforceCodeStyleInBuild

[.NET code style analysis](#) is disabled, by default, on build for all .NET projects. You can enable code style analysis for .NET projects by setting the `EnforceCodeStyleInBuild` property to `true`.

```
<PropertyGroup>
  <EnforceCodeStyleInBuild>true</EnforceCodeStyleInBuild>
</PropertyGroup>
```

All code style rules that are [configured](#) to be warnings or errors will execute on build and report violations.

NOTE

If you install .NET 6 (or Visual Studio 2022, which includes .NET 6) but want to build your project using Visual Studio 2019, you might see new CS8032 warnings if you have the `EnforceCodeStyleInBuild` property set to `true`. To resolve the warnings, you can specify the version of the .NET SDK to build your project with (in this case, something like `5.0.404`) by adding a [global.json entry](#).

SkipUpgradeNetAnalyzersNuGetWarning

The `_SkipUpgradeNetAnalyzersNuGetWarning` property lets you configure whether you receive a warning if you're using code analyzers from a NuGet package that's out-of-date when compared with the code analyzers in the latest .NET SDK. The warning looks similar to:

The .NET SDK has newer analyzers with version '6.0.0' than what version '5.0.3' of 'Microsoft.CodeAnalysis.NetAnalyzers' package provides. Update or remove this package reference.

To remove this warning and continue to use the version of code analyzers in the NuGet package, set `_SkipUpgradeNetAnalyzersNuGetWarning` to `true` in your project file.

```
<PropertyGroup>
  <_SkipUpgradeNetAnalyzersNuGetWarning>true</_SkipUpgradeNetAnalyzersNuGetWarning>
</PropertyGroup>
```

Runtime configuration properties

You can configure some run-time behaviors by specifying MSBuild properties in the project file of the app. For information about other ways of configuring run-time behavior, see [Runtime configuration settings](#).

- [ConcurrentGarbageCollection](#)
- [InvariantGlobalization](#)
- [PredefinedCulturesOnly](#)
- [RetainVMGarbageCollection](#)
- [ServerGarbageCollection](#)
- [ThreadPoolMaxThreads](#)
- [ThreadPoolMinThreads](#)
- [TieredCompilation](#)
- [TieredCompilationQuickJit](#)
- [TieredCompilationQuickJitForLoops](#)

ConcurrentGarbageCollection

The `ConcurrentGarbageCollection` property configures whether [background \(concurrent\) garbage collection](#) is enabled. Set the value to `false` to disable background garbage collection. For more information, see [Background GC](#).

```
<PropertyGroup>
  <ConcurrentGarbageCollection>false</ConcurrentGarbageCollection>
</PropertyGroup>
```

InvariantGlobalization

The `InvariantGlobalization` property configures whether the app runs in [globalization-invariant mode](#), which means it doesn't have access to culture-specific data. Set the value to `true` to run in globalization-invariant mode. For more information, see [Invariant mode](#).

```
<PropertyGroup>
  <InvariantGlobalization>true</InvariantGlobalization>
</PropertyGroup>
```

PredefinedCulturesOnly

In .NET 6 and later versions, the `PredefinedCulturesOnly` property configures whether apps can create cultures other than the invariant culture when [globalization-invariant mode](#) is enabled. The default is `true`. Set the value to `false` to allow creation of any new culture in globalization-invariant mode.

```
<PropertyGroup>
  <PredefinedCulturesOnly>false</PredefinedCulturesOnly>
</PropertyGroup>
```

For more information, see [Culture creation and case mapping in globalization-invariant mode](#).

RetainVMGarbageCollection

The `RetainVMGarbageCollection` property configures the garbage collector to put deleted memory segments on a standby list for future use or release them. Setting the value to `true` tells the garbage collector to put the

segments on a standby list. For more information, see [Retain VM](#).

```
<PropertyGroup>
  <RetainVMGarbageCollection>true</RetainVMGarbageCollection>
</PropertyGroup>
```

ServerGarbageCollection

The `ServerGarbageCollection` property configures whether the application uses [workstation garbage collection](#) or [server garbage collection](#). Set the value to `true` to use server garbage collection. For more information, see [Workstation vs. server](#).

```
<PropertyGroup>
  <ServerGarbageCollection>true</ServerGarbageCollection>
</PropertyGroup>
```

ThreadPoolMaxThreads

The `ThreadPoolMaxThreads` property configures the maximum number of threads for the worker thread pool. For more information, see [Maximum threads](#).

```
<PropertyGroup>
  <ThreadPoolMaxThreads>20</ThreadPoolMaxThreads>
</PropertyGroup>
```

ThreadPoolMinThreads

The `ThreadPoolMinThreads` property configures the minimum number of threads for the worker thread pool. For more information, see [Minimum threads](#).

```
<PropertyGroup>
  <ThreadPoolMinThreads>4</ThreadPoolMinThreads>
</PropertyGroup>
```

TieredCompilation

The `TieredCompilation` property configures whether the just-in-time (JIT) compiler uses [tiered compilation](#). Set the value to `false` to disable tiered compilation. For more information, see [Tiered compilation](#).

```
<PropertyGroup>
  <TieredCompilation>false</TieredCompilation>
</PropertyGroup>
```

TieredCompilationQuickJit

The `TieredCompilationQuickJit` property configures whether the JIT compiler uses quick JIT. Set the value to `false` to disable quick JIT. For more information, see [Quick JIT](#).

```
<PropertyGroup>
  <TieredCompilationQuickJit>false</TieredCompilationQuickJit>
</PropertyGroup>
```

TieredCompilationQuickJitForLoops

The `TieredCompilationQuickJitForLoops` property configures whether the JIT compiler uses quick JIT on methods that contain loops. Set the value to `true` to enable quick JIT on methods that contain loops. For more information, see [Quick JIT for loops](#).

```
<PropertyGroup>
  <TieredCompilationQuickJitForLoops>true</TieredCompilationQuickJitForLoops>
</PropertyGroup>
```

Reference properties

The following MSBuild properties are documented in this section:

- [AssetTargetFallback](#)
- [DisableImplicitFrameworkReferences](#)
- [DisableTransitiveProjectReferences](#)
- [Restore-related properties](#)
- [ValidateExecutableReferencesMatchSelfContained](#)

AssetTargetFallback

The `AssetTargetFallback` property lets you specify additional compatible framework versions for project references and NuGet packages. For example, if you specify a package dependency using `PackageReference` but that package doesn't contain assets that are compatible with your project's `TargetFramework`, the `AssetTargetFallback` property comes into play. The compatibility of the referenced package is rechecked using each target framework that's specified in `AssetTargetFallback`. This property replaces the deprecated property `PackageTargetFallback`.

You can set the `AssetTargetFallback` property to one or more [target framework versions](#).

```
<PropertyGroup>
  <AssetTargetFallback>net461</AssetTargetFallback>
</PropertyGroup>
```

DisableImplicitFrameworkReferences

The `DisableImplicitFrameworkReferences` property controls implicit `FrameworkReference` items when targeting .NET Core 3.0 and later versions. When targeting .NET Core 2.1 or .NET Standard 2.0 and earlier versions, it controls implicit `PackageReference` items to packages in a metapackage. (A metapackage is a framework-based package that consists only of dependencies on other packages.) This property also controls implicit references such as `System` and `System.Core` when targeting .NET Framework.

Set this property to `true` to disable implicit `FrameworkReference` or `PackageReference` items. If you set this property to `true`, you can add explicit references to just the frameworks or packages you need.

```
<PropertyGroup>
  <DisableImplicitFrameworkReferences>true</DisableImplicitFrameworkReferences>
</PropertyGroup>
```

DisableTransitiveProjectReferences

The `DisableTransitiveProjectReferences` property controls implicit project references. Set this property to `true` to disable implicit `ProjectReference` items. Disabling implicit project references results in non-transitive behavior similar to the [legacy project system](#).

When this property is `true`, it has a similar effect to that of setting `PrivateAssets="All"` on all of the dependencies of the depended-upon project.

If you set this property to `true`, you can add explicit references to just the projects you need.

```
<PropertyGroup>
  <DisableTransitiveProjectReferences>true</DisableTransitiveProjectReferences>
</PropertyGroup>
```

Restore-related properties

Restoring a referenced package installs all of its direct dependencies and all the dependencies of those dependencies. You can customize package restoration by specifying properties such as `RestorePackagesPath` and `RestoreIgnoreFailedSources`. For more information about these and other properties, see [restore target](#).

```
<PropertyGroup>
  <RestoreIgnoreFailedSource>true</RestoreIgnoreFailedSource>
</PropertyGroup>
```

ValidateExecutableReferencesMatchSelfContained

The `ValidateExecutableReferencesMatchSelfContained` property can be used to disable errors related to executable project references. If .NET detects that a self-contained executable project references a framework-dependent executable project, or vice versa, it generates errors NETSDK1150 and NETSDK1151, respectively. To avoid these errors when the reference is intentional, set the `ValidateExecutableReferencesMatchSelfContained` property to `false`.

```
<PropertyGroup>
  <ValidateExecutableReferencesMatchSelfContained>false</ValidateExecutableReferencesMatchSelfContained>
</PropertyGroup>
```

WindowsSdkPackageVersion

The `WindowsSdkPackageVersion` property can be used to override the version of the [Windows SDK targeting package](#). This property was introduced in .NET 5, and replaces the use of the `FrameworkReference` item for this purpose.

```
<PropertyGroup>
  <WindowsSdkPackageVersion>10.0.19041.18</WindowsSdkPackageVersion>
</PropertyGroup>
```

NOTE

We don't recommend overriding the Windows SDK version, because the Windows SDK targeting packages are included with the .NET 5+ SDK. Instead, to reference the latest Windows SDK package, update your version of the .NET SDK. This property should only be used in rare cases such as using preview packages or needing to override the version of C#/WinRT.

Run-related properties

The following properties are used for launching an app with the `dotnet run` command:

- [RunArguments](#)
- [RunWorkingDirectory](#)

RunArguments

The `RunArguments` property defines the arguments that are passed to the app when it is run.

```
<PropertyGroup>
  <RunArguments>-mode dryrun</RunArguments>
</PropertyGroup>
```

TIP

You can specify additional arguments to be passed to the app by using the `--` option for `dotnet run`.

RunWorkingDirectory

The `RunWorkingDirectory` property defines the working directory for the application process to be started in. It can be an absolute path or a path that's relative to the project directory. If you don't specify a directory, `OutDir` is used as the working directory.

```
<PropertyGroup>
  <RunWorkingDirectory>c:\temp</RunWorkingDirectory>
</PropertyGroup>
```

Hosting-related properties

The following MSBuild properties are documented in this section:

- [EnableComHosting](#)
- [EnableDynamicLoading](#)

EnableComHosting

The `EnableComHosting` property indicates that an assembly provides a COM server. Setting the `EnableComHosting` to `true` also implies that `EnableDynamicLoading` is `true`.

```
<PropertyGroup>
  <EnableComHosting>True</EnableComHosting>
</PropertyGroup>
```

For more information, see [Expose .NET components to COM](#).

EnableDynamicLoading

The `EnableDynamicLoading` property indicates that an assembly is a dynamically loaded component. The component could be a [COM library](#) or a non-COM library that can be [used from a native host](#) or [used as a plugin](#). Setting this property to `true` has the following effects:

- A `.runtimeconfig.json` file is generated.
- `RollForward` is set to `LatestMinor`.
- NuGet references are copied locally.

```
<PropertyGroup>
  <EnableDynamicLoading>true</EnableDynamicLoading>
</PropertyGroup>
```

Generated file properties

The following properties concern code in generated files:

- [DisableImplicitNamespaceImports](#)
- [ImplicitUsings](#)

DisableImplicitNamespaceImports

The `DisableImplicitNamespaceImports` property can be used to disable implicit namespace imports in Visual Basic projects that target .NET 6 or a later version. Implicit namespaces are the default namespaces that are imported globally in a Visual Basic project. Set this property to `true` to disable implicit namespace imports.

```
<PropertyGroup>
  <DisableImplicitNamespaceImports>true</DisableImplicitNamespaceImports>
</PropertyGroup>
```

ImplicitUsings

The `ImplicitUsings` property can be used to enable and disable implicit `global using` directives in C# projects that target .NET 6 or a later version and C# 10 or a later version. When the feature is enabled, the .NET SDK adds `global using` directives for a set of default namespaces based on the type of project SDK. Set this property to `true` or `enable` to enable implicit `global using` directives. To disable implicit `global using` directives, remove the property or set it to `false` or `disable`.

```
<PropertyGroup>
  <ImplicitUsings>enable</ImplicitUsings>
</PropertyGroup>
```

NOTE

The templates for new C# projects that target .NET 6 or later have `ImplicitUsings` set to `enable` by default.

To define an explicit `global using` directive, add a [Using](#) item.

Items

MSBuild items are inputs into the build system. Items are specified according to their type, which is the element name. For example, `Compile` and `Reference` are two [common item types](#). The following additional item types are made available by the .NET SDK:

- [AssemblyMetadata](#)
- [InternalsVisibleTo](#)
- [PackageReference](#)
- [TrimmerRootAssembly](#)
- [Using](#)

You can use any of the standard [item attributes](#), for example, `Include` and `Update`, on these items. Use `Include` to add a new item, and use `Update` to modify an existing item. For example, `Update` is often used to modify an item that has implicitly been added by the .NET SDK.

AssemblyMetadata

The `AssemblyMetadata` item specifies a key-value pair `AssemblyMetadataAttribute` assembly attribute. The `Include` metadata becomes the key, and the `Value` metadata becomes the value.

```
<ItemGroup>
  <AssemblyMetadata Include="Serviceable" Value="True" />
</ItemGroup>
```

InternalsVisibleTo

The `InternalsVisibleTo` item generates an `InternalsVisibleToAttribute` assembly attribute for the specified friend assembly.

```
<ItemGroup>
  <InternalsVisibleTo Include="MyProject.Tests" />
</ItemGroup>
```

If the friend assembly is signed, you can specify an optional `Key` metadata to specify its full public key. If you don't specify `Key` metadata and a `$(PublicKey)` is available, that key is used. Otherwise, no public key is added to the attribute.

PackageReference

The `PackageReference` item defines a reference to a NuGet package.

The `Include` attribute specifies the package ID. The `Version` attribute specifies the version or version range. For information about how to specify a minimum version, maximum version, range, or exact match, see [Version ranges](#).

The project file snippet in the following example references the `System.Runtime` package.

```
<ItemGroup>
  <PackageReference Include="System.Runtime" Version="4.3.0" />
</ItemGroup>
```

You can also [control dependency assets](#) using metadata such as `PrivateAssets`.

```
<ItemGroup>
  <PackageReference Include="Contoso.Utility.UsefulStuff" Version="3.6.0">
    <PrivateAssets>all</PrivateAssets>
  </PackageReference>
</ItemGroup>
```

For more information, see [Package references in project files](#).

TrimmerRootAssembly

The `TrimmerRootAssembly` item lets you exclude an assembly from [trimming](#). Trimming is the process of removing unused parts of the runtime from a packaged application. In some cases, trimming might incorrectly remove required references.

The following XML excludes the `System.Security` assembly from trimming.

```
<ItemGroup>
  <TrimmerRootAssembly Include="System.Security" />
</ItemGroup>
```

For more information, see [Trimming options](#).

Using

The `Using` item lets you [globally include a namespace](#) across your C# project, such that you don't have to add a `using` directive for the namespace at the top of your source files. This item is similar to the `Import` item that can be used for the same purpose in Visual Basic projects. This property is available starting in .NET 6.

```
<ItemGroup>
  <Using Include="My.Awesome.Namespace" />
</ItemGroup>
```

You can also use the `Using` item to define global `using <alias>` and `using static <type>` directives.

```
<ItemGroup>
  <Using Include="My.Awesome.Namespace" Alias="Awesome" />
</ItemGroup>
```

For example:

- `<Using Include="Microsoft.AspNetCore.Http.Results" Alias="Results" />` emits
`global using Results = global::Microsoft.AspNetCore.Http.Results;`
- `<Using Include="Microsoft.AspNetCore.Http.Results" Static="True" />` emits
`global using static global::Microsoft.AspNetCore.Http.Results;`

For more information about aliased `using` directives and `using static <type>` directives, see [using directive](#).

Item metadata

In addition to the standard [MSBuild item attributes](#), the following item metadata tags are made available by the .NET SDK:

- [CopyToPublishDirectory](#)
- [LinkBase](#)

[CopyToPublishDirectory](#)

The `CopyToPublishDirectory` metadata on an MSBuild item controls when the item is copied to the publish directory. Allowable values are `PreserveNewest`, which only copies the item if it has changed, `Always`, which always copies the item, and `Never`, which never copies the item. From a performance standpoint, `PreserveNewest` is preferable because it enables an incremental build.

```
<ItemGroup>
  <None Update="appsettings.Development.json" CopyToOutputDirectory="PreserveNewest"
        CopyToPublishDirectory="PreserveNewest" />
</ItemGroup>
```

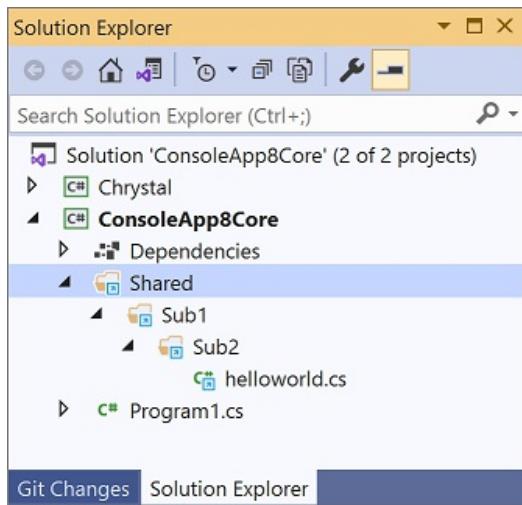
[LinkBase](#)

For an item that's outside of the project directory and its subdirectories, the publish target uses the item's [Link metadata](#) to determine where to copy the item to. `Link` also determines how items outside of the project tree are displayed in the Solution Explorer window of Visual Studio.

If `Link` is not specified for an item that's outside of the project cone, it defaults to `%({LinkBase})%({RecursiveDir})%({Filename})%({Extension})`. `LinkBase` lets you specify a sensible base folder for items outside of the project cone. The folder hierarchy under the base folder is preserved via `RecursiveDir`. If `LinkBase` is not specified, it's omitted from the `Link` path.

```
<ItemGroup>
  <Content Include="..\Extras\**\*.cs" LinkBase="Shared"/>
</ItemGroup>
```

The following image shows how a file that's included via the previous item `Include` glob displays in Solution Explorer.



See also

- [MSBuild schema reference](#)
- [Common MSBuild properties](#)
- [MSBuild properties for NuGet pack](#)
- [MSBuild properties for NuGet restore](#)
- [Customize a build](#)

MSBuild reference for .NET Desktop SDK projects

9/20/2022 • 6 minutes to read • [Edit Online](#)

This page is a reference for the MSBuild properties and items that you use to configure Windows Forms (WinForms) and Windows Presentation Foundation (WPF) projects with the .NET Desktop SDK.

NOTE

This article documents a subset of the MSBuild properties for the .NET SDK as it relates to desktop apps. For a list of common .NET SDK-specific MSBuild properties, see [MSBuild reference for .NET SDK projects](#). For a list of common MSBuild properties, see [Common MSBuild properties](#).

Enable .NET Desktop SDK

To use WinForms or WPF, configure your project file.

.NET 5 and later versions

Specify the following settings in the project file of your WinForms or WPF project:

- Target the .NET SDK `Microsoft.NET.Sdk`. For more information, see [Project files](#).
- Set `TargetFramework` to a Windows-specific target framework moniker, such as `net6.0-windows`.
- Add a UI framework property (or both, if necessary):
 - Set `UseWPF` to `true` to import and use WPF.
 - Set `UseWindowsForms` to `true` to import and use WinForms.
- (Optional) Set `OutputType` to `WinExe`. This produces an app as opposed to a library. To produce a library, omit this property.

```
<Project Sdk="Microsoft.NET.Sdk">

<PropertyGroup>
  <OutputType>WinExe</OutputType>
  <TargetFramework>net6.0-windows</TargetFramework>

  <UseWPF>true</UseWPF>
  <!-- and/or -->
  <UseWindowsForms>true</UseWindowsForms>
</PropertyGroup>

</Project>
```

.NET Core 3.1

Specify the following settings in the project file of your WinForms or WPF project:

- Target the .NET SDK `Microsoft.NET.Sdk.WindowsDesktop`. For more information, see [Project files](#).
- Set `TargetFramework` to `netcoreapp3.1`.
- Add a UI framework property (or both, if necessary):
 - Set `UseWPF` to `true` to import and use WPF.
 - Set `UseWindowsForms` to `true` to import and use WinForms.
- (Optional) Set `OutputType` to `WinExe`. This produces an app as opposed to a library. To produce a library, omit this property.

```

<Project Sdk="Microsoft.NET.Sdk.WindowsDesktop">

  <PropertyGroup>
    <OutputType>WinExe</OutputType>
    <TargetFramework>netcoreapp3.1</TargetFramework>

    <UseWPF>true</UseWPF>
    <!-- and/or -->
    <UseWindowsForms>true</UseWindowsForms>
  </PropertyGroup>

</Project>

```

WPF default includes and excludes

SDK projects define a set of rules to implicitly include or exclude files from the project. These rules also automatically set the file's build action. This is unlike the older non-SDK .NET Framework projects, which have no default include or exclude rules. .NET Framework projects require you to explicitly declare which files to include in the project.

.NET project files include a [standard set of rules](#) for automatically processing files. WPF projects add additional rules.

The following table shows which elements and [glob](#)s are included and excluded in the .NET Desktop SDK when the `UseWPF` project property is set to `true`:

ELEMENT	INCLUDE GLOB	EXCLUDE GLOB	REMOVE GLOB
ApplicationDefinition	App.xaml or Application.xaml	N/A	N/A
Page	**/*.xaml	**/*.user; **/*.proj; **/*.sln; **/*.vsscc Any XAML defined by <i>ApplicationDefinition</i>	N/A
None	N/A	N/A	**/*.xaml

Here are the default include and exclude settings for all project types. For more information, see [Default includes and excludes](#).

ELEMENT	INCLUDE GLOB	EXCLUDE GLOB	REMOVE GLOB
Compile	**/*.cs; **/*.vb (or other language extensions)	**/*.user; **/*.proj; **/*.sln; **/*.vsscc	N/A
EmbeddedResource	**/*.resx	**/*.user; **/*.proj; **/*.sln; **/*.vsscc	N/A
None	**/*	**/*.user; **/*.proj; **/*.sln; **/*.vsscc	**/*.cs; **/*.resx

Errors related to "duplicate" items

If you explicitly added files to your project, or have XAML globs to automatically include files in your project, you might get one of the following errors:

- Duplicate 'ApplicationDefinition' items were included.
- Duplicate 'Page' items were included.

These errors are a result of the implicit `/Include` globs conflicting with your settings. To work around this problem, set either `EnableDefaultApplicationDefinition` or `EnableDefaultPageItems` to `false`. Setting these values to `false` reverts to the behavior of previous SDKs where you had to explicitly define the default globs in your project, or explicitly define the files to include in the project.

You can completely disable all implicit includes by setting the `EnableDefaultItems` property to `false`.

WPF settings

- [UseWPF](#)
- [EnableDefaultApplicationDefinition](#)
- [EnableDefaultPageItems](#)

For information about non-WPF-specific project settings, see [MSBuild reference for .NET SDK projects](#).

UseWPF

The `useWPF` property controls whether or not to include references to WPF libraries. This also alters the MSBuild pipeline to correctly process a WPF project and related files. The default value is `false`. Set the `useWPF` property to `true` to enable WPF support. You can only target the Windows platform when this property is enabled.

```
<PropertyGroup>
  <UseWPF>true</UseWPF>
</PropertyGroup>
```

When this property is set to `true`, .NET 5+ projects will automatically import the [.NET Desktop SDK](#).

.NET Core 3.1 projects need to explicitly target the [.NET Desktop SDK](#) to use this property.

EnableDefaultApplicationDefinition

The `EnableDefaultApplicationDefinition` property controls whether `ApplicationDefinition` items are implicitly included in the project. The default value is `true`. Set the `EnableDefaultApplicationDefinition` property to `false` to disable the implicit file inclusion.

```
<PropertyGroup>
  <EnableDefaultApplicationDefinition>false</EnableDefaultApplicationDefinition>
</PropertyGroup>
```

This property requires that the `EnableDefaultItems` property is set to `true`, which is the default setting.

EnableDefaultPageItems

The `EnableDefaultPageItems` property controls whether `Page` items, which are `.xaml` files, are implicitly included in the project. The default value is `true`. Set the `EnableDefaultPageItems` property to `false` to disable the implicit file inclusion.

```
<PropertyGroup>
  <EnableDefaultPageItems>false</EnableDefaultPageItems>
</PropertyGroup>
```

This property requires that the `EnableDefaultItems` property is set to `true`, which is the default

setting.

Windows Forms settings

- [ApplicationDefaultFont](#)
- [ApplicationHighDpiMode](#)
- [ApplicationUseCompatibleTextRendering](#)
- [ApplicationVisualStyles](#)
- [UseWindowsForms](#)

For information about non-WinForms-specific project properties, see [MSBuild reference for .NET SDK projects](#).

ApplicationDefaultFont

The `ApplicationDefaultFont` property specifies custom font information to be applied application-wide. It controls whether or not the source-generated `ApplicationConfiguration.Initialize()` API emits a call to the `Application.SetDefaultFont(Font)` method. The default value is an empty string, and it means the application default font is sourced from the `Control.DefaultFont` property.

A non-empty value must conform to a format equivalent to the output of the `FontConverter.ConvertTo` method invoked with the [invariant culture](#) (that is, list separator= , and decimal separator= .). The format is:

```
name, size[units[, style=style1[, style2, ...]]].
```

```
<PropertyGroup>
  <ApplicationDefaultFont>Calibri, 11pt, style=regular</ApplicationDefaultFont>
</PropertyGroup>
```

This property is supported by .NET 6 and later versions, and Visual Studio 2022 and later versions.

ApplicationHighDpiMode

The `ApplicationHighDpiMode` property specifies the application-wide default for the high DPI mode. It controls the argument of the `Application.SetHighDpiMode(HighDpiMode)` method emitted by the source-generated `ApplicationConfiguration.Initialize()` API. The default value is `SystemAware`.

```
<PropertyGroup>
  <ApplicationHighDpiMode>PerMonitorV2</ApplicationHighDpiMode>
</PropertyGroup>
```

The `ApplicationHighDpiMode` can be set to one of the `HighDpiMode` enum values:

VALUE	DESCRIPTION
<code>DpiUnaware</code>	The application window does not scale for DPI changes and always assumes a scale factor of 100%.
<code>DpiUnawareGdiScaled</code>	Similar to <code>DpiUnaware</code> , but improves the quality of GDI/GDI+ based content.
<code>PerMonitor</code>	The window checks for DPI when it's created and adjusts scale factor when the DPI changes.
<code>PerMonitorV2</code>	Similar to <code>PerMonitor</code> , but enables child window DPI change notification, improved scaling of comctl32 controls, and dialog scaling.

VALUE	DESCRIPTION
SystemAware	Default if not specified. The window queries for the DPI of the primary monitor once and uses this for the application on all monitors.

This property is supported by .NET 6 and later versions.

ApplicationUseCompatibleTextRendering

The `ApplicationUseCompatibleTextRendering` property specifies the application-wide default for the `UseCompatibleTextRendering` property defined on certain controls. It controls the argument of the `Application.SetCompatibleTextRenderingDefault(Boolean)` method emitted by the source-generated `ApplicationConfiguration.Initialize()` API. The default value is `false`.

```
<PropertyGroup>
  <ApplicationUseCompatibleTextRendering>true</ApplicationUseCompatibleTextRendering>
</PropertyGroup>
```

This property is supported by .NET 6 and later versions.

ApplicationVisualStyles

The `ApplicationVisualStyles` property specifies the application-wide default for enabling visual styles. It controls whether or not the source-generated `ApplicationConfiguration.Initialize()` API emits a call to `Application.EnableVisualStyles()`. The default value is `true`.

```
<PropertyGroup>
  <ApplicationVisualStyles>true</ApplicationVisualStyles>
</PropertyGroup>
```

This property is supported by .NET 6 and later versions.

UseWindowsForms

The `useWindowsForms` property controls whether or not your application is built to target Windows Forms. This property alters the MSBuild pipeline to correctly process a Windows Forms project and related files. The default value is `false`. Set the `useWindowsForms` property to `true` to enable Windows Forms support. You can only target the Windows platform when this setting is enabled.

```
<PropertyGroup>
  <UseWindowsForms>true</UseWindowsForms>
</PropertyGroup>
```

When this property is set to `true`, .NET 5+ projects will automatically import the [.NET Desktop SDK](#).

.NET Core 3.1 projects need to explicitly target the [.NET Desktop SDK](#) to use this property.

Shared settings

- [DisableWinExeOutputInference](#)

DisableWinExeOutputInference

Applies to .NET 5 SDK and later.

When an app has the `Exe` value set for the `outputType` property, a console window is created if the app isn't running from a console. This is generally not the desired behavior of a Windows Desktop app. With the `winExe`

value, a console window isn't created. Starting with the .NET 5 SDK, the `Exe` value is automatically transformed to `WinExe`.

The `DisableWinExeOutputInference` property reverts the behavior of treating `Exe` as `WinExe`. Set this value to `true` to restore the behavior of the `OutputType` property value of `Exe`. The default value is `false`.

```
<PropertyGroup>
  <DisableWinExeOutputInference>true</DisableWinExeOutputInference>
</PropertyGroup>
```

See also

- [.NET project SDKs](#)
- [MSBuild reference for .NET SDK projects](#)
- [MSBuild schema reference](#)
- [Common MSBuild properties](#)
- [Customize a build](#)

Target frameworks in SDK-style projects

9/20/2022 • 9 minutes to read • [Edit Online](#)

When you target a framework in an app or library, you're specifying the set of APIs that you'd like to make available to the app or library. You specify the target framework in your project file using a target framework moniker (TFM).

An app or library can target a version of [.NET Standard](#). .NET Standard versions represent standardized sets of APIs across all .NET implementations. For example, a library can target .NET Standard 1.6 and gain access to APIs that function across .NET Core and .NET Framework using the same codebase.

An app or library can also target a specific .NET implementation to gain access to implementation-specific APIs. For example, an app that targets Xamarin.iOS (for example, `xamarin.ios10`) has access to Xamarin-provided iOS API wrappers for iOS 10, or an app that targets Universal Windows Platform (UWP, `uap10.0`) has access to APIs that compile for devices that run Windows 10.

For some target frameworks, such as .NET Framework, the APIs are defined by the assemblies that the framework installs on a system and may include application framework APIs (for example, ASP.NET).

For package-based target frameworks (for example, .NET 5+, .NET Core, and .NET Standard), the APIs are defined by the NuGet packages included in the app or library.

Latest versions

The following table defines the most common target frameworks, how they're referenced, and which version of [.NET Standard](#) they implement. These target framework versions are the latest stable versions. Prerelease versions aren't shown. A target framework moniker (TFM) is a standardized token format for specifying the target framework of a .NET app or library.

TARGET FRAMEWORK	LATEST STABLE VERSION	TARGET FRAMEWORK MONIKER (TFM)	IMPLEMENTED .NET STANDARD VERSION
.NET 6	6	net6.0	2.1
.NET 5	5	net5.0	2.1
.NET Standard	2.1	netstandard2.1	N/A
.NET Core	3.1	netcoreapp3.1	2.1
.NET Framework	4.8	net48	2.0

Supported target frameworks

A target framework is typically referenced by a TFM. The following table shows the target frameworks supported by the .NET SDK and the NuGet client. Equivalents are shown within brackets. For example, `win81` is an equivalent TFM to `netcore451`.

TARGET FRAMEWORK	TFM
.NET 5+ (and .NET Core)	netcoreapp1.0 netcoreapp1.1 netcoreapp2.0 netcoreapp2.1 netcoreapp2.2 netcoreapp3.0 netcoreapp3.1 net5.0* net6.0* net7.0*
.NET Standard	netstandard1.0 netstandard1.1 netstandard1.2 netstandard1.3 netstandard1.4 netstandard1.5 netstandard1.6 netstandard2.0 netstandard2.1
.NET Framework	net11 net20 net35 net40 net403 net45 net451 net452 net46 net461 net462 net47 net471 net472 net48
Windows Store	netcore [netcore45] netcore45 [win] [win8] netcore451 [win81]
.NET Micro Framework	netmf
Silverlight	sl4 sl5
Windows Phone	wp [wp7] wp7 wp75 wp8 wp81 wpa81
Universal Windows Platform	uap [uap10.0] uap10.0 [win10] [netcore50]

* .NET 5 and later TFMs include some operating system-specific variations. For more information, see the following section, [.NET 5+ OS-specific TFMs](#).

.NET 5+ OS-specific TFMs

The `net5.0`, `net6.0`, and `net7.0` TFMs include technologies that work across different platforms. Specifying an *OS-specific TFM* makes APIs that are specific to an operating system available to your app, for example, Windows Forms or iOS bindings. OS-specific TFMs also inherit every API available to their base TFM, for example, the `net6.0` TFM.

.NET 5 introduced the `net5.0-windows` OS-specific TFM, which includes Windows-specific bindings for WinForms, WPF, and UWP APIs. .NET 6 introduces further OS-specific TFMs.

The following table shows the compatibility of the .NET 5+ TFMs.

TFM	COMPATIBLE WITH
net5.0	net1.4 (with NU1701 warning) netcoreapp1..3.1 (warning when WinForms or WPF is referenced) netstandard1..2.1
net5.0-windows	netcoreapp1..3.1 (plus everything else inherited from <code>net5.0</code>)
net6.0	(subsequent version of <code>net5.0</code>)
net6.0-android	<code>xamarin.android</code> (+everything else inherited from <code>net6.0</code>)
net6.0-ios	<code>xamarin.ios</code> (+everything else inherited from <code>net6.0</code>)
net6.0-maccatalyst	<code>xamarin.ios</code> (+everything else inherited from <code>net6.0</code>)
net6.0-macos	<code>xamarin.mac</code> (+everything else inherited from <code>net6.0</code>)
net6.0-tvos	<code>xamarin.tvos</code> (+everything else inherited from <code>net6.0</code>)
net6.0-windows	(subsequent version of <code>net5.0-windows</code>)
net7.0	(subsequent version of <code>net6.0</code>)
net7.0-android	(subsequent version of <code>net6.0-android</code>)
net7.0-ios	(subsequent version of <code>net6.0-ios</code>)
net7.0-maccatalyst	(subsequent version of <code>net6.0-maccatalyst</code>)
net7.0-macos	(subsequent version of <code>net6.0-macos</code>)
net7.0-tvos	(subsequent version of <code>net6.0-tvos</code>)
net7.0-windows	(subsequent version of <code>net6.0-windows</code>)

To make your app portable across different platforms but still have access to OS-specific APIs, you can target multiple OS-specific TFMs and add platform guards around OS-specific API calls using `#if` preprocessor directives.

Suggested targets

Use these guidelines to determine which TFM to use in your app:

- Apps that are portable to multiple platforms should target a base TFM, for example, `net6.0`. This includes most libraries but also ASP.NET Core and Entity Framework.
- Platform-specific libraries should target platform-specific flavors. For example, WinForms and WPF projects should target `net6.0-windows`.
- Cross-platform application models (Xamarin Forms, ASP.NET Core) and bridge packs (Xamarin Essentials) should at least target the base TFM, for example, `net6.0`, but might also target additional platform-specific flavors to light-up more APIs or features.

OS version in TFMs

You can also specify an optional OS version at the end of an OS-specific TFM, for example, `net6.0-ios15.0`. The version indicates which APIs are available to your app or library. It does not control the OS version that your app or library supports at run time. It's used to select the reference assemblies that your project compiles against, and to select assets from NuGet packages. Think of this version as the "platform version" or "OS API version" to disambiguate it from the run-time OS version.

When an OS-specific TFM doesn't specify the platform version explicitly, it has an implied value that can be inferred from the base TFM and platform name. For example, the default platform value for iOS in .NET 6 is `15.0`, which means that `net6.0-ios` is shorthand for the canonical `net6.0-ios15.0` TFM. The implied platform version for a newer base TFM may be higher, for example, a future `net8.0-ios` TFM could map to `net8.0-ios16.0`. The shorthand form is intended for use in project files only, and is expanded to the canonical form by the .NET SDK's MSBuild targets before being passed to other tools, such as NuGet.

The .NET SDK is designed to be able to support newly released APIs for an individual platform without a new version of the base TFM. This enables you to access platform-specific functionality without waiting for a major release of .NET. You can gain access to these newly released APIs by incrementing the platform version in the TFM. For example, if the iOS platform added iOS 15.1 APIs in a .NET 6.0.x SDK update, you could access them by using the TFM `net6.0-ios15.1`.

Support older OS versions

Although a platform-specific app or library is compiled against APIs from a specific version of that OS, you can make it compatible with earlier OS versions by adding the `SupportedOSPlatformVersion` property to your project file. The `SupportedOSPlatformVersion` property indicates the minimum OS version required to run your app or library. If you don't explicitly specify this minimum run-time OS version in the project, it defaults to the platform version from the TFM.

For your app to run correctly on an older OS version, it can't call APIs that don't exist on that version of the OS. However, you can add guards around calls to newer APIs so they are only called when running on a version of the OS that supports them. This pattern allows you to design your app or library to support running on older OS versions while taking advantage of newer OS functionality when running on newer OS versions.

The `SupportedOSPlatformVersion` value (whether explicit or default) is used by the [platform compatibility analyzer](#), which detects and warns about unguarded calls to newer APIs. It's burned into the project's compiled assembly as an `UnsupportedOSPlatformAttribute` assembly attribute, so that the platform compatibility analyzer can detect unguarded calls to that assembly's APIs from projects with a lower `SupportedOSPlatformVersion` value. On some platforms, the `SupportedOSPlatformVersion` value affects platform-specific app packaging and build processes, which is covered in the documentation for those platforms.

Here is an example excerpt of a project file that uses the `TargetFramework` and `SupportedOSPlatformVersion` MSBuild properties to specify that the app or library has access to iOS 15.0 APIs but supports running on iOS 13.0 and above:

```
<Project Sdk="Microsoft.NET.Sdk">

<PropertyGroup>
  <TargetFramework>net6.0-ios15.0</TargetFramework>
  <SupportedOSPlatformVersion>13.0</SupportedOSPlatformVersion>
</PropertyGroup>

...

</Project>
```

How to specify a target framework

Target frameworks are specified in a project file. When a single target framework is specified, use the [TargetFramework element](#). The following console app project file demonstrates how to target .NET 6:

```
<Project Sdk="Microsoft.NET.Sdk">

<PropertyGroup>
  <OutputType>Exe</OutputType>
  <TargetFramework>net6.0</TargetFramework>
</PropertyGroup>

</Project>
```

When you specify multiple target frameworks, you may conditionally reference assemblies for each target framework. In your code, you can conditionally compile against those assemblies by using preprocessor symbols with *if-then-else* logic.

The following library project targets APIs of .NET Standard (`netstandard1.4`) and .NET Framework (`net40` and `net45`). Use the plural [TargetFrameworks element](#) with multiple target frameworks. The `Condition` attributes include implementation-specific packages when the library is compiled for the two .NET Framework TFM's:

```
<Project Sdk="Microsoft.NET.Sdk">

<PropertyGroup>
  <TargetFrameworks>netstandard1.4;net40;net45</TargetFrameworks>
</PropertyGroup>

<!-- Conditionally obtain references for the .NET Framework 4.0 target -->
<ItemGroup Condition=" '$(TargetFramework)' == 'net40' ">
  <Reference Include="System.Net" />
</ItemGroup>

<!-- Conditionally obtain references for the .NET Framework 4.5 target -->
<ItemGroup Condition=" '$(TargetFramework)' == 'net45' ">
  <Reference Include="System.Net.Http" />
  <Reference Include="System.Threading.Tasks" />
</ItemGroup>

</Project>
```

Within your library or app, you write conditional code using [preprocessor directives](#) to compile for each target framework:

```

public class MyClass
{
    static void Main()
    {
#if NET40
        Console.WriteLine("Target framework: .NET Framework 4.0");
#elif NET45
        Console.WriteLine("Target framework: .NET Framework 4.5");
#else
        Console.WriteLine("Target framework: .NET Standard 1.4");
#endif
    }
}

```

The build system is aware of preprocessor symbols representing the target frameworks shown in the [Supported target framework versions](#) table when you're using SDK-style projects. When using a symbol that represents a .NET Standard, .NET Core, or .NET 5+ TFM, replace dots and hyphens with an underscore, and change lowercase letters to uppercase (for example, the symbol for `netstandard1.4` is `NETSTANDARD1_4`). You can disable generation of these symbols via the `DisableImplicitFrameworkDefines` property. For more information about this property, see [DisableImplicitFrameworkDefines](#).

The complete list of preprocessor symbols for .NET target frameworks is:

TARGET FRAMEWORKS	SYMBOLS	ADDITIONAL SYMBOLS AVAILABLE IN .NET 5+ SDK
.NET Framework	NETFRAMEWORK , NET48 , NET472 , NET471 , NET47 , NET462 , NET461 , NET46 , NET452 , NET451 , NET45 , NET40 , NET35 , NET20	NET48_OR_GREATER , NET472_OR_GREATER , NET471_OR_GREATER , NET47_OR_GREATER , NET462_OR_GREATER , NET461_OR_GREATER , NET46_OR_GREATER , NET452_OR_GREATER , NET451_OR_GREATER , NET45_OR_GREATER , NET40_OR_GREATER , NET35_OR_GREATER , NET20_OR_GREATER
.NET Standard	NETSTANDARD , NETSTANDARD2_1 , NETSTANDARD2_0 , NETSTANDARD1_6 , NETSTANDARD1_5 , NETSTANDARD1_4 , NETSTANDARD1_3 , NETSTANDARD1_2 , NETSTANDARD1_1 , NETSTANDARD1_0	NETSTANDARD2_1_OR_GREATER , NETSTANDARD2_0_OR_GREATER , NETSTANDARD1_6_OR_GREATER , NETSTANDARD1_5_OR_GREATER , NETSTANDARD1_4_OR_GREATER , NETSTANDARD1_3_OR_GREATER , NETSTANDARD1_2_OR_GREATER , NETSTANDARD1_1_OR_GREATER , NETSTANDARD1_0_OR_GREATER

TARGET FRAMEWORKS	SYMBOLS	ADDITIONAL SYMBOLS AVAILABLE IN .NET 5+ SDK
.NET 5+ (and .NET Core)	NET , NET7_0 , NET6_0 , NET5_0 , NETCOREAPP , NETCOREAPP3_1 , NETCOREAPP3_0 , NETCOREAPP2_2 , NETCOREAPP2_1 , NETCOREAPP2_0 , NETCOREAPP1_1 , NETCOREAPP1_0	NET7_0_OR_GREATER , NET6_0_OR_GREATER , NET5_0_OR_GREATER , NETCOREAPP3_1_OR_GREATER , NETCOREAPP3_0_OR_GREATER , NETCOREAPP2_2_OR_GREATER , NETCOREAPP2_1_OR_GREATER , NETCOREAPP2_0_OR_GREATER , NETCOREAPP1_1_OR_GREATER , NETCOREAPP1_0_OR_GREATER

NOTE

- Versionless symbols are defined regardless of the version you're targeting.
- Version-specific symbols are only defined for the version you're targeting.
- The `<framework>_OR_GREATER` symbols are defined for the version you're targeting and all earlier versions. For example, if you're targeting .NET Framework 2.0, the following symbols are defined: `NET20` , `NET20_OR_GREATER` , `NET11_OR_GREATER` , and `NET10_OR_GREATER` .
- These are different from the target framework monikers (TFMs) used by the MSBuild `TargetFramework` property and NuGet.

Deprecated target frameworks

The following target frameworks are deprecated. Packages that target these target frameworks should migrate to the indicated replacements.

DEPRECATED TFM	REPLACEMENT
aspnet50 aspnetcore50 dnxcore50 dnx dnx45 dnx451 dnx452	netcoreapp
dotnet dotnet50 dotnet51 dotnet52 dotnet53 dotnet54 dotnet55 dotnet56	netstandard
netcore50	uap10.0
win	netcore45
win8	netcore45
win81	netcore451

DEPRECATED TFM	REPLACEMENT
win10	uap10.0
winrt	netcore45

See also

- [Target framework names in .NET 5](#)
- [Call Windows Runtime APIs in desktop apps](#)
- [Developing Libraries with Cross Platform Tools](#)
- [.NET Standard](#)
- [.NET Core Versioning](#)
- [NuGet Tools GitHub Repository](#)
- [Framework Profiles in .NET](#)
- [Platform compatibility analyzer](#)

Manage package dependencies in .NET applications

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article explains how to add and remove package dependencies by editing the project file or by using the CLI.

The `<PackageReference>` element

The `<PackageReference>` project file element has the following structure:

```
<PackageReference Include="PACKAGE_ID" Version="PACKAGE_VERSION" />
```

The `Include` attribute specifies the ID of the package to add to the project. The `Version` attribute specifies the version to get. Versions are specified as per [NuGet version rules](#).

NOTE

If you're not familiar with project-file syntax, see the [MSBuild project reference](#) documentation for more information.

Use conditions to add a dependency that's available only in a specific target, as shown in the following example:

```
<PackageReference Include="PACKAGE_ID" Version="PACKAGE_VERSION" Condition="'$(TargetFramework)' == 'netcoreapp2.1'" />
```

The dependency in the preceding example will only be valid if the build is happening for that given target. The `$(TargetFramework)` in the condition is an MSBuild property that's being set in the project. For most common .NET applications, you don't need to do this.

Add a dependency by editing the project file

To add a dependency, add a `<PackageReference>` element inside an `<ItemGroup>` element. You can add to an existing `<ItemGroup>` or create a new one. The following example uses the default console application project that's created by `dotnet new console`:

```
<Project Sdk="Microsoft.NET.Sdk.Web">

  <PropertyGroup>
    <TargetFramework>netcoreapp3.1</TargetFramework>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.EntityFrameworkCore" Version="3.1.2" />
  </ItemGroup>

</Project>
```

Add a dependency by using the CLI

To add a dependency, run the `dotnet add package` command, as shown in the following example:

```
dotnet add package Microsoft.EntityFrameworkCore
```

Remove a dependency by editing the project file

To remove a dependency, remove its `<PackageReference>` element from the project file.

Remove a dependency by using the CLI

To remove a dependency, run the [dotnet remove package](#) command, as shown in the following example:

```
dotnet remove package Microsoft.EntityFrameworkCore
```

See also

- [Package references in project files](#)
- [dotnet list package command](#)

How to manage .NET tools

9/20/2022 • 8 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 2.1 SDK and later versions.

A .NET tool is a special NuGet package that contains a console application. You can install a tool on your machine in the following ways:

- As a global tool.

The tool binaries are installed in a default directory that's added to the PATH environment variable. You can invoke the tool from any directory on the machine without specifying its location. One version of a tool is used for all directories on the machine.

- As a global tool in a custom location (also known as a tool-path tool).

The tool binaries are installed in a location that you specify. You can invoke the tool from the installation directory, by providing the directory with the command name, or by adding the directory to the PATH environment variable. One version of a tool is used for all directories on the machine.

- As a local tool (applies to .NET Core SDK 3.0 and later versions).

The tool binaries are installed in a default directory. You can invoke the tool from the installation directory or any of its subdirectories. Different directories can use different versions of the same tool.

The .NET CLI uses manifest files to keep track of tools that are installed as local to a directory. When the manifest file is saved in the root directory of a source code repository, a contributor can clone the repository and invoke a single .NET CLI command to install all of the tools listed in the manifest files.

IMPORTANT

.NET tools run in full trust. Don't install a .NET tool unless you trust the author.

Find a tool

Here are some ways to find tools:

- Use the [dotnet tool search](#) command to find a tool that's published to NuGet.org.
- Use the ".NET tool" package type filter to search for the [NuGet](#) website. For more information, see [Finding and choosing packages](#).
- See the source code for the tools the ASP.NET Core team created in the [Tools directory of the dotnet/aspnetcore GitHub repository](#).
- Learn about diagnostic tools at [.NET diagnostic tools](#).

Check the author and statistics

.NET tools can be powerful because they run in full trust, and global tools are added to the PATH environment variable. Don't download tools from people you don't trust.

If the tool is hosted on NuGet, you can check the author and statistics by searching for the tool.

Install a global tool

To install a tool as a global tool, use the `-g` or `--global` option of [dotnet tool install](#), as shown in the following example:

```
dotnet tool install -g dotnetsay
```

The output shows the command used to invoke the tool and the version installed, similar to the following example:

```
You can invoke the tool using the following command: dotnetsay
Tool 'dotnetsay' (version '2.1.4') was successfully installed.
```

The default location for a tool's binaries depends on the operating system:

OS	PATH
Linux/macOS	<code>\$HOME/.dotnet/tools</code>
Windows	<code>%USERPROFILE%\.dotnet\tools</code>

This location is added to the user's path when the SDK is first run. So global tools can be invoked from any directory without specifying the tool location.

Tool access is user-specific, not machine global. A global tool is only available to the user that installed the tool.

Install a global tool in a custom location

To install a tool as a global tool in a custom location, use the `--tool-path` option of [dotnet tool install](#), as shown in the following examples:

On Windows:

```
dotnet tool install dotnetsay --tool-path c:\dotnet-tools
```

On Linux or macOS:

```
dotnet tool install dotnetsay --tool-path ~/bin
```

The .NET SDK doesn't add this location automatically to the PATH environment variable. To [invoke a tool-path tool](#), you must ensure that the command is available by using one of the following methods:

- Add the installation directory to the PATH environment variable.
- Specify the full path to the tool when you invoke it.
- Invoke the tool from within the installation directory.

Install a local tool

Applies to .NET Core 3.0 SDK and later.

If you want to install a tool for local access only (for the current directory and subdirectories), you must add the tool to a tool manifest file. To create a tool manifest file, run the `dotnet new tool-manifest` command:

```
dotnet new tool-manifest
```

This command creates a manifest file named `dotnet-tools.json` under the `.config` directory. To add a local tool to the manifest file, use the `dotnet tool install` command and omit the `--global` and `--tool-path` options, as shown in the following example:

```
dotnet tool install dotnetsay
```

The command output shows in which manifest file the newly installed tool is present, similar to the following example:

```
You can invoke the tool from this directory using the following command:  
dotnet tool run dotnetsay  
Tool 'dotnetsay' (version '2.1.4') was successfully installed.  
Entry is added to the manifest file /home/name/botsay/.config/dotnet-tools.json.
```

The following example shows a manifest file with two local tools installed:

```
{  
  "version": 1,  
  "isRoot": true,  
  "tools": {  
    "botsay": {  
      "version": "1.0.0",  
      "commands": [  
        "botsay"  
      ]  
    },  
    "dotnetsay": {  
      "version": "2.1.3",  
      "commands": [  
        "dotnetsay"  
      ]  
    }  
  }  
}
```

You typically add a local tool to the root directory of the repository. After you check in the manifest file to the repository, developers who check out code from the repository get the latest manifest file. To install all of the tools listed in the manifest file, they run the `dotnet tool restore` command:

```
dotnet tool restore
```

The output indicates the restored tools:

```
Tool 'botsay' (version '1.0.0') was restored. Available commands: botsay  
Tool 'dotnetsay' (version '2.1.3') was restored. Available commands: dotnetsay  
Restore was successful.
```

Install a specific tool version

To install a pre-release version or a specific version of a tool, specify the version number by using the `--version` option, as shown in the following example:

```
dotnet tool install dotnetsay --version 2.1.3
```

To install a pre-release version of the tool without specifying the exact version number, use the `--version` option and provide a wildcard, as shown in the following example:

```
dotnet tool install --global dotnetsay --version "*-rc*"
```

Use a tool

The command that you use to invoke a tool might be different from the name of the package that you install. To display all of the tools currently installed on the machine for the current user, use the [dotnet tool list](#) command:

```
dotnet tool list
```

The output shows each tool's version and command, similar to the following example:

Package Id	Version	Commands	Manifest
botsay	1.0.0	botsay	/home/name/repository/.config/dotnet-tools.json
dotnetsay	2.1.3	dotnetsay	/home/name/repository/.config/dotnet-tools.json

As shown in the preceding example, the list shows local tools. To see global tools, use the `--global` option. To see tool-path tools, use the `--tool-path` option.

Invoke a global tool

For global tools, use the tool command by itself. For example, if the command is `dotnetsay` or `dotnet-doc`, that's what you use to invoke the global tool:

```
dotnetsay  
dotnet-doc
```

If the command begins with the prefix `dotnet-`, an alternative way to invoke the tool is to use the `dotnet` command and omit the tool command prefix. For example, if the command is `dotnet-doc`, the following command invokes the tool:

```
dotnet doc
```

However, in the following scenario you can't use the `dotnet` command to invoke a global tool:

- A global tool and a local tool have the same command prefixed by `dotnet-`.
- You want to invoke the global tool from a directory that's in scope for the local tool.

In this scenario, `dotnet doc` and `dotnet dotnet-doc` invoke the local tool. To invoke the global tool, use the command by itself:

```
dotnet-doc
```

Invoke a tool-path tool

To invoke a global tool that's installed by using the `tool-path` option, ensure that the command is available as explained [earlier in this article](#).

Invoke a local tool

To invoke a local tool, you must use the `dotnet` command from within the installation directory. You can use the

long form (`dotnet tool run <COMMAND_NAME>`) or the short form (`dotnet <COMMAND_NAME>`), as shown in the following examples:

```
dotnet tool run dotnetsay  
dotnet dotnetsay
```

If the command is prefixed by `dotnet-`, you can include or omit the prefix when you invoke the tool. For example, if the command is `dotnet-doc`, any of the following examples invokes the local tool:

```
dotnet tool run dotnet-doc  
dotnet dotnet-doc  
dotnet doc
```

Update a tool

Updating a tool involves uninstalling and reinstalling it with the latest stable version. To update a tool, use the [dotnet tool update](#) command with the same option that you used to install the tool:

```
dotnet tool update --global <packagename>  
dotnet tool update --tool-path <packagename>  
dotnet tool update <packagename>
```

For a local tool, the SDK looks in the current directory and parent directories to find the first manifest file containing the package ID. If there's no such package ID in any manifest file, the SDK adds a new entry to the closest manifest file.

Uninstall a tool

Uninstall a tool by using the [dotnet tool uninstall](#) command with the same option that you used to install the tool:

```
dotnet tool uninstall --global <packagename>  
dotnet tool uninstall --tool-path <packagename>  
dotnet tool uninstall <packagename>
```

For a local tool, the SDK looks in the current directory and parent directories to find the first manifest file containing the package ID.

Get help and troubleshoot

If a tool fails to install or run, see [Troubleshoot .NET tool usage issues](#). You can get a list of available `dotnet tool` commands and parameters by using the `--help` parameter:

```
dotnet tool --help
```

To get tool usage instructions, enter one of the following commands or see the tool's website:

```
<command> --help  
dotnet <command> --help
```

See also

- [Tutorial: Create a .NET tool using the .NET CLI](#)
- [Tutorial: Install and use a .NET global tool using the .NET CLI](#)
- [Tutorial: Install and use a .NET local tool using the .NET CLI](#)

Troubleshoot .NET tool usage issues

9/20/2022 • 8 minutes to read • [Edit Online](#)

You might come across issues when trying to install or run a .NET tool, which can be a global tool or a local tool. This article describes the common root causes and some possible solutions.

Installed .NET tool fails to run

When a .NET tool fails to run, most likely you ran into one of the following issues:

- [The executable file for the tool wasn't found.](#)
- [The correct version of the .NET runtime wasn't found.](#)

Executable file not found

If the executable file isn't found, you'll see a message similar to the following:

```
Could not execute because the specified command or file was not found.  
Possible reasons for this include:  
* You misspelled a built-in dotnet command.  
* You intended to execute a .NET program, but dotnet-xyz does not exist.  
* You intended to run a global tool, but a dotnet-prefixed executable with this name could not be found on  
the PATH.
```

The name of the executable determines how you invoke the tool. The following table describes the format:

EXECUTABLE NAME FORMAT	INVOCATION FORMAT
<code>dotnet-<toolName>.exe</code>	<code>dotnet <toolName></code>
<code><toolName>.exe</code>	<code><toolName></code>

Global tools

Global tools can be installed in the default directory or in a specific location. The default directories are:

OS	PATH
Linux/macOS	<code>\$HOME/.dotnet/tools</code>
Windows	<code>%USERPROFILE%\.dotnet\tools</code>

If you're trying to run a global tool, check that the `PATH` environment variable on your machine contains the path where you installed the global tool and that the executable is in that path.

The .NET CLI tries to add the default location to the `PATH` environment variable on its first usage. However, there are some scenarios where the location might not be added to `PATH` automatically:

- If you're using Linux and you've installed the .NET SDK using `.tar.gz` files and not `apt-get` or `rpm`.
- If you're using macOS 10.15 "Catalina" or later versions.
- If you're using macOS 10.14 "Mojave" or earlier versions, and you've installed the .NET SDK using `.tar.gz` files and not `.pkg`.
- If you've installed the .NET Core 3.0 SDK and you've set the `DOTNET_ADD_GLOBAL_TOOLS_TO_PATH` environment

variable to `false`.

- If you've installed .NET Core 2.2 SDK or earlier versions, and you've set the `DOTNET_SKIP_FIRST_TIME_EXPERIENCE` environment variable to `true`.

In these scenarios or if you specified the `--tool-path` option, the `PATH` environment variable on your machine doesn't automatically contain the path where you installed the global tool. In that case, append the tool location (for example, `$HOME/.dotnet/tools`) to the `PATH` environment variable by using whatever method your shell provides for updating environment variables. For more information, see [.NET tools](#).

Local tools

If you're trying to run a local tool, verify that there's a manifest file called `dotnet-tools.json` in the current directory or any of its parent directories. This file can also live under a folder named `.config` anywhere in the project folder hierarchy, instead of the root folder. If `dotnet-tools.json` exists, open it and check for the tool you're trying to run. If the file doesn't contain an entry for `"isRoot": true`, then also check further up the file hierarchy for additional tool manifest files.

If you're trying to run a .NET tool that was installed with a specified path, you need to include that path when using the tool. An example of using a tool-path installed tool is:

```
..\<toolDirectory>\dotnet-<toolName>
```

Runtime not found

.NET tools are [framework-dependent applications](#), which means they rely on a .NET runtime installed on your machine. If the expected runtime isn't found, they follow normal .NET runtime roll-forward rules such as:

- An application rolls forward to the highest patch release of the specified major and minor version.
- If there's no matching runtime with a matching major and minor version number, the next higher minor version is used.
- Roll forward doesn't occur between preview versions of the runtime or between preview versions and release versions. So, .NET tools created using preview versions must be rebuilt and republished by the author and reinstalled.

Roll-forward won't occur by default in two common scenarios:

- Only lower versions of the runtime are available. Roll-forward only selects later versions of the runtime.
- Only higher major versions of the runtime are available. Roll-forward doesn't cross major version boundaries.

If an application can't find an appropriate runtime, it fails to run and reports an error.

You can find out which .NET runtimes are installed on your machine using one of the following commands:

```
dotnet --list-runtimes  
dotnet --info
```

If you think the tool should support the runtime version you currently have installed, you can contact the tool author and see if they can update the version number or multi-target. Once they've recompiled and republished their tool package to NuGet with an updated version number, you can update your copy. While that doesn't happen, the quickest solution for you is to install a version of the runtime that would work with the tool you're trying to run. To download a specific .NET runtime version, visit the [.NET download page](#).

If you install the .NET SDK to a non-default location, you need to set the environment variable `DOTNET_ROOT` to the directory that contains the `dotnet` executable.

.NET tool installation fails

There are a number of reasons the installation of a .NET global or local tool may fail. When the tool installation fails, you'll see a message similar to the following one:

```
Tool '{0}' failed to install. This failure may have been caused by:  
* You are attempting to install a preview release and did not use the --version option to specify the version.  
* A package by this name was found, but it was not a .NET tool.  
* The required NuGet feed cannot be accessed, perhaps because of an Internet connection problem.  
* You mistyped the name of the tool.  
  
For more reasons, including package naming enforcement, visit https://aka.ms/failure-installing-tool
```

To help diagnose these failures, NuGet messages are shown directly to the user, along with the previous message. The NuGet message may help you identify the problem.

- [Package naming enforcement](#)
- [Preview releases](#)
- [Package isn't a .NET tool](#)
- [NuGet feed can't be accessed](#)
- [Package ID incorrect](#)
- [401 \(Unauthorized\)](#)

Package naming enforcement

Microsoft has changed its guidance on the Package ID for tools, resulting in a number of tools not being found with the predicted name. The new guidance is that all Microsoft tools be prefixed with "Microsoft." This prefix is reserved and can only be used for packages signed with a Microsoft authorized certificate.

During the transition, some Microsoft tools will have the old form of the package ID, while others will have the new form:

```
dotnet tool install -g Microsoft.<toolName>  
dotnet tool install -g <toolName>
```

As package IDs are updated, you'll need to change to the new package ID to get the latest updates. Packages with the simplified tool name will be deprecated.

Preview releases

- You're attempting to install a preview release and didn't use the `--version` option to specify the version.

.NET tools that are in preview must be specified with a portion of the name to indicate that they are in preview. You don't need to include the entire preview. Assuming the version numbers are in the expected format, you can use something like the following example:

```
dotnet tool install -g --version 1.1.0-pre <toolName>
```

Package isn't a .NET tool

- A NuGet package by this name was found, but it wasn't a .NET tool.

If you try to install a NuGet package that is a regular NuGet package and not a .NET tool, you'll see an error similar to the following:

NU1212: Invalid project-package combination for <toolName>. DotnetToolReference project style can only contain references of the DotnetTool type.

NuGet feed can't be accessed

- The required NuGet feed can't be accessed, perhaps because of an Internet connection problem.

Tool installation requires access to the NuGet feed that contains the tool package. It fails if the feed isn't available. You can alter feeds with `nuget.config`, request a specific `nuget.config` file, or specify additional feeds with the `--add-source` switch. By default, NuGet throws an error for any feed that can't connect. The flag `--ignore-failed-sources` can skip these non-reachable sources.

Package ID incorrect

- You mistyped the name of the tool.

A common reason for failure is that the tool name isn't correct. This can happen because of mistyping, or because the tool has moved or been deprecated. For tools on NuGet.org, one way to ensure you have the name correct is to search for the tool at NuGet.org and copy the installation command.

401 (Unauthorized)

Most likely you've specified an alternative NuGet feed, and that feed requires authentication. There are a few different ways to solve this:

- Add the `--ignore-failed-sources` parameter to bypass the error from the private feed and use the public Microsoft feed.

If you're installing a tool from the Microsoft NuGet feed, your custom feed is returning this error before Microsoft's NuGet feed returns a result. The error terminates the request, canceling any other pending feed requests, which could be Microsoft's NuGet feed. Adding the `--ignore-failed-sources` option causes the command to treat this error as a warning and allows other feeds to process the request.

```
dotnet tool install -g --ignore-failed-sources <toolName>
```

- Force the Microsoft NuGet feed with the `--add-source` parameter.

It's possible that the global or local NuGet config file is missing the public Microsoft NuGet feed. Use a combination of the `--add-source` and `--ignore-failed-sources` parameters to avoid the erroneous feed and rely on the public Microsoft feed.

```
dotnet tool install -g --add-source 'https://api.nuget.org/v3/index.json' --ignore-failed-sources <toolName>
```

- Use a custom NuGet config, `--configfile <FILE>` parameter.

Create a local `nuget.config` file with just the public Microsoft NuGet feed, and reference it with the `--configfile` parameter:

```
dotnet tool install -g --configfile "./nuget.config" <toolName>
```

Here's an example config file:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <packageSources>
    <add key="nuget.org" value="https://api.nuget.org/v3/index.json" protocolVersion="3" />
  </packageSources>
</configuration>
```

For more information, see [nuget.config reference](#)

- Add the required credentials to the config file.

If you know the package exists in the configured feed, provide the login credentials in the NuGet config file. For more information about credentials in a nuget config file, see the [nuget.config reference's packageSourceCredentials section](#).

See also

- [.NET tools](#)

Tutorial: Create a .NET tool using the .NET CLI

9/20/2022 • 4 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 2.1 SDK and later versions

This tutorial teaches you how to create and package a .NET tool. The .NET CLI lets you create a console application as a tool, which others can install and run. .NET tools are NuGet packages that are installed from the .NET CLI. For more information about tools, see [.NET tools overview](#).

The tool that you'll create is a console application that takes a message as input and displays the message along with lines of text that create the image of a robot.

This is the first in a series of three tutorials. In this tutorial, you create and package a tool. In the next two tutorials you [use the tool as a global tool](#) and [use the tool as a local tool](#). The procedures for creating a tool are the same whether you use it as a global tool or as a local tool.

Prerequisites

- [.NET SDK 6.0.100](#) or a later version.

This tutorial uses .NET SDK 6.0, but global tools are available starting in .NET Core SDK 2.1. Local tools are available starting in .NET Core SDK 3.0.

- A text editor or code editor of your choice.

Create a project

1. Open a command prompt and create a folder named *repository*.
2. Navigate to the *repository* folder and enter the following command:

```
dotnet new console -n microsoft.botsay -f net6.0
```

The command creates a new folder named *microsoft.botsay* under the *repository* folder.

NOTE

For this tutorial you create a tool that targets .NET 6.0. To target a different framework, change the `-f|--framework` option. To target multiple frameworks, change the `TargetFramework` element to a `TargetFrameworks` element in the project file, as shown in the following example:

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFrameworks>netcoreapp3.1;net5.0;net6.0</TargetFrameworks>
  </PropertyGroup>
</Project>
```

3. Navigate to the *microsoft.botsay* folder.

```
cd microsoft.botsay
```

Add the code

1. Open the *Program.cs* file with your code editor.
2. Replace the code in *Program.cs* with the following code:

```
using System.Reflection;

namespace TeleprompterConsole;

internal class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("Hello World!");
    }
}
```

3. Replace the `Main` method with the following code to process the command-line arguments for the application.

```
static void Main(string[] args)
{
    if (args.Length == 0)
    {
        var versionString = Assembly.GetEntryAssembly()?
            .GetCustomAttribute<AssemblyInformationalVersionAttribute>()?
            .InformationalVersion
            .ToString();

        Console.WriteLine($"botsay v{versionString}");
        Console.WriteLine("-----");
        Console.WriteLine("\nUsage:");
        Console.WriteLine("  botsay <message>");
        return;
    }

    ShowBot(string.Join(' ', args));
}
```

If no arguments are passed, a short help message is displayed. Otherwise, all of the arguments are concatenated into a single string and printed by calling the `ShowBot` method that you create in the next step.

4. Add a new method named `ShowBot` that takes a string parameter. The method prints out the message and an image of a robot using lines of text.

5. Save your changes.

```
dotnet run  
dotnet run -- "Hello from the bot"  
dotnet run -- Hello from the bot
```

All arguments after the `--` delimiter are passed to your application.

Package the tool

Before you can pack and distribute the application as a tool, you need to modify the project file.

1. Open the `microsoft.botsav.csproj` file and add three new XML nodes to the end of the `<PropertyGroup>`

node:

```
<PackAsTool>true</PackAsTool>
<ToolCommandName>botsay</ToolCommandName>
<PackageOutputPath>./nupkg</PackageOutputPath>
```

`<ToolCommandName>` is an optional element that specifies the command that will invoke the tool after it's installed. If this element isn't provided, the command name for the tool is the project file name without the `.csproj` extension.

`<PackageOutputPath>` is an optional element that determines where the NuGet package will be produced. The NuGet package is what the .NET CLI uses to install your tool.

The project file now looks like the following example:

```
<Project Sdk="Microsoft.NET.Sdk">

<PropertyGroup>

    <OutputType>Exe</OutputType>
    <TargetFramework>net6.0</TargetFramework>

    <PackAsTool>true</PackAsTool>
    <ToolCommandName>botsay</ToolCommandName>
    <PackageOutputPath>./nupkg</PackageOutputPath>

</PropertyGroup>

</Project>
```

2. Create a NuGet package by running the [dotnet pack](#) command:

```
dotnet pack
```

The `microsoft.botsay.1.0.0.nupkg` file is created in the folder identified by the `<PackageOutputPath>` value from the `microsoft.botsay.csproj` file, which in this example is the `./nupkg` folder.

When you want to release a tool publicly, you can upload it to <https://www.nuget.org>. Once the tool is available on NuGet, developers can install the tool by using the [dotnet tool install](#) command. For this tutorial you install the package directly from the local `nupkg` folder, so there's no need to upload the package to NuGet.

Troubleshoot

If you get an error message while following the tutorial, see [Troubleshoot .NET tool usage issues](#).

Next steps

In this tutorial, you created a console application and packaged it as a tool. To learn how to use the tool as a global tool, advance to the next tutorial.

[Install and use a global tool](#)

If you prefer, you can skip the global tools tutorial and go directly to the local tools tutorial.

[Install and use a local tool](#)

Tutorial: Install and use a .NET global tool using the .NET CLI

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 2.1 SDK and later versions

This tutorial teaches you how to install and use a global tool. You use a tool that you create in the [first tutorial of this series](#).

Prerequisites

- Complete the [first tutorial of this series](#).

Use the tool as a global tool

1. Install the tool from the package by running the `dotnet tool install` command in the `microsoft.botsay` project folder:

```
dotnet tool install --global --add-source ./nupkg microsoft.botsay
```

The `--global` parameter tells the .NET CLI to install the tool binaries in a default location that is automatically added to the PATH environment variable.

The `--add-source` parameter tells the .NET CLI to temporarily use the `./nupkg` directory as an additional source feed for NuGet packages. You gave your package a unique name to make sure that it will only be found in the `./nupkg` directory, not on the Nuget.org site.

The output shows the command used to call the tool and the version installed:

```
You can invoke the tool using the following command: botsay
Tool 'microsoft.botsay' (version '1.0.0') was successfully installed.
```

2. Invoke the tool:

```
botsay hello from the bot
```

NOTE

If this command fails, you may need to open a new terminal to refresh the PATH.

3. Remove the tool by running the `dotnet tool uninstall` command:

```
dotnet tool uninstall -g microsoft.botsay
```

Use the tool as a global tool installed in a custom location

1. Install the tool from the package.

On Windows:

```
dotnet tool install --tool-path c:\dotnet-tools --add-source ./nupkg microsoft.botsay
```

On Linux or macOS:

```
dotnet tool install --tool-path ~/bin --add-source ./nupkg microsoft.botsay
```

The `--tool-path` parameter tells the .NET CLI to install the tool binaries in the specified location. If the directory doesn't exist, it is created. This directory is not automatically added to the PATH environment variable.

The output shows the command used to call the tool and the version installed:

```
You can invoke the tool using the following command: botsay
Tool 'microsoft.botsay' (version '1.0.0') was successfully installed.
```

2. Invoke the tool:

On Windows:

```
c:\dotnet-tools\botsay hello from the bot
```

On Linux or macOS:

```
~/bin/botsay hello from the bot
```

3. Remove the tool by running the `dotnet tool uninstall` command:

On Windows:

```
dotnet tool uninstall --tool-path c:\dotnet-tools microsoft.botsay
```

On Linux or macOS:

```
dotnet tool uninstall --tool-path ~/bin microsoft.botsay
```

Troubleshoot

If you get an error message while following the tutorial, see [Troubleshoot .NET tool usage issues](#).

Next steps

In this tutorial, you installed and used a tool as a global tool. For more information about how to install and use global tools, see [Managing global tools](#). To install and use the same tool as a local tool, advance to the next tutorial.

[Install and use local tools](#)

Tutorial: Install and use a .NET local tool using the .NET CLI

9/20/2022 • 4 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 3.0 SDK and later versions

This tutorial teaches you how to install and use a local tool. You use a tool that you create in the [first tutorial of this series](#).

Prerequisites

- Complete the [first tutorial of this series](#).
- Install the .NET Core 2.1 runtime.

For this tutorial you install and use a tool that targets .NET Core 2.1, so you need to have that runtime installed on your machine. To install the 2.1 runtime, go to the [.NET Core 2.1 download page](#) and find the runtime installation link in the **Run apps - Runtime** column.

Create a manifest file

To install a tool for local access only (for the current directory and subdirectories), it has to be added to a manifest file.

From the *microsoft.botsay* folder, navigate up one level to the *repository* folder:

```
cd ..
```

Create a manifest file by running the [dotnet new](#) command:

```
dotnet new tool-manifest
```

The output indicates successful creation of the file.

```
The template "Dotnet local tool manifest file" was created successfully.
```

The *.config/dotnet-tools.json* file has no tools in it yet:

```
{
  "version": 1,
  "isRoot": true,
  "tools": {}
}
```

The tools listed in a manifest file are available to the current directory and subdirectories. The current directory is the one that contains the *.config* directory with the manifest file.

When you use a CLI command that refers to a local tool, the SDK searches for a manifest file in the current directory and parent directories. If it finds a manifest file, but the file doesn't include the referenced tool, it continues the search up through parent directories. The search ends when it finds the referenced tool or it finds

a manifest file with `isRoot` set to `true`.

Install botsay as a local tool

Install the tool from the package that you created in the first tutorial:

```
dotnet tool install --add-source ./microsoft.botsay/nupkg microsoft.botsay
```

This command adds the tool to the manifest file that you created in the preceding step. The command output shows which manifest file the newly installed tool is in:

```
You can invoke the tool from this directory using the following command:  
'dotnet tool run botsay' or 'dotnet botsay'  
Tool 'microsoft.botsay' (version '1.0.0') was successfully installed.  
Entry is added to the manifest file /home/name/repository/.config/dotnet-tools.json
```

The `.config/dotnet-tools.json` file now has one tool:

```
{
  "version": 1,
  "isRoot": true,
  "tools": {
    "microsoft.botsay": {
      "version": "1.0.0",
      "commands": [
        "botsay"
      ]
    }
  }
}
```

Use the tool

Invoke the tool by running the `dotnet tool run` command from the `repository` folder:

```
dotnet tool run botsay hello from the bot
```

Restore a local tool installed by others

You typically install a local tool in the root directory of the repository. After you check in the manifest file to the repository, other developers can get the latest manifest file. To install all of the tools listed in the manifest file, they can run a single `dotnet tool restore` command.

1. Open the `.config/dotnet-tools.json` file, and replace the contents with the following JSON:

```
{  
    "version": 1,  
    "isRoot": true,  
    "tools": {  
        "microsoft.botsay": {  
            "version": "1.0.0",  
            "commands": [  
                "botsay"  
            ]  
        },  
        "dotnetsay": {  
            "version": "2.1.3",  
            "commands": [  
                "dotnetsay"  
            ]  
        }  
    }  
}
```

2. Save your changes.

Making this change is the same as getting the latest version from the repository after someone else installed the package `dotnetsay` for the project directory.

3. Run the `dotnet tool restore` command.

```
dotnet tool restore
```

The command produces output like the following example:

```
Tool 'microsoft.botsay' (version '1.0.0') was restored. Available commands: botsay  
Tool 'dotnetsay' (version '2.1.3') was restored. Available commands: dotnetsay  
Restore was successful.
```

4. Verify that the tools are available:

```
dotnet tool list
```

The output is a list of packages and commands, similar to the following example:

Package Id	Version	Commands	Manifest
microsoft.botsay	1.0.0	botsay	/home/name/repository/.config/dotnet-tools.json
dotnetsay	2.1.3	dotnetsay	/home/name/repository/.config/dotnet-tools.json

5. Test the tools:

```
dotnet tool run dotnetsay hello from dotnetsay  
dotnet tool run botsay hello from botsay
```

Update a local tool

The installed version of local tool `dotnetsay` is 2.1.3. Use the `dotnet tool update` command to update the tool to the latest version.

```
dotnet tool update dotnetsay
```

The output indicates the new version number:

```
Tool 'dotnetsay' was successfully updated from version '2.1.3' to version '2.1.7'  
(manifest file /home/name/repository/.config/dotnet-tools.json).
```

The update command finds the first manifest file that contains the package ID and updates it. If there is no such package ID in any manifest file that is in the scope of the search, the SDK adds a new entry to the closest manifest file. The search scope is up through parent directories until a manifest file with `isRoot = true` is found.

Remove local tools

Remove the installed tools by running the [dotnet tool uninstall](#) command:

```
dotnet tool uninstall microsoft.botsay
```

```
dotnet tool uninstall dotnetsay
```

Troubleshoot

If you get an error message while following the tutorial, see [Troubleshoot .NET tool usage issues](#).

See also

For more information, see [.NET tools](#)

.NET additional tools overview

9/20/2022 • 2 minutes to read • [Edit Online](#)

This section compiles a list of tools that support and extend the .NET functionality, in addition to the .NET CLI.

.NET Uninstall Tool

The [.NET Uninstall Tool](#) (`dotnet-core-uninstall`) lets you clean up .NET SDKs and Runtimes on a system such that only the specified versions remain. A collection of options is available to specify which versions are uninstalled.

.NET diagnostic tools

[dotnet-counters](#) is a performance monitoring tool for first-level health monitoring and performance investigation.

[dotnet-dump](#) provides a way to collect and analyze Windows and Linux core dumps without a native debugger.

[dotnet-gcdump](#) provides a way to collect GC (Garbage Collector) dumps of live .NET processes.

[dotnet-monitor](#) provides a way to monitor .NET applications in production environments and to collect diagnostic artifacts (for example, dumps, traces, logs, and metrics) on-demand or using automated rules for collecting under specified conditions.

[dotnet-trace](#) collects profiling data from your app that can help in scenarios where you need to find out what causes an app to run slow.

.NET Install tool for extension authors

The [.NET Install tool for extension authors](#) is a Visual Studio Code extension that allows acquisition of the .NET runtime specifically for VS Code extension authors. This tool is intended to be leveraged in extensions that are written in .NET and require .NET to boot pieces of the extension (for example, a language server). The extension is not intended to be used directly by users to install .NET for development.

WCF Web Service Reference tool

The WCF (Windows Communication Foundation) [Web Service Reference tool](#) is a Visual Studio connected service provider that made its debut in [Visual Studio 2017 version 15.5](#). This tool retrieves metadata from a web service in the current solution, on a network location, or from a WSDL file. It generates a source file compatible with .NET, defining a WCF proxy class with methods that you can use to access the web service operations.

WCF dotnet-svcutil tool

The WCF [dotnet-svcutil tool](#) is a .NET tool that retrieves metadata from a web service on a network location or from a WSDL file. It generates a source file compatible with .NET, defining a WCF proxy class with methods that you can use to access the web service operations.

The [dotnet-svcutil](#) tool is an alternative to the [WCF Web Service Reference](#) Visual Studio connected service provider, which first shipped with Visual Studio 2017 version 15.5. The [dotnet-svcutil](#) tool, as a .NET tool, is available on Linux, macOS, and Windows.

WCF dotnet-svcutil.xmlserializer tool

On the .NET Framework, you can pre-generate a serialization assembly using the svcutil tool. The WCF [dotnet-svcutil.xmlserializer tool](#) provides similar functionality on .NET 5 (and .NET Core) and later versions. It pre-generates C# serialization code for the types in the client application that are used by the WCF Service Contract and that can be serialized by the [XmlSerializer](#). This improves the startup performance of XML serialization when serializing or deserializing objects of those types.

XML Serializer Generator

Like the [Xml Serializer Generator \(sgen.exe\)](#) for the .NET Framework, the [Microsoft.XmlSerializerGenerator NuGet package](#) is the solution for libraries that target .NET 5 (and .NET Core) and later versions. It creates an XML serialization assembly for types contained in an assembly to improve the startup performance of XML serialization when serializing or de-serializing objects of those types using [XmlSerializer](#).

Generating Self-Signed Certificates

You can use [dotnet dev-certs](#) to create self-signed certificates for development and testing scenarios.

.NET code coverage tool

You can use [dotnet-coverage](#) to collect [code coverage](#) from any .NET process.

.NET Uninstall Tool

9/20/2022 • 11 minutes to read • [Edit Online](#)

The [.NET Uninstall Tool](#) (`dotnet-core-uninstall`) lets you remove .NET SDKs and Runtimes from a system. A collection of options is available to specify which versions you want to uninstall.

The tool supports Windows and macOS. Linux is currently not supported.

On Windows, the tool can only uninstall SDKs and Runtimes that were installed using one of the following installers:

- The .NET SDK and runtime installer.
- The Visual Studio installer in versions earlier than Visual Studio 2019 version 16.3.

On macOS, the tool can only uninstall SDKs and runtimes located in the `/usr/local/share/dotnet` folder.

Because of these limitations, the tool may not be able to uninstall all of the .NET SDKs and runtimes on your machine. You can use the `dotnet --info` command to find all of the .NET SDKs and runtimes installed, including those SDKs and runtimes that this tool can't remove. The `dotnet-core-uninstall list` command displays which SDKs can be uninstalled with the tool. Versions 1.2 and later can uninstall SDKs and runtimes with version 5.0 or earlier, and previous versions of the tool can uninstall 3.1 and earlier.

Install the tool

You can download the .NET Uninstall Tool from [the tool's releases page](#) and find the source code at the [dotnet/cli-lab](#) GitHub repository.

NOTE

The tool requires elevation to uninstall .NET SDKs and runtimes. Therefore, it should be installed in a write-protected directory such as `C:\Program Files` on Windows or `/usr/local/bin` on macOS. See also [Elevated access for dotnet commands](#). For more information, see the [detailed installation instructions](#).

Run the tool

The following steps show the recommended approach for running the uninstall tool:

- [Step 1 - Display installed .NET SDKs and runtimes](#)
- [Step 2 - Do a dry run](#)
- [Step 3 - Uninstall .NET SDKs and Runtimes](#)
- [Step 4 - Delete the NuGet fallback folder \(optional\)](#)

Step 1 - Display installed .NET SDKs and runtimes

The `dotnet-core-uninstall list` command lists the installed .NET SDKs and runtimes that can be removed with this tool. Some SDKs and runtimes may be required by Visual Studio and they're displayed with a note of why it isn't recommended to uninstall them.

NOTE

The output of the `dotnet-core-uninstall list` command will not match the list of installed versions in the output of `dotnet --info` in most cases. Specifically, this tool will not display versions installed by zip files or managed by Visual Studio (any version installed with Visual Studio 2019 16.3 or later). One way to check if a version is managed by Visual Studio is to view it in `Add or Remove Programs`, where Visual Studio managed versions are marked as such in their display names.

For more information, see [list command](#) later in this article.

Step 2 - Do a dry run

The `dotnet-core-uninstall dry-run` and `dotnet-core-uninstall whatif` commands display the .NET SDKs and runtimes that will be removed based on the options provided without performing the uninstall. These commands are synonyms.

For more information, see [dry-run](#) and [whatif](#) commands later in this article.

Step 3 - Uninstall .NET SDKs and Runtimes

`dotnet-core-uninstall remove` uninstalls .NET SDKs and Runtimes that are specified by a collection of options. Versions 1.2 and later can uninstall SDKs and runtimes with version 5.0 or earlier, and previous versions of the tool can uninstall 3.1 and earlier.

Since this tool has a destructive behavior, it's **highly** recommended that you do a dry run before running the `remove` command. The dry run will show you what .NET SDKs and runtimes will be removed when you use the `remove` command. Refer to [Should I remove a version?](#) to learn which SDKs and runtimes are safe to remove.

Caution

Keep in mind the following caveats:

- This tool can uninstall versions of the .NET SDK that are required by `global.json` files on your machine. You can reinstall .NET SDKs from the [Download .NET](#) page.
- This tool can uninstall versions of the .NET Runtime that are required by framework dependent applications on your machine. You can reinstall .NET Runtimes from the [Download .NET](#) page.
- This tool can uninstall versions of the .NET SDK and runtime that Visual Studio relies on. If you break your Visual Studio installation, run "Repair" in the Visual Studio installer to get back to a working state.

By default, all commands keep the .NET SDKs and runtimes that may be required by Visual Studio or other SDKs. These SDKs and runtimes can be uninstalled by listing them explicitly as arguments or by using the `--force` option.

The tool requires elevation to uninstall .NET SDKs and runtimes. Run the tool in an Administrator command prompt on Windows and with `sudo` on macOS. The `dry-run` and `whatif` commands don't require elevation.

For more information, see [remove command](#) later in this article.

Step 4 - Delete the NuGet fallback folder (optional)

In some cases, you no longer need the `NuGetFallbackFolder` and may wish to delete it. For more information, see [Remove the NuGetFallbackFolder](#).

Uninstall the tool

- [Windows](#)
- [macOS](#)

1. Open [Add or Remove Programs](#).

2. Search for `Microsoft .NET SDK Uninstall Tool`.

3. Select **Uninstall**.

`list` command

Synopsis

```
dotnet-core-uninstall list [options]
```

Options

- [Windows](#)
- [macOS](#)

- `--aspnet-runtime`

Lists all the ASP.NET Runtimes that can be uninstalled with this tool.

- `--hosting-bundle`

Lists all the .NET hosting bundles that can be uninstalled with this tool.

- `--runtime`

Lists all .NET Runtimes that can be uninstalled with this tool.

- `--sdk`

Lists all .NET SDKs that can be uninstalled with this tool.

- `-v, --verbosity <LEVEL>`

Sets the verbosity level. Allowed values are `q[uiet]`, `m[inimal]`, `n[ormal]`, `d[etailed]`, and `diag[nostic]`. The default value is `normal`.

- `--x64`

Lists all x64 .NET SDKs and runtimes that can be uninstalled with this tool.

- `--x86`

Lists all x86 .NET SDKs and runtimes that can be uninstalled with this tool.

Examples

- List all .NET SDKs and runtimes that can be removed with this tool:

```
dotnet-core-uninstall list
```

- List all x64 .NET SDKs and runtimes:

```
dotnet-core-uninstall list --x64
```

- List all x86 .NET SDKs:

```
dotnet-core-uninstall list --sdk --x86
```

`dry-run` and `whatif` commands

Synopsis

```
dotnet-core-uninstall dry-run [options] [<VERSION>...]  
dotnet-core-uninstall whatif [options] [<VERSION>...]
```

Arguments

- `VERSION`

The specified version to uninstall. You may list several versions one after the other, separated by spaces. Response files are also supported.

TIP

Response files are an alternative to placing all the versions on the command line. They're text files, typically with a `*.rsp` extension, and each version is listed on a separate line. To specify a response file for the `VERSION` argument, use the `@` character immediately followed by the response file name.

Options

- [Windows](#)
- [macOS](#)
- `--all`

Removes all .NET SDKs and runtimes.

- `--all-below <VERSION>[<VERSION>...]`

Removes only the .NET SDKs and runtimes with a version smaller than the specified version. The specified version remains installed.

- `--all-but <VERSIONS>[<VERSION>...]`

Removes all .NET SDKs and runtimes, except those versions specified.

- `--all-but-latest`

Removes .NET SDKs and runtimes, except the one highest version.

- `--all-lower-patches`

Removes .NET SDKs and runtimes superseded by higher patches. This option protects `global.json`.

- `--all-previews`

Removes .NET SDKs and runtimes marked as previews.

- `--all-previews-but-latest`

Removes .NET SDKs and runtimes marked as previews except the one highest preview.

- `--aspnet-runtime`

Removes ASP.NET Runtimes only.

- `--hosting-bundle`

Removes .NET Runtime and hosting bundles only.

- `--major-minor <MAJOR_MINOR>`

Removes .NET SDKs and runtimes that match the specified `<MAJOR_MINOR>` version.

- `--runtime`

Removes .NET Runtimes only.

- `--sdk`

Removes .NET SDKs only.

- `-v, --verbosity <LEVEL>`

Sets the verbosity level. Allowed values are `q[uiet]`, `m[inimal]`, `n[ormal]`, `d[etailed]`, and `diag[nostic]`. The default value is `normal`.

- `--x64`

Must be used with `--sdk`, `--runtime`, and `--aspnet-runtime` to remove x64 SDKs or runtimes.

- `--x86`

Must be used with `--sdk`, `--runtime`, and `--aspnet-runtime` to remove x86 SDKs or runtimes.

- `--force` Forces removal of versions that might be used by Visual Studio.

Notes:

1. Exactly one of `--sdk`, `--runtime`, `--aspnet-runtime`, and `--hosting-bundle` is required.
2. `--all`, `--all-below`, `--all-but`, `--all-but-latest`, `--all-lower-patches`, `--all-previews`, `--all-previews-but-latest`, `--major-minor`, and `[<VERSION>...]` are exclusive.
3. If `--x64` or `--x86` aren't specified, then both x64 and x86 will be removed.

Examples

NOTE

By default, .NET SDKs and runtimes that may be required by Visual Studio or other SDKs are not included in `dotnet-core-uninstall dry-run` output. In the following examples, some of the specified SDKs and runtimes may not be included in the output, depending on the state of the machine. To include all SDKs and runtimes, list them explicitly as arguments or use the `--force` option.

- Dry run of removing all .NET Runtimes that have been superseded by higher patches:

```
dotnet-core-uninstall dry-run --all-lower-patches --runtime
```

- Dry run of removing all .NET SDKs below the version `2.2.301`:

```
dotnet-core-uninstall whatif --all-below 2.2.301 --sdk
```

`remove` command

Synopsis

```
dotnet-core-uninstall remove [options] [<VERSION>...]
```

Arguments

- `VERSION`

The specified version to uninstall. You may list several versions one after the other, separated by spaces. Response files are also supported.

TIP

Response files are an alternative to placing all the versions on the command line. They're text files, typically with a `*.rsp` extension, and each version is listed on a separate line. To specify a response file for the `VERSION` argument, use the `@` character immediately followed by the response file name.

Options

- [Windows](#)
- [macOS](#)

- `--all`

Removes all .NET SDKs and runtimes.

- `--all-below <VERSION>[<VERSION>...]`

Removes only the .NET SDKs and runtimes with a version smaller than the specified version. The specified version remains installed.

- `--all-but <VERSIONS>[<VERSION>...]`

Removes all .NET SDKs and runtimes, except those versions specified.

- `--all-but-latest`

Removes .NET SDKs and runtimes, except the one highest version.

- `--all-lower-patches`

Removes .NET SDKs and runtimes superseded by higher patches. This option protects `global.json`.

- `--all-previews`

Removes .NET SDKs and runtimes marked as previews.

- `--all-previews-but-latest`

Removes .NET SDKs and runtimes marked as previews except the one highest preview.

- `--aspnet-runtime`

Removes ASP.NET Runtimes only.

- `--hosting-bundle`

Removes .NET hosting bundles only.

- `--major-minor <MAJOR_MINOR>`

Removes .NET SDKs and runtimes that match the specified `major.minor` version.

- `--runtime`
Removes .NET Runtimes only.
- `--sdk`
Removes .NET SDKs only.
- `-v, --verbosity <LEVEL>`
Sets the verbosity level. Allowed values are `q[uiet]`, `m[inimal]`, `n[ormal]`, `d[etailed]`, and `diag[nostic]`. The default value is `normal`.
- `--x64`
Must be used with `--sdk`, `--runtime`, and `--aspnet-runtime` to remove x64 SDKs or runtimes.
- `--x86`
Must be used with `--sdk`, `--runtime`, and `--aspnet-runtime` to remove x86 SDKs or runtimes.
- `-y, --yes` Executes the command without requiring a yes or no confirmation.
- `--force` Forces removal of versions that might be used by Visual Studio.

Notes:

1. Exactly one of `--sdk`, `--runtime`, `--aspnet-runtime`, and `--hosting-bundle` is required.
2. `--all`, `--all-below`, `--all-but`, `--all-but-latest`, `--all-lower-patches`, `--all-previews`, `--all-previews-but-latest`, `--major-minor`, and `[<VERSION>...]` are exclusive.
3. If `--x64` or `--x86` aren't specified, then both x64 and x86 will be removed.

Examples

NOTE

By default, .NET SDKs and runtimes that may be required by Visual Studio or other SDKs are kept. In the following examples, some of the specified SDKs and runtimes may remain, depending on the state of the machine. To remove all SDKs and runtimes, list them explicitly as arguments or use the `--force` option.

- Remove all .NET Runtimes except the version `3.0.0-preview6-27804-01` without requiring Y/N confirmation:

```
dotnet-core-uninstall remove --all-but 3.0.0-preview6-27804-01 --runtime --yes
```

- Remove all .NET Core 1.1 SDKs without requiring Y/n confirmation:

```
dotnet-core-uninstall remove --sdk --major-minor 1.1 -y
```

- Remove the .NET Core 1.1.11 SDK with no console output:

```
dotnet-core-uninstall remove 1.1.11 --sdk --yes --verbosity q
```

- Remove all .NET SDKs that can safely be removed by this tool:

```
dotnet-core-uninstall remove --all --sdk
```

- Remove all .NET SDKs that can be removed by this tool, including those SDKs that may be required by Visual Studio (not recommended):

```
dotnet-core-uninstall remove --all --sdk --force
```

- Remove all .NET SDKs that are specified in the response file `versions.rsp`

```
dotnet-core-uninstall remove --sdk @versions.rsp
```

The content of `versions.rsp` is as follows:

```
2.2.300  
2.1.700
```

.NET install tool for extension authors

9/20/2022 • 2 minutes to read • [Edit Online](#)

The [.NET install tool for extension authors](#) is a Visual Studio Code extension that allows acquisition of the .NET runtime specifically for VS Code extension authors. This tool is intended to be leveraged in extensions that are written in .NET and require .NET to boot pieces of the extension (for example, a language server). The extension is not intended to be used directly by users to install .NET for development.

Getting started: extension authors

To ensure that the .NET install tool for extension authors is the right fit for your scenario, start by reviewing the [goals of this extension](#) on our GitHub page.

NOTE

This tool can be used to install the .NET runtime only, it currently does not have the capability to install the .NET SDK.

Once you have verified that the .NET install tool for extension authors fits your needs, you can take a dependency on it in your [extension manifest](#) and begin using the commands we expose with the [VS Code API](#). You can find the list of commands this extension exposes on our [GitHub](#).

Check out this [sample extension](#) to see these steps in action.

For more examples, check out these open source extensions that currently leverage this tool:

- [Azure Resource Manager \(ARM\) Tools for Visual Studio Code](#)
- [.NET Interactive Notebooks](#)

Getting started: end users

In general, the end user should not need to interact with the .NET install tool for extension authors at all. If you are having problems with the extension, check out our [troubleshooting page](#) or file an issue on our [GitHub](#).

Generate self-signed certificates with the .NET CLI

9/20/2022 • 7 minutes to read • [Edit Online](#)

There are different ways to create and use self-signed certificates for development and testing scenarios. This article covers using self-signed certificates with `dotnet dev-certs`, and other options like `PowerShell` and `OpenSSL`.

You can then validate that the certificate will load using an example such as an [ASP.NET Core app](#) hosted in a container.

Prerequisites

In the sample, you can utilize either .NET Core 3.1 or .NET 5.

For `dotnet dev-certs`, be sure to have the appropriate version of .NET installed:

- [Install .NET on Windows](#)
- [Install .NET on Linux](#)
- [Install .NET on macOS](#)

This sample requires [Docker 17.06](#) or later of the [Docker client](#).

Prepare sample app

You'll need to prepare the sample app depending on which runtime you'd like to use for testing, either [.NET Core 3.1](#) or [.NET 5](#).

For this guide, you'll use a [sample app](#) and make changes where appropriate.

.NET Core 3.1 sample app

Get the sample app.

```
git clone https://github.com/dotnet/dotnet-docker/
```

Navigate to the repository locally and open up the workspace in an editor.

NOTE

If you're looking to use dotnet publish parameters to *trim* the deployment, you should make sure that the appropriate dependencies are included for supporting SSL certificates. Update the `dotnet-docker\samples\aspnetapp\aspnetapp.csproj` to ensure that the appropriate assemblies are included in the container. For reference, check how to update the .csproj file to [support ssl certificates](#) when using trimming for self-contained deployments.

Make sure the `aspnetapp.csproj` includes the appropriate target framework:

```
<Project Sdk="Microsoft.NET.Sdk.Web">

<PropertyGroup>
  <TargetFramework>.netcoreapp3.1</TargetFramework>
  <!--Other Properties-->
</PropertyGroup>

</Project>
```

Modify the Dockerfile to make sure the runtime points to .NET Core 3.1:

```
# https://hub.docker.com/_/microsoft-dotnet-core
FROM mcr.microsoft.com/dotnet/core/sdk:3.1 AS build
WORKDIR /source

# copy csproj and restore as distinct layers
COPY *.sln .
COPY aspnetapp/*.csproj ./aspnetapp/
RUN dotnet restore

# copy everything else and build app
COPY aspnetapp/. ./aspnetapp/
WORKDIR /source/aspnetapp
RUN dotnet publish -c release -o /app --no-restore

# final stage/image
FROM mcr.microsoft.com/dotnet/core/aspnet:3.1
WORKDIR /app
COPY --from=build /app .
ENTRYPOINT ["dotnet", "aspnetapp.dll"]
```

Make sure you're pointing to the sample app.

```
cd ..\dotnet-docker\samples\aspnetapp
```

Build the container for testing locally.

```
docker build -t aspnetapp:my-sample -f Dockerfile .
```

.NET 5 sample app

For this guide, the [sample aspnetapp](#) should be checked for .NET 5.

Check sample app [Dockerfile](#) is using .NET 5.

Depending on the host OS, the ASP.NET runtime may need to be updated. For example, changing from `mcr.microsoft.com/dotnet/aspnet:5.0-nanoservercore-2009 AS runtime` to `mcr.microsoft.com/dotnet/aspnet:5.0-windowsservercore-ltsc2019 AS runtime` in the Dockerfile will help with targeting the appropriate Windows runtime.

For example, this will help with testing the certificates on Windows:

```

# https://hub.docker.com/_/microsoft-dotnet
FROM mcr.microsoft.com/dotnet/sdk:5.0 AS build
WORKDIR /source

# copy csproj and restore as distinct layers
COPY *.sln .
COPY aspnetapp/*.csproj ./aspnetapp/
RUN dotnet restore -r win-x64

# copy everything else and build app
COPY aspnetapp/. ./aspnetapp/
WORKDIR /source/aspnetapp
RUN dotnet publish -c release -o /app -r win-x64 --self-contained false --no-restore

# final stage/image
# Uses the 2009 release; 2004, 1909, and 1809 are other choices
FROM mcr.microsoft.com/dotnet/aspnet:5.0-windowsservercore-ltsc2019 AS runtime
WORKDIR /app
COPY --from=build /app ./
ENTRYPOINT ["aspnetapp"]

```

If we're testing the certificates on Linux, you can use the existing Dockerfile.

Make sure the `aspnetapp.csproj` includes the appropriate target framework:

```

<Project Sdk="Microsoft.NET.Sdk.Web">

<PropertyGroup>
  <TargetFramework>net5.0</TargetFramework>
  <!--Other Properties-->
</PropertyGroup>

</Project>

```

NOTE

If you want to use `dotnet publish` parameters to *trim* the deployment, make sure that the appropriate dependencies are included for supporting SSL certificates. Update the `dotnet-docker\samples\aspnetapp\aspnetapp.csproj` to ensure that the appropriate assemblies are included in the container. For reference, check how to update the .csproj file to [support ssl certificates](#) when using trimming for self-contained deployments.

Make sure you're pointing to the sample app.

```
cd .\dotnet-docker\samples\aspnetapp
```

Build the container for testing locally.

```
docker build -t aspnetapp:my-sample -f Dockerfile .
```

Create a self-signed certificate

You can create a self-signed certificate:

- [With dotnet dev-certs](#)
- [With PowerShell](#)

- With OpenSSL

With dotnet dev-certs

You can use `dotnet dev-certs` to work with self-signed certificates.

```
dotnet dev-certs https -ep $env:USERPROFILE\.aspnet\https\aspnetapp.pfx -p crypticpassword  
dotnet dev-certs https --trust
```

NOTE

The certificate name, in this case `aspnetapp.pfx` must match the project assembly name. `crypticpassword` is used as a stand-in for a password of your own choosing. If console returns "A valid HTTPS certificate is already present.", a trusted certificate already exists in your store. It can be exported using MMC Console.

Configure application secrets, for the certificate:

```
dotnet user-secrets -p aspnetapp\aspnetapp.csproj set "Kestrel:Certificates:Development:Password"  
"crypticpassword"
```

NOTE

Note: The password must match the password used for the certificate.

Run the container image with ASP.NET Core configured for HTTPS:

```
docker run --rm -it -p 8000:80 -p 8001:443 -e ASPNETCORE_URLS="https://+;http://+" -e  
ASPNETCORE_HTTPS_PORT=8001 -e ASPNETCORE_ENVIRONMENT=Development -v  
$env:APPDATA\microsoft\UserSecrets\:$env:APPDATA\microsoft\UserSecrets -v  
$env:USERPROFILE\.aspnet\https:$env:USERPROFILE\.aspnet\https -v  
mcr.microsoft.com/dotnet/samples:aspnetapp
```

Once the application starts, navigate to `https://localhost:8001` in your web browser.

Clean up

If the secrets and certificates aren't in use, be sure to clean them up.

```
dotnet user-secrets remove "Kestrel:Certificates:Development:Password" -p aspnetapp\aspnetapp.csproj  
dotnet dev-certs https --clean
```

With PowerShell

You can use PowerShell to generate self-signed certificates. The [PKI Client](#) can be used to generate a self-signed certificate.

```
$cert = New-SelfSignedCertificate -DnsName @("contoso.com", "www.contoso.com") -CertStoreLocation  
"cert:\LocalMachine\My"
```

The certificate will be generated, but for the purposes of testing, should be placed in a cert store for testing in a browser.

```
$certKeyPath = "c:\certs\contoso.com.pfx"
$password = ConvertTo-SecureString 'password' -AsPlainText -Force
$cert | Export-PfxCertificate -FilePath $certKeyPath -Password $password
$rootCert = $(Import-PfxCertificate -FilePath $certKeyPath -CertStoreLocation 'Cert:\LocalMachine\Root' -
>Password $password)
```

At this point, the certificates should be viewable from an [MMC snap-in](#).

You can run the sample container in Windows Subsystem for Linux (WSL):

```
docker run --rm -it -p 8000:80 -p 8001:443 -e ASPNETCORE_URLS="https://+;http://+" -e
ASPNETCORE_HTTPS_PORT=8001 -e ASPNETCORE_ENVIRONMENT=Development -e
ASPNETCORE_Kestrel_Certificates_Default_Password="password" -e
ASPNETCORE_Kestrel_Certificates_Default_Path=https://contoso.com.pfx -v /c/certs:/https/
mcr.microsoft.com/dotnet/samples:aspnetapp
```

NOTE

Note that with the volume mount the file path could be handled differently based on host. For example, in WSL we may replace `/c/certs` with `/mnt/c/certs`.

If you're using the container built earlier for Windows, the run command would look like the following:

```
docker run --rm -it -p 8000:80 -p 8001:443 -e ASPNETCORE_URLS="https://+;http://+" -e
ASPNETCORE_HTTPS_PORT=8001 -e ASPNETCORE_ENVIRONMENT=Development -e
ASPNETCORE_Kestrel_Certificates_Default_Password="password" -e
ASPNETCORE_Kestrel_Certificates_Default_Path=c:\https\contoso.com.pfx -v c:\certs:C:\https aspnetapp:my-
sample
```

Once the application is up, navigate to `contoso.com:8001` in a browser.

Be sure that the host entries are updated for `contoso.com` to answer on the appropriate IP address (for example 127.0.0.1). If the certificate isn't recognized, make sure that the certificate that is loaded with the container is also trusted on the host, and that there's appropriate SAN / DNS entries for `contoso.com`.

Clean up

```
$cert | Remove-Item
Get-ChildItem $certFilePath | Remove-Item
$rootCert | Remove-item
```

With OpenSSL

You can use [OpenSSL](#) to create self-signed certificates. This example will use WSL / Ubuntu and a bash shell with `OpenSSL`.

This will generate a .crt and a .key.

```

PARENT="contoso.com"
openssl req \
-x509 \
-newkey rsa:4096 \
-sha256 \
-days 365 \
-nodes \
-keyout $PARENT.key \
-out $PARENT.crt \
-subj "/CN=${PARENT}" \
-extensions v3_ca \
-extensions v3_req \
-config <(
echo '[req]'; \
echo 'default_bits= 4096'; \
echo 'distinguished_name=req'; \
echo 'x509_extension = v3_ca'; \
echo 'req_extensions = v3_req'; \
echo '[v3_req]'; \
echo 'basicConstraints = CA:FALSE'; \
echo 'keyUsage = nonRepudiation, digitalSignature, keyEncipherment'; \
echo 'subjectAltName = @alt_names'; \
echo '[ alt_names ]'; \
echo "DNS.1 = www.${PARENT}"; \
echo "DNS.2 = ${PARENT}"; \
echo '[ v3_ca ]'; \
echo 'subjectKeyIdentifier=hash'; \
echo 'authorityKeyIdentifier=keyid:always,issuer'; \
echo 'basicConstraints = critical, CA:TRUE, pathlen:0'; \
echo 'keyUsage = critical, cRLSign, keyCertSign'; \
echo 'extendedKeyUsage = serverAuth, clientAuth')

openssl x509 -noout -text -in $PARENT.crt

```

To get a .pfx, use the following command:

```
openssl pkcs12 -export -out $PARENT.pfx -inkey $PARENT.key -in $PARENT.crt
```

NOTE

The .aspnetcore 3.1 example will use `.pfx` and a password. Starting with the `.net 5` runtime, Kestrel can also take `.crt` and PEM-encoded `.key` files.

Depending on the host os, the certificate will need to be trusted. On a Linux host, 'trusting' the certificate is different and distro dependent.

For the purposes of this guide, here's an example in Windows using PowerShell:

```
Import-Certificate -FilePath $certFilePath -CertStoreLocation 'Cert:\LocalMachine\Root'
```

For .NET Core 3.1, run the following command in WSL:

```
docker run --rm -it -p 8000:80 -p 8001:443 -e ASPNETCORE_URLS="https://+;http://+" -e
ASPNETCORE_HTTPS_PORT=8001 -e ASPNETCORE_ENVIRONMENT=Development -e
ASPNETCORE_Kestrel_Certificates_Default_Password="password" -e
ASPNETCORE_Kestrel_Certificates_Default_Path=/https/contoso.com.pfx -v /c/path/to/certs/:/https/
mcr.microsoft.com/dotnet/samples:aspnetapp
```

Starting with .NET 5, Kestrel can take the `.crt` and PEM-encoded `.key` files. You can run the sample with the following command for .NET 5:

```
docker run --rm -it -p 8000:80 -p 8001:443 -e ASPNETCORE_URLS="https://+;http://+" -e  
ASPNETCORE_HTTPS_PORT=8001 -e ASPNETCORE_ENVIRONMENT=Development -e  
ASPNETCORE_Kestrel_Certificates_Default_Path=/https/contoso.com.crt -e  
ASPNETCORE_Kestrel_Certificates_Default_KeyPath=/https/contoso.com.key -v /c/path/to/certs:/https/  
mcr.microsoft.com/dotnet/samples:aspnetapp
```

NOTE

Note that in WSL, the volume mount path may change depending on the configuration.

For .NET Core 3.1 in Windows, run the following command in `Powershell`:

```
docker run --rm -it -p 8000:80 -p 8001:443 -e ASPNETCORE_URLS="https://+;http://+" -e  
ASPNETCORE_HTTPS_PORT=8001 -e ASPNETCORE_ENVIRONMENT=Development -e  
ASPNETCORE_Kestrel_Certificates_Default_Password="password" -e  
ASPNETCORE_Kestrel_Certificates_Default_Path=c:\https\contoso.com.pfx -v c:\certs:C:\https aspnetapp:my-  
sample
```

For .NET 5 in Windows, run the following command in PowerShell:

```
docker run --rm -it -p 8000:80 -p 8001:443 -e ASPNETCORE_URLS="https://+;http://+" -e  
ASPNETCORE_HTTPS_PORT=8001 -e ASPNETCORE_ENVIRONMENT=Development -e  
ASPNETCORE_Kestrel_Certificates_Default_Path=c:\https\contoso.com.crt -e  
ASPNETCORE_Kestrel_Certificates_Default_KeyPath=c:\https\contoso.com.key -v c:\certs:C:\https  
aspnetapp:my-sample
```

Once the application is up, navigate to `contoso.com:8001` in a browser.

Be sure that the host entries are updated for `contoso.com` to answer on the appropriate IP address (for example 127.0.0.1). If the certificate isn't recognized, make sure that the certificate that is loaded with the container is also trusted on the host, and that there's appropriate SAN / DNS entries for `contoso.com`.

Clean up

Be sure to clean up the self-signed certificates once done testing.

```
Get-ChildItem $certFilePath | Remove-Item
```

See also

- [dotnet dev-certs](#)

Use the WCF Web Service Reference Provider Tool

9/20/2022 • 3 minutes to read • [Edit Online](#)

Over the years, many Visual Studio developers have enjoyed the productivity that the [Add Service Reference](#) tool provided when their .NET Framework projects needed to access web services.

The **WCF Web Service Reference** tool is a Visual Studio connected service extension that lets you connect your .NET 5+, .NET Core, or ASP.NET Core project to a web service. It provides an experience similar to the [Add Service Reference](#) functionality, which is for .NET Framework projects only. The **WCF Web Service Reference** tool retrieves metadata from a web service in the current solution, on a network location, or from a WSDL file, and generates a source file containing Windows Communication Foundation (WCF) client proxy code that your .NET app can use to access the web service.

IMPORTANT

You should only reference services from a trusted source. Adding references from an untrusted source may compromise security.

Prerequisites

- [Visual Studio 2017 version 15.5](#) or a later version

The screenshots in this article are from Visual Studio 2022.

How to use the extension

NOTE

The **WCF Web Service Reference** tool is applicable only to C# .NET Core and .NET Standard projects, including ASP.NET Core Web apps.

Using the **ASP.NET Core Web Application** project template as an example, this article walks you through adding a WCF service reference to the project.

1. In Solution Explorer, double-click the **Connected Services** node of the project. (For a .NET Core or .NET Standard project, this option is available when you right-click on the **Dependencies** node of the project in Solution Explorer and choose **Manage Connected Services**.)

The **Connected Services** page appears as shown in the following image:

The screenshot shows the 'Connected Services' page in Visual Studio. The left sidebar has three tabs: 'Overview', 'Connected Services' (which is selected and highlighted in blue), and 'Publish'. The main content area has two sections. The first section, 'Service Dependencies', displays the message 'There are currently no service dependencies configured.' and a link 'Add a service dependency'. The second section, 'Service References (OpenAPI, gRPC, WCF Web Service)', also displays the message 'There are currently no service references configured.' and a link 'Add a service reference'. At the top right of the main content area are four icons: a green plus sign, a circular arrow, a refresh symbol, and a more options menu.

2. On the **Connected Services** page, select **Add Service Reference**.

The **Add service reference** page opens.

3. Select **WCF Web Service**, and then choose **Next**.

This brings up the **Add new WCF Web Service service reference** wizard.

The screenshot shows the 'Add new WCF Web Service service reference' wizard. The title bar says 'Add new WCF Web Service service reference'. The main area is titled 'Specify the service to add'. It contains several sections: 'URI:' with a dropdown containing 'http://localhost:8733/Design_Time_Addresses/MyWCFService/Service1/mex'; 'Services:' showing a tree with 'Service1' expanded and 'GreetingsService' selected; 'Operations:' showing a list with 'GetData' and 'GetDataUsingDataContract' selected; 'Status:' showing 'Number of services found: 1'; 'Namespace:' with the value 'ServiceReference1'; and a footer with buttons for 'Back', 'Next', 'Finish', and 'Cancel'. The 'Discover' button is also visible near the URI input.

4. Select a service.

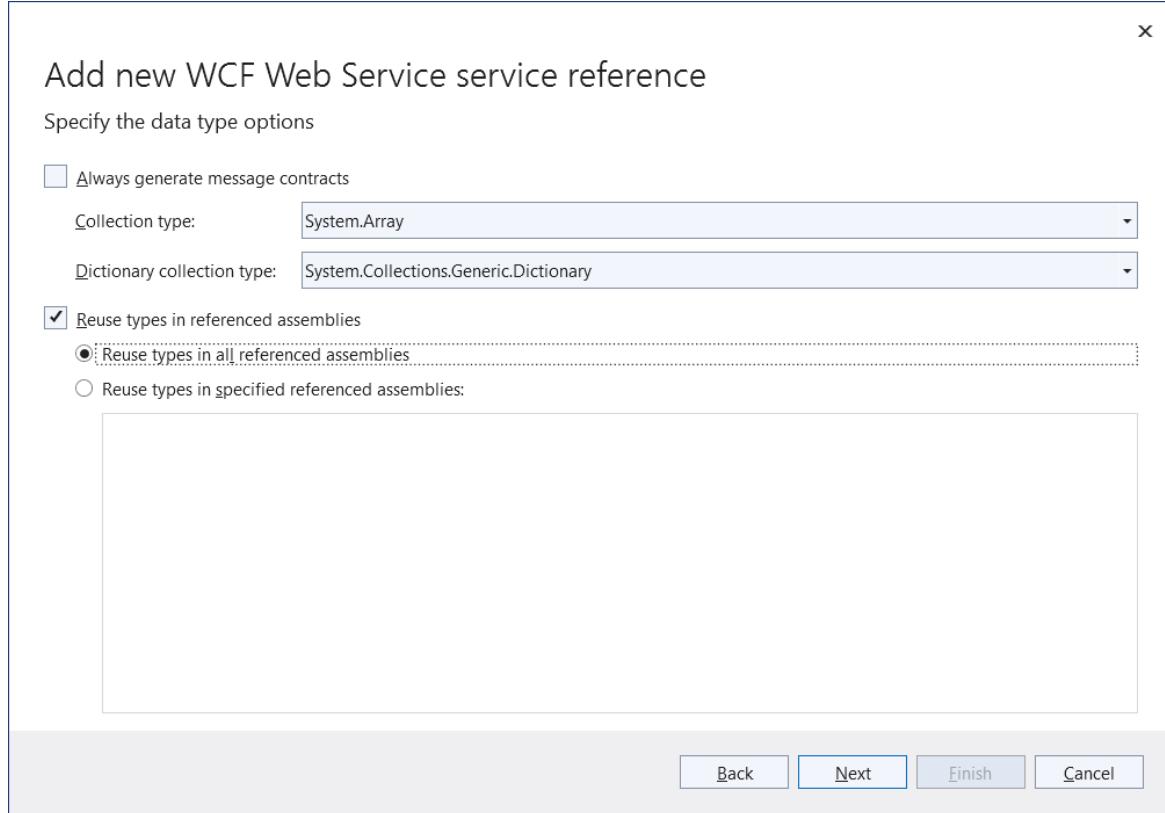
a. There are several services search options available:

- To search for services defined in the current solution, click the **Discover** button.
 - To search for services hosted at a specified address, enter a service URL in the **Address** box and click the **Go** button.
 - To select a WSDL file that contains the web service metadata information, click the **Browse** button.
- b. Select the service from the search results list in the **Services** box. If needed, enter the namespace for

the generated code in the corresponding **Namespace** text box.

c. Click the **Next** button to specify data type options or client options. Alternatively, click the **Finish** button to use the default options.

5. The **data type options** page lets refine the generated service reference configuration settings:



NOTE

The **Reuse types in referenced assemblies** check box option is useful when data types needed for service reference code generation are defined in one of your project's referenced assemblies. It's important to reuse those existing data types to avoid compile-time type clash or runtime issues.

There may be a delay while type information is loaded, depending on the number of project dependencies and other system performance factors. The **Finish** button is disabled during loading unless the **Reuse types in referenced assemblies** check box is unchecked.

6. Click **Finish** when you are done.

While displaying progress, the tool:

- Downloads metadata from the WCF service.
- Generates the service reference code in a file named *reference.cs*, and adds it to your project under the **Connected Services** node.
- Updates the project file (.csproj) with NuGet package references required to compile and run on the target platform.

Service reference configuration progress

| Importing web service metadata ...
Scaffolding service reference code ...
Number of service endpoints found: 1
Updating project ...
Done.

Successfully added service reference(s)

Automatically close when succeeded

[Back](#) [Next](#) [Close](#) [Cancel](#)

When these processes complete, you can create an instance of the generated WCF client type and invoke the service operations.

See also

- [Get started with Windows Communication Foundation applications](#)
- [Windows Communication Foundation services and WCF data services in Visual Studio](#)
- [WCF supported features on .NET Core](#)

Feedback & questions

If you have any product feedback, report it at [Developer Community](#) using the [Report a problem](#) tool.

Release notes

- Refer to the [Release notes](#) for updated release information, including known issues.

WCF dotnet-svcutil tool for .NET Core

9/20/2022 • 3 minutes to read • [Edit Online](#)

The Windows Communication Foundation (WCF) **dotnet-svcutil** tool is a .NET tool that retrieves metadata from a web service on a network location or from a WSDL file, and generates a WCF class containing client proxy methods that access the web service operations.

Similar to the [Service Model Metadata - svcutil](#) tool for .NET Framework projects, the **dotnet-svcutil** is a command-line tool for generating a web service reference compatible with .NET Core and .NET Standard projects.

The **dotnet-svcutil** tool is an alternative option to the [WCF Web Service Reference](#) Visual Studio connected service provider that first shipped with Visual Studio 2017 version 15.5. The **dotnet-svcutil** tool as a .NET tool, is available cross-platform on Linux, macOS, and Windows.

IMPORTANT

You should only reference services from a trusted source. Adding references from an untrusted source may compromise security.

Prerequisites

- [dotnet-svcutil 2.x](#)
- [dotnet-svcutil 1.x](#)
- [.NET Core 2.1 SDK](#) or later versions
- Your favorite code editor

Getting started

The following example walks you through the steps required to add a web service reference to a .NET Core web project and invoke the service. You'll create a .NET Core web application named *HelloSvcutil* and add a reference to a web service that implements the following contract:

```
[ServiceContract]
public interface ISayHello
{
    [OperationContract]
    string Hello(string name);
}
```

For this example, let's assume the web service will be hosted at the following address:

`http://contoso.com/SayHello.svc`

From a Windows, macOS, or Linux command window perform the following steps:

1. Create a directory named *HelloSvcutil* for your project and make it your current directory, as in the following example:

```
mkdir HelloSvculil  
cd HelloSvculil
```

2. Create a new C# web project in that directory using the `dotnet new` command as follows:

```
dotnet new web
```

3. Install the `dotnet-svcutil` NuGet package as a CLI tool:

- `dotnet-svcutil 2.x`
- `dotnet-svcutil 1.x`

```
dotnet tool install --global dotnet-svcutil
```

4. Run the `dotnet-svcutil` command to generate the web service reference file as follows:

- `dotnet-svcutil 2.x`
- `dotnet-svcutil 1.x`

```
dotnet-svcutil http://contoso.com/SayHello.svc
```

The generated file is saved as `HelloSvculil/ServiceReference/Reference.cs`. The `dotnet-svcutil` tool also adds to the project the appropriate WCF packages required by the proxy code as package references.

Using the Service Reference

1. Restore the WCF packages using the `dotnet restore` command as follows:

```
dotnet restore
```

2. Find the name of the client class and operation you want to use. `Reference.cs` will contain a class that inherits from `System.ServiceModel.ClientBase`, with methods that can be used to call operations on the service. In this example, you want to call the `SayHello` service's `Hello` operation.

`ServiceReference.SayHelloClient` is the name of the client class, and has a method called `HelloAsync` that can be used to call the operation.

3. Open the `Startup.cs` file in your editor, and add a `using` directive for the service reference namespace at the top:

```
using ServiceReference;
```

4. Edit the `Configure` method to invoke the web service. You do this by creating an instance of the class that inherits from `ClientBase` and calling the method on the client object:

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }

    app.Run(async (context) =>
    {
        var client = new SayHelloClient();
        var response = await client.HelloAsync();
        await context.Response.WriteAsync(response);
    });
}
```

- Run the application using the `dotnet run` command as follows:

```
dotnet run
```

- Navigate to the URL listed in the console (for example, `http://localhost:5000`) in your web browser.

You should see the following output: "Hello dotnet-svcutil!"

For a detailed description of the `dotnet-svcutil` tool parameters, invoke the tool passing the help parameter as follows:

- [dotnet-svcutil 2.x](#)
- [dotnet-svcutil 1.x](#)

```
dotnet-svcutil --help
```

Feedback & questions

If you have any questions or feedback, [open an issue on GitHub](#). You can also review any existing questions or issues at the [WCF repo on GitHub](#).

Release notes

- Refer to the [Release notes](#) for updated release information, including known issues.

Information

- [dotnet-svcutil NuGet Package](#)

Using dotnet-svcutil.xmlserializer on .NET Core

9/20/2022 • 2 minutes to read • [Edit Online](#)

The `dotnet-svcutil.xmlserializer` NuGet package can pre-generate a serialization assembly for .NET Core projects. It pre-generates C# serialization code for the types in the client application that are used by the WCF Service Contract and that can be serialized by the XmlSerializer. This improves the startup performance of XML serialization when serializing or deserializing objects of those types.

Prerequisites

- [.NET Core 2.1 SDK](#) or later
- Your favorite code editor

You can use the command `dotnet --info` to check which versions of .NET SDK and runtime you already have installed.

Getting started

To use `dotnet-svcutil.xmlserializer` in a .NET Core console application:

1. Create a WCF Service named 'MyWCFService' using the default template 'WCF Service Application' in .NET Framework. Add `[XmlSerializerFormat]` attribute on the service method like the following:

```
[ServiceContract]
public interface IService1
{
    [XmlSerializerFormat]
    [OperationContract(Action = "http://tempuri.org/IService1/GetData", ReplyAction =
    "http://tempuri.org/IService1/GetDataResponse")]
    string GetData(int value);
}
```

2. Create a .NET Core console application as WCF client application that targets at .NET Core 2.1 or later versions. For example, create an app named 'MyWCFClient' with the following command:

```
dotnet new console --name MyWCFClient
```

To ensure your project is targeting .NET Core 2.1 or later, inspect the `<TargetFramework>` XML element in your project file:

```
<TargetFramework>netcoreapp2.1</TargetFramework>
```

3. Add a package reference to `System.ServiceModel.Http` by running the following command:

```
dotnet add package System.ServiceModel.Http
```

4. Add the WCF Client code:

```

using System.ServiceModel;

class Program
{
    static void Main(string[] args)
    {
        var myBinding = new BasicHttpBinding();
        var myEndpoint = new EndpointAddress("http://localhost:2561/Service1.svc"); //Fill your
service url here
        var myChannelFactory = new ChannelFactory<IService1>(myBinding, myEndpoint);
        IService1 client = myChannelFactory.CreateChannel();
        string s = client.GetData(1);
        ((ICommunicationObject)client).Close();
    }
}

[ServiceContract]
public interface IService1
{
    [XmlSerializerFormat]
    [OperationContract(Action = "http://tempuri.org/IService1/GetData", ReplyAction =
"http://tempuri.org/IService1/GetDataResponse")]
    string GetData(int value);
}

```

5. Add a reference to the `dotnet-svcutil.xmlserializer` package by running the following command:

```
dotnet add package dotnet-svcutil.xmlserializer
```

Running the command should add an entry to your project file similar to this:

```

<ItemGroup>
    <DotNetCliToolReference Include="dotnet-svcutil.xmlserializer" Version="1.0.0" />
</ItemGroup>

```

6. Build the application by running `dotnet build`. If everything succeeds, an assembly named `MyWCFClient.XmlSerializers.dll` is generated in the output folder. If the tool failed to generate the assembly, you'll see warnings in the build output.
7. Start the WCF service by, for example, running `http://localhost:2561/Service1.svc` in the browser. Then start the client application, and it will automatically load and use the pre-generated serializers at run time.

Using Microsoft XML Serializer Generator on .NET Core

9/20/2022 • 2 minutes to read • [Edit Online](#)

This tutorial teaches you how to use the Microsoft XML Serializer Generator in a C# .NET Core application. During the course of this tutorial, you learn:

- How to create a .NET Core app
- How to add a reference to the Microsoft.XmlSerializerGenerator package
- How to edit your `MyApp.csproj` to add dependencies
- How to add a class and an `XmlSerializer`
- How to build and run the application

Like the [Xml Serializer Generator \(sgen.exe\)](#) for the .NET Framework, the [Microsoft.XmlSerializerGenerator NuGet package](#) is the equivalent for .NET Core and .NET Standard projects. It creates an XML serialization assembly for types contained in an assembly to improve the startup performance of XML serialization when serializing or de-serializing objects of those types using [XmlSerializer](#).

Prerequisites

To complete this tutorial:

- [.NET Core 2.1 SDK](#) or later.
- Your favorite code editor.

TIP

Need to install a code editor? Try [Visual Studio!](#)

Use Microsoft XML Serializer Generator in a .NET Core console application

The following instructions show you how to use XML Serializer Generator in a .NET Core console application.

Create a .NET Core console application

Open a command prompt and create a folder named `MyApp`. Navigate to the folder you created and type the following command:

```
dotnet new console
```

Add a reference to the Microsoft.XmlSerializerGenerator package in the MyApp project

Use the `dotnet add package` command to add the reference in your project.

Type:

```
dotnet add package Microsoft.XmlSerializerGenerator -v 1.0.0
```

Verify changes to `MyApp.csproj` after adding the package

Open your code editor and let's get started! We're still working from the `MyApp` directory we built the app in.

Open `MyApp.csproj` in your text editor.

After running the `dotnet add package` command, the following lines are added to your `MyApp.csproj` project file:

```
<ItemGroup>
  <PackageReference Include="Microsoft.XmlSerializer.Generator" Version="1.0.0" />
</ItemGroup>
```

Add another `ItemGroup` section for .NET CLI Tool support

Add the following lines after the `ItemGroup` section that we inspected:

```
<ItemGroup>
  <DotNetCliToolReference Include="Microsoft.XmlSerializer.Generator" Version="1.0.0" />
</ItemGroup>
```

Add a class in the application

Open `Program.cs` in your text editor. Add the class named `MyClass` in `Program.cs`.

```
public class MyClass
{
    public int Value;
}
```

Create an `XmlSerializer` for `MyClass`

Add the following line inside `Main` to create an `XmlSerializer` for `MyClass`:

```
var serializer = new System.Xml.Serialization.XmlSerializer(typeof(MyClass));
```

Build and run the application

Still within the `MyApp` folder, run the application via `dotnet run` and it automatically loads and uses the pre-generated serializers at run time.

Type the following command in your console window:

```
dotnet run
```

NOTE

`dotnet run` calls `dotnet build` to ensure that the build targets have been built, and then calls `dotnet <assembly.dll>` to run the target application.

IMPORTANT

The commands and steps shown in this tutorial to run your application are used during development time only. Once you're ready to deploy your app, take a look at the different [deployment strategies](#) for .NET Core apps and the `dotnet publish` command.

If everything succeeds, an assembly named *MyApp.XmlSerializers.dll* is generated in the output folder.

Congratulations! You have just:

- Created a .NET Core app.
- Added a reference to the Microsoft.XmlSerializer.Generator package.
- Edited your *MyApp.csproj* to add dependencies.
- Added a class and an *XmlSerializer*.
- Built and ran the application.

Further customize XML serialization assembly (optional)

Add the following XML to your *MyApp.csproj* to further customize assembly generation:

```
<PropertyGroup>
  <SGenReferences>C:\myfolder\abc.dll;C:\myfolder\def.dll</SGenReferences>
  <SGenTypes>MyApp.MyClass;MyApp.MyClass1</SGenTypes>
  <SGenProxyTypes>false</SGenProxyTypes>
  <SGenVerbose>true</SGenVerbose>
  <SGenKeyFile>mykey.snk</SGenKeyFile>
  <SGenDelaySign>true</SGenDelaySign>
</PropertyGroup>
```

Related resources

- [Introducing XML Serialization](#)
- [How to serialize using *XmlSerializer* \(C#\)](#)
- [How to: Serialize Using *XmlSerializer* \(Visual Basic\)](#)

What diagnostic tools are available in .NET Core?

9/20/2022 • 4 minutes to read • [Edit Online](#)

Software doesn't always behave as you would expect, but .NET Core has tools and APIs that will help you diagnose these issues quickly and effectively.

This article helps you find the various tools you need.

Debuggers

[Debuggers](#) allow you to interact with your program. Pausing, incrementally executing, examining, and resuming gives you insight into the behavior of your code. A debugger is a good choice for diagnosing functional problems that can be easily reproduced.

Unit testing

[Unit testing](#) is a key component of continuous integration and deployment of high-quality software. Unit tests are designed to give you an early warning when you break something.

Instrumentation for observability

.NET supports industry standard instrumentation techniques using metrics, logs, and distributed traces. Instrumentation is code that is added to a software project to record what it is doing. This information can then be collected in files, databases, or in-memory and analyzed to understand how a software program is operating. This is often used in production environments to monitor for problems and diagnose them. The .NET runtime has built-in instrumentation that can be optionally enabled and APIs that allow you to add custom instrumentation specialized for your application.

Metrics

[Metrics](#) are numerical measurements recorded over time to monitor application performance and health. Metrics are often used to generate alerts when potential problems are detected. Metrics have very low performance overhead and many services configure them as always-on telemetry.

Logs

[Logging](#) is a technique where code is instrumented to produce a log, a record of interesting events that occurred while the program was running. Often a baseline set of log events are configured on by default and more extensive logging can be enabled on-demand to diagnose particular problems. Performance overhead is variable depending on how much data is being logged.

Distributed traces

[Distributed Tracing](#) is a specialized form of logging that helps you localize failures and performance issues within applications distributed across multiple machines or processes. This technique tracks requests through an application correlating together work done by different application components and separating it from other work the application may be doing for concurrent requests. It is possible to trace every request and sampling can be optionally employed to bound the performance overhead.

Dumps

A [dump](#) is a file that contains a snapshot of the process at the time of creation. These can be useful for examining the state of your application for debugging purposes.

Symbols

[Symbols](#) are a mapping between the source code and the binary produced by the compiler. These are commonly used by .NET debuggers to resolve source line numbers, local variable names, and other types of diagnostic information.

Collect diagnostics in containers

The same diagnostics tools that are used in non-containerized Linux environments can also be used to [collect diagnostics in containers](#). There are just a few usage changes needed to make sure the tools work in a Docker container.

.NET Core diagnostic global tools

dotnet-counters

[dotnet-counters](#) is a performance monitoring tool for first-level health monitoring and performance investigation. It observes performance counter values published via the [EventCounter](#) API. For example, you can quickly monitor things like the CPU usage or the rate of exceptions being thrown in your .NET Core application.

dotnet-dump

The [dotnet-dump](#) tool is a way to collect and analyze Windows and Linux core dumps without a native debugger.

dotnet-gcdump

The [dotnet-gcdump](#) tool is a way to collect GC (Garbage Collector) dumps of live .NET processes.

dotnet-monitor

The [dotnet-monitor](#) tool is a way to monitor .NET applications in production environments and to collect diagnostic artifacts (for example, dumps, traces, logs, and metrics) on-demand or using automated rules for collecting under specified conditions.

dotnet-trace

.NET Core includes what is called the [EventPipe](#) through which diagnostics data is exposed. The [dotnet-trace](#) tool allows you to consume interesting profiling data from your app that can help in scenarios where you need to root cause apps running slow.

dotnet-stack

The [dotnet-stack](#) tool allows you to quickly print the managed stacks for all threads in a running .NET process.

dotnet-symbol

[dotnet-symbol](#) downloads files (symbols, DAC/DBI, host files, etc.) needed to open a core dump or minidump. Use this tool if you need symbols and modules to debug a dump file captured on a different machine.

dotnet-sos

[dotnet-sos](#) installs the [SOS debugging extension](#) on Linux and macOS (and on Windows if you're using [Windbg/cdb](#)).

PerfCollect

[PerfCollect](#) is a bash script you can use to collect traces with [perf](#) and [LTTng](#) for a more in-depth performance analysis of .NET apps running on Linux distributions.

.NET Core diagnostics tutorials

Write your own diagnostic tool

The [diagnostics client library](#) lets you write your own custom diagnostic tool best suited for your diagnostic

scenario. Look up information in the [Microsoft.Diagnostics.NETCore.Client API reference](#).

Debug a memory leak

[Tutorial: Debug a memory leak](#) walks through finding a memory leak. The [dotnet-counters](#) tool is used to confirm the leak and the [dotnet-dump](#) tool is used to diagnose the leak.

Debug high CPU usage

[Tutorial: Debug high CPU usage](#) walks you through investigating high CPU usage. It uses the [dotnet-counters](#) tool to confirm the high CPU usage. It then walks you through using [Trace for performance analysis utility \(dotnet-trace\)](#) or Linux `perf` to collect and view CPU usage profile.

Debug deadlock

[Tutorial: Debug deadlock](#) shows you how to use the [dotnet-dump](#) tool to investigate threads and locks.

Debug ThreadPool Starvation

[Tutorial: Debug threadPool starvation](#) shows you how to use the [dotnet-counters](#) and [dotnet-stack](#) tools to investigate ThreadPool starvation.

Debug a StackOverflow

[Tutorial: Debug a StackOverflow](#) demonstrates how to debug a [StackOverflowException](#) on Linux.

Debug Linux dumps

[Debug Linux dumps](#) explains how to collect and analyze dumps on Linux.

Measure performance using EventCounters

[Tutorial: Measure performance using EventCounters in .NET](#) shows you how to use the [EventCounter API](#) to measure performance in your .NET app.

.NET Core managed debuggers

9/20/2022 • 2 minutes to read • [Edit Online](#)

Debuggers allow programs to be paused or executed step-by-step. When paused, the current state of the process can be viewed. By stepping through key sections, you gain understanding of your code and why it produces the result that it does.

Microsoft provides debuggers for managed code in **Visual Studio** and **Visual Studio Code**.

Visual Studio managed debugger

Visual Studio is an integrated development environment with the most comprehensive debugger available. Visual Studio is an excellent choice for developers working on Windows.

- [Tutorial - Debugging a .NET Core application on Windows with Visual Studio](#)
- [Debug ASP.NET Core apps in Visual Studio](#)

While Visual Studio is a Windows application, it can also be used to debug Linux apps running remotely, in WSL, or in Docker containers:

- [Remotely debug a .NET Core app on Linux](#)
- [Debug a .NET Core app in WSL2](#)
- [Debug a .NET Core app in a Docker container](#)

Visual Studio Code managed debugger

Visual Studio Code is a lightweight cross-platform code editor. It uses the same .NET Core debugger implementation as Visual Studio, but with a simplified user interface.

- [Tutorial - Debugging a .NET Core application with Visual Studio Code](#)
- [Debugging in Visual Studio Code](#)

Diagnostic port

9/20/2022 • 6 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 3.1 and later versions

The .NET Core runtime exposes a service endpoint that allows other processes to send diagnostic commands and receive responses over an [IPC channel](#). This endpoint is called a *diagnostic port*. Commands can be sent to the diagnostic port to:

- Capture a memory dump.
- Start an [EventPipe](#) trace.
- Request the command-line used to launch the app.

The diagnostic port supports different transports depending on platform. Currently both the CoreCLR and Mono runtime implementations use Named Pipes on Windows and Unix Domain Sockets on Linux and macOS. The Mono runtime implementation on Android, iOS, and tvOS uses TCP/IP. The channel uses a [custom binary protocol](#). Most developers will never directly interact with the underlying channel and protocol, but rather will use GUI or CLI tools that communicate on their behalf. For example, the [dotnet-dump](#) and [dotnet-trace](#) tools abstract sending protocol commands to capture dumps and start traces. For developers that want to write custom tooling, the [Microsoft.Diagnostics.NETCore.Client NuGet package](#) provides a .NET API abstraction of the underlying transport and protocol.

Security considerations

The diagnostic port exposes sensitive information about a running application. If an untrusted user gains access to this channel they could observe detailed program state, including any secrets that are in memory, and arbitrarily modify the execution of the program. On the CoreCLR runtime, the default diagnostic port is configured to only be accessible by the same user account that launched the app or by an account with super-user permissions. If your security model does not trust other processes with the same user account credentials, all diagnostic ports can be disabled by setting environment variable `DOTNET_EnableDiagnostics=0`. Doing this will block your ability to use external tooling such as .NET debugging or any of the `dotnet-*` diagnostic tools.

NOTE

.NET 6 standardizes on the prefix `DOTNET_` instead of `COMPlus_` for environment variables that configure .NET run-time behavior. However, the `COMPlus_` prefix will continue to work. If you're using a previous version of the .NET runtime, you should still use the `COMPlus_` prefix for environment variables.

Default diagnostic port

On Windows, Linux, and macOS, the runtime has one diagnostic port open by default at a well-known endpoint. This is the port that the `dotnet-*` diagnostic tools are connecting to automatically when they haven't been explicitly configured to use an alternate port. The endpoint is:

- Windows - Named Pipe `\\\.\pipe\dotnet-diagnostic-{pid}`
- Linux and macOS - Unix Domain Socket `{temp}/dotnet-diagnostic-{pid}-{disambiguation_key}-socket`

`{pid}` is the process id written in decimal, `{temp}` is the `TMPDIR` environment variable or the value `/tmp` if `TMPDIR` is undefined/empty, and `{disambiguation_key}` is the process start time written in decimal. On macOS and NetBSD, the process start time is number of seconds since UNIX epoch time and on all other platforms it is

jiffies since boot time.

Suspend the runtime at startup

By default, the runtime executes managed code as soon as it starts, regardless whether any diagnostic tools have connected to the diagnostic port. Sometimes it is useful to have the runtime wait to run managed code until after a diagnostic tool is connected, to observe the initial program behavior. Setting environment variable `DOTNET_DefaultDiagnosticPortSuspend=1` causes the runtime to wait until a tool connects to the default port. If no tool is attached after several seconds, the runtime prints a warning message to the console explaining that it is still waiting for a tool to attach.

Configure additional diagnostic ports

IMPORTANT

This works for apps running .NET 5 or later only.

Both the Mono and CoreCLR runtimes can use custom configured diagnostic ports. These ports are in addition to the default port that remains available. There are a few common reasons this is useful:

- On Android, iOS and tvOS there is no default port, so configuring a port is necessary to use diagnostic tools.
- In environments with containers or firewalls, you may want to set up a predictable endpoint address that doesn't vary based on process id as the default port does. Then the custom port can be explicitly added to an allow list or proxied across some security boundary.
- For monitoring tools it is useful to have the tool listen on an endpoint, and the runtime actively attempts to connect to it. This avoids needing the monitoring tool to continuously poll for new apps starting. In environments where the default diagnostic port isn't accessible, it also avoids needing to configure the monitor with a custom endpoint for each monitored app.

In each communication channel between a diagnostic tool and the .NET runtime, one side needs to be the listener and wait for the other side to connect. The runtime can be configured to act in either role for each port. Ports can also be independently configured to suspend at startup, waiting for a diagnostic tool to issue a resume command. Ports configured to connect will repeat their connection attempts indefinitely if the remote endpoint isn't listening or if the connection is lost, but the app does not automatically suspend managed code while waiting to establish that connection. If you want the app to wait for a connection to be established, use the suspend at startup option.

Custom ports are configured using the `DOTNET_DiagnosticPorts` environment variable. This variable should be set to a semicolon delimited list of port descriptions. Each port description consists of an endpoint address and optional modifiers that control the runtime's connect or listen role and if the runtime should suspend on startup. On Windows, the endpoint address is the name of a named pipe without the `\.\pipe\` prefix. On Linux and macOS it is the full path to a Unix Domain Socket. On Android, iOS, and tvOS the address is an IP and port. For example:

1. `DOTNET_DiagnosticPorts=my_diag_port1` - (Windows) The runtime connects to the named pipe `\.\pipe\my_diag_port1`.
2. `DOTNET_DiagnosticPorts=/foo/tool1.socket;foo/tool2.socket` - (Linux and macOS) The runtime connects to both the Unix Domain Sockets `/foo/tool1.socket` and `/foo/tool2.socket`.
3. `DOTNET_DiagnosticPorts=127.0.0.1:9000` - (Android, iOS, and tvOS) The runtime connects to IP 127.0.0.1 on port 9000.
4. `DOTNET_DiagnosticPorts=/foo/tool1.socket,listen,nosuspend` - (Linux and macOS) This example has the `listen` and `nosuspend` modifiers. The runtime creates and listens to Unix Domain Socket `/foo/tool1.socket`.

instead of connecting to it. Additional diagnostic ports would normally cause the runtime to suspend at startup waiting for a resume command, but `nosuspend` causes the runtime not to wait.

The complete syntax for a port is `address[,,(listen|connect)][,(suspend|nosuspend)]`. `connect` is the default if neither `connect` or `listen` is specified. `suspend` is the default if neither `suspend` or `nosuspend` is specified.

Usage in dotnet diagnostic tools

Tools such as [dotnet-dump](#), [dotnet-counters](#), or [dotnet-trace](#) all support `collect` or `monitor` verbs which communicate to a .NET app via the diagnostic port. When these tools are using the `--processId` argument the tool automatically computes the default diagnostic port address and connects to it. When specifying the `--diagnostic-port` argument, the tool listens at the given address and you should use the `DOTNET_DiagnosticPorts` environment variable to configure your app to connect. For a complete example with `dotnet-counters`, see [Using the Diagnostic Port](#).

Use ds-router to proxy the diagnostic port

All of the `dotnet-*` diagnostic tools expect to connect to a diagnostic port that is a local Named Pipe or Unix Domain Socket. Mono often runs on isolated hardware or in emulators that need a proxy over TCP to become accessible. The [dotnet-dsrouter tool](#) can proxy a local Named Pipe or Unix Domain Socket to TCP so that the tools can be used in those environments. For more information, see [dotnet-dsrouter](#).

Dumps

9/20/2022 • 3 minutes to read • [Edit Online](#)

A dump is a file that contains a snapshot of the process at the time it was created and can be useful for examining the state of your application. Dumps can be used to debug your .NET application when it is difficult to attach a debugger to it such as production or CI environments. Using dumps allows you to capture the state of the problematic process and examine it without having to stop the application.

Collect dumps

Dumps can be collected in a variety of ways depending on which platform you are running your app on.

NOTE

Collecting a dump inside a container requires PTRACE capability, which can be added via `--cap-add=SYS_PTRACE` or `--privileged`.

NOTE

Dumps may contain sensitive information because they can contain the full memory of the running process. Handle them with any security restrictions and guidances in mind.

Collect dumps on crash

You can use environment variables to configure your application to collect a dump upon a crash. This is helpful when you want to get an understanding of why a crash happened. For example, capturing a dump when an exception is thrown helps you identify an issue by examining the state of the app when it crashed.

The following table shows the environment variables you can configure for collecting dumps on a crash.

ENVIRONMENT VARIABLE	DESCRIPTION	DEFAULT VALUE
<code>COMPlus_DbgEnableMiniDump</code> or <code>DOTNET_DbgEnableMiniDump</code>	If set to 1, enable core dump generation.	0
<code>COMPlus_DbgMiniDumpType</code> or <code>DOTNET_DbgMiniDumpType</code>	Type of dump to be collected. For more information, see the table below	2 (<code>MiniDumpWithPrivateReadWriteMemory</code>)
<code>COMPlus_DbgMiniDumpName</code> or <code>DOTNET_DbgMiniDumpName</code>	Path to a file to write the dump to.	<code>/tmp/coredump.<pid></code>
<code>COMPlus_CreateDumpDiagnostics</code> or <code>DOTNET_CreateDumpDiagnostics</code>	If set to 1, enable diagnostic logging of dump process.	0

ENVIRONMENT VARIABLE	DESCRIPTION	DEFAULT VALUE
<code>COMPlus_EnableCrashReport</code> or <code>DOTNET_EnableCrashReport</code>	(Requires .NET 6 or later) If set to 1, the runtime generates a JSON-formatted crash report that includes information about the threads and stack frames of the crashing application. The crash report name is the dump path/name with <code>.crashreport.json</code> appended.	
<code>COMPlus_CreateDumpVerboseDiagnostics</code> or <code>DOTNET_CreateDumpVerboseDiagnostics</code>	(Requires .NET 7 or later) If set to 1, enables verbose diagnostic logging of the dump process.	0
<code>COMPlus_CreateDumpLogFile</code> or <code>DOTNET_CreateDumpLogFile</code>	(Requires .NET 7 or later) The path of the file to which the diagnostic messages should be written. If unset, the diagnostic messages are written to the console of the crashing application.	

NOTE

.NET 7 standardizes on the prefix `DOTNET_` instead of `COMPlus_` for these environment variables. However, the `COMPlus_` prefix will continue to work. If you're using a previous version of the .NET runtime, you should still use the `COMPlus_` prefix for environment variables.

Starting in .NET 5, `DOTNET_MiniDumpName` may also include formatting template specifiers that will be filled in dynamically:

SPECIFIER	VALUE
<code>%%</code>	A single % character
<code>%p</code>	PID of dumped process
<code>%e</code>	The process executable filename
<code>%h</code>	Host name return by <code>gethostname()</code>
<code>%t</code>	Time of dump, expressed as seconds since the Epoch, 1970-01-01 00:00:00 +0000 (UTC)

The following table shows all the values you can use for `DOTNET_DbgMiniDumpType`. For example, setting `DOTNET_DbgMiniDumpType` to 1 means `MiniDumpNormal` type dump will be collected on a crash.

VALUE	NAME	DESCRIPTION
1	<code>Mini</code>	A small dump containing module lists, thread lists, exception information, and all stacks.

Value	Name	Description
2	Heap	A large and relatively comprehensive dump containing module lists, thread lists, all stacks, exception information, handle information, and all memory except for mapped images.
3	Triage	Same as <code>Mini</code> , but removes personal user information, such as paths and passwords.
4	Full	The largest dump containing all memory including the module images.

Collect dumps at a specific point in time

You may want to collect a dump when the app hasn't crashed yet. For example, if you want to examine the state of an application that seems to be in a deadlock, configuring the environment variables to collect dumps on crash will not be helpful because the app is still running.

To collect dump at your own request, you can use `dotnet-dump`, which is a CLI tool for collecting and analyzing dumps. For more information on how to use it to collect dumps with `dotnet-dump`, see [Dump collection and analysis utility](#).

Analyze dumps

You can analyze dumps using the `dotnet-dump` CLI tool or with [Visual Studio](#).

NOTE

Visual Studio version 16.8 and later allows you to [open Linux dumps](#) generated on .NET Core 3.1.7 or later.

NOTE

If native debugging is necessary, the [SOS debugger extension](#) can be used with [LLDB on Linux and macOS](#). SOS is also supported with [Windbg/cdb](#) on Windows, although Visual Studio is recommended.

See also

Learn more about how you can leverage dumps to help diagnosing problems in your .NET application.

- [Debug Linux dumps](#) tutorial walks you through how to debug a dump that was collected in Linux.
- [Debug deadlock](#) tutorial walks you through how to debug a deadlock in your .NET application using dumps.

Debug Linux dumps

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 3.0 SDK and later versions

Collect dumps on Linux

The two recommended ways of collecting dumps on Linux are:

- `dotnet-dump` CLI tool
- [Environment variables](#) that collect dumps on crashes

Analyze dumps on Linux

After a dump is collected, it can be analyzed using the `dotnet-dump` tool with the `dotnet-dump analyze` command. This analysis step needs to be run on a machine that has the same architecture and Linux distro as the environment the dump was captured in. The `dotnet-dump` tool supports displaying information about .NET code, but is not useful for understanding code issues for other languages like C and C++.

Alternatively, [LLDB](#) can be used to analyze dumps on Linux, which allows analysis of both managed and native code. LLDB uses the SOS extension to debug managed code. The `dotnet-sos` CLI tool can be used to install SOS, which has [many useful commands](#) for debugging managed code. In order to analyze .NET Core dumps, LLDB and SOS require the following .NET Core binaries from the environment the dump was created in:

1. libmscordaccore.so
2. libcoreclr.so
3. dotnet (the host used to launch the app)

In most cases, these binaries can be downloaded using the `dotnet-symbol` tool. If the necessary binaries can't be downloaded with `dotnet-symbol` (for example, if a private version of .NET Core built from source was in use), it may be necessary to copy the files listed above from the environment the dump was created in. If the files aren't located next to the dump file, you can use the LLDB/SOS command `setclrpath <path>` to set the path they should be loaded from and `setsymbolserver -directory <path>` to set the path to look in for symbol files.

Once the necessary files are available, the dump can be loaded in LLDB by specifying the dotnet host as the executable to debug:

```
lldb --core <dump-file> <host-program>
```

In the above command line, `<dump-file>` is the path of the dump to analyze and `<host-program>` is the native program that started the .NET Core application. This is typically the `dotnet` binary unless the app is self-contained, in which case it is the name of the application without the dll extension.

Once LLDB starts, it may be necessary to use the `setsymbolserver` command to point at the correct symbol location (`setsymbolserver -ms` to use Microsoft's symbol server or `setsymbolserver -directory <path>` to specify a local path). Native symbols can be loaded by running `loadsymbols`. At this point, [SOS commands](#) can be used to analyze the dump.

Analyze dumps on Windows

Dumps collected from a Linux machine can also be analyzed on a Windows machine using [Visual Studio](#), [Windbg](#), or the [dotnet-dump](#) tool.

- **Visual Studio** - See the [Visual Studio dump debugging guide](#).
- **Windbg** - You can debug Linux dumps on windbg using the [same instructions](#) you would use to debug a Windows user-mode dump. Use the x64 version of windbg for dumps collected from a Linux x64 or Arm64 environment and the x86 version for dumps collected from a Linux x86 environment.
- **dotnet-dump** - View the dump using the [dotnet-dump analyze](#) command. Use the x64 version of dotnet-dump for dumps collected from a Linux x64 or Arm64 environment and the x86 version for dumps collected from a Linux x86 environment.

See also

- [dotnet-sos](#) for more details on installing the SOS extension.
- [dotnet-symbol](#) for more details on installing and using the symbol download tool.
- [.NET Core diagnostics repo](#) for more details on debugging, including a useful FAQ.
- [Installing LLDB](#) for instructions on installing LLDB on Linux or Mac.

SOS debugging extension

9/20/2022 • 26 minutes to read • [Edit Online](#)

The SOS Debugging Extension lets you view information about code that is running inside the .NET Core runtime, both on live processes and dumps. The extension is preinstalled with [dotnet-dump](#) and [Windbg/dbg](#), and can be [downloaded](#) for use with LLDB. You can use the SOS Debugging Extension to:

- Collect information about the managed heap.
- Look for heap corruptions.
- Display internal data types used by the runtime.
- View information about all managed code running inside the runtime.

Syntax

Windows

```
![command] [options]
```

Linux and macOS

```
sos [command] [options]
```

Many of the commands have aliases or shortcuts under lldb:

```
clystack [options]
```

Commands

The following table of commands is also available under [Help](#) or [soshelp](#). Individual command help is available using `soshelp <command>`.

COMMAND	DESCRIPTION
<code>bpm [-nofuturemodule] [<module name> <method name>] [-md <MethodDesc>] -list -clear <pending breakpoint number> -clearall</code>	<p>Creates a breakpoint at the specified method in the specified module.</p> <p>If the specified module and method have not been loaded, this command waits for a notification that the module was loaded and just-in-time (JIT) compiled before creating a breakpoint.</p> <p>You can manage the list of pending breakpoints by using the <code>-list</code>, <code>-clear</code>, and <code>-clearall</code> options:</p> <p>The <code>-list</code> option generates a list of all the pending breakpoints. If a pending breakpoint has a non-zero module ID, that breakpoint is specific to a function in that particular loaded module. If the pending breakpoint has a zero module ID, that breakpoint applies to modules that have not yet been loaded.</p> <p>Use the <code>-clear</code> or <code>-clearall</code> option to remove pending breakpoints from the list.</p>

COMMAND	DESCRIPTION
<p>CLRStack [-a] [-l] [-p] [-n] [-f] [-r] [-all]</p>	<p>Provides a stack trace of managed code only.</p> <p>The -p option shows arguments to the managed function.</p> <p>The -l option shows information on local variables in a frame. The SOS Debugging Extension cannot retrieve local names, so the output for local names is in the format <i><local/ address> = <value></i>.</p> <p>The -a option is a shortcut for -l and -p combined.</p> <p>The -n option disables the display of source file names and line numbers. If the debugger has the option SYMOPT_LOAD_LINES specified, SOS will look up the symbols for every managed frame and if successful will display the corresponding source file name and line number. The -n (No line numbers) parameter can be specified to disable this behavior.</p> <p>The -f option (full mode) displays the native frames intermixing them with the managed frames and the assembly name and function offset for the managed frames. This option does not display native frames when used with <code>dotnet-dump</code>.</p> <p>The -r option dumps the registers for each stack frame.</p> <p>The -all option dumps all the managed threads' stacks.</p>
<p>COMState</p>	<p>Lists the COM apartment model for each thread and a <code>Context</code> pointer, if available. This command is only supported on Windows.</p>
<p>DumpArray [-start <i><startIndex></i>] [-length <i><length></i>] [-details] [-nofields] <i><array object address></i></p> <p>-or-</p> <p>DA [-start <i><startIndex></i>] [-length <i><length></i>] [-detail] [-nofields] <i><array object address></i></p>	<p>Examines elements of an array object.</p> <p>The -start option specifies the starting index at which to display elements.</p> <p>The -length option specifies how many elements to show.</p> <p>The -details option displays details of the element using the DumpObj and DumpVC formats.</p> <p>The -nofields option prevents arrays from displaying. This option is available only when the -detail option is specified.</p>

COMMAND	DESCRIPTION
<pre>DumpAsync (dumpasync) [-mt <MethodTable address>] [-type <partial type name>] [-waiting] [-roots]</pre>	<p>DumpAsync traverses the garbage collected heap and looks for objects representing async state machines as created when an async method's state is transferred to the heap. This command recognizes async state machines defined as <code>async void</code>, <code>async Task</code>, <code>async Task<T></code>, <code>async ValueTask</code>, and <code>async ValueTask<T></code>.</p> <p>The output includes a block of details for each async state machine object found. These details include:</p> <ul style="list-style-type: none"> - A line for the type of the async state machine object, including its MethodTable address, its object address, its size, and its type name. - A line for the state machine type name as contained in the object. - A listing of each field on the state machine. - A line for a continuation from this state machine object, if one or more has been registered. - Discovered GC roots for this async state machine object.
<pre>DumpAssembly <assembly address></pre>	<p>Displays information about an assembly.</p> <p>The DumpAssembly command lists multiple modules, if they exist.</p> <p>You can get an assembly address by using the DumpDomain command.</p>
<pre>DumpClass <EEClass address></pre>	<p>Displays information about the <code>EEClass</code> structure associated with a type.</p> <p>The DumpClass command displays static field values but does not display nonstatic field values.</p> <p>Use the DumpMT, DumpObj, Name2EE, or Token2EE command to get an <code>EEClass</code> structure address.</p>
<pre>DumpDomain [<domain address>]</pre>	<p>Enumerates each Assembly object that is loaded within the specified AppDomain object address. When called with no parameters, the DumpDomain command lists all AppDomain objects in a process. Since .NET Core only has one AppDomain, DumpDomain will only return one object.</p>

COMMAND	DESCRIPTION
<pre>DumpHeap [-stat] [-strings] [-short] [-min <size>] [-max <size>] [-thinlock] [-startAtLowerBound] [-mt <MethodTable address>] [-type <partial type name>] [start [end]]</pre>	<p>Displays information about the garbage-collected heap and collection statistics about objects.</p> <p>The DumpHeap command displays a warning if it detects excessive fragmentation in the garbage collector heap.</p> <p>The -stat option restricts the output to the statistical type summary.</p> <p>The -strings option restricts the output to a statistical string value summary.</p> <p>The -short option limits output to just the address of each object. This lets you easily pipe output from the command to another debugger command for automation.</p> <p>The -min option ignores objects that are less than the <code><size></code> parameter, specified in bytes.</p> <p>The -max option ignores objects that are larger than the <code><size></code> parameter, specified in bytes.</p> <p>The -thinlock option reports ThinLocks. For more information, see the SyncBlk command.</p> <p>The <code>-startAtLowerBound</code> option forces the heap walk to begin at the lower bound of a supplied address range. During the planning phase, the heap is often not walkable because objects are being moved. This option forces DumpHeap to begin its walk at the specified lower bound. You must supply the address of a valid object as the lower bound for this option to work. You can display memory at the address of a bad object to manually find the next method table. If the garbage collection is currently in a call to <code>memcpy</code>, you may also be able to find the address of the next object by adding the size to the start address, which is supplied as a parameter.</p> <p>The -mt option lists only those objects that correspond to the specified <code><MethodTable></code> structure.</p> <p>The -type option lists only those objects whose type name is a substring match of the specified string.</p> <p>The <code>start</code> parameter begins listing from the specified address.</p> <p>The <code>end</code> parameter stops listing at the specified address.</p>
<pre>DumpIL <Managed DynamicMethod object> <DynamicMethodDesc pointer> <MethodDesc pointer></pre>	<p>Displays the Microsoft intermediate language (MSIL) that is associated with a managed method.</p> <p>Dynamic MSIL is emitted differently than MSIL that's loaded from an assembly. Dynamic MSIL refers to objects in a managed object array rather than to metadata tokens.</p>

COMMAND	DESCRIPTION
DumpLog [-addr <addressOfStressLog>] [<Filename>]	<p>Writes the contents of an in-memory stress log to the specified file. If you do not specify a name, this command creates a file called StressLog.txt in the current directory.</p> <p>The in-memory stress log helps you diagnose stress failures without using locks or I/O. To enable the stress log, set the following registry keys under HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\.NETFramework:</p> <p>(DWORD) StressLog = 1 (DWORD) LogFacility = 0xffffffff (DWORD) StressLogSize = 65536</p> <p>The optional <code>-addr</code> option lets you specify a stress log other than the default log.</p>
DumpMD <MethodDesc address>	<p>Displays information about a <code>MethodDesc</code> structure at the specified address.</p> <p>You can use the IP2MD command to get the <code>MethodDesc</code> structure address from a managed function.</p>
DumpMT [-MD] <MethodTable address>	<p>Displays information about a method table at the specified address. Specifying the <code>-MD</code> option displays a list of all methods defined with the object.</p> <p>Each managed object contains a method table pointer.</p>
DumpModule [-mt] <Module address>	<p>Displays information about a module at the specified address. The <code>-mt</code> option displays the types defined in a module and the types referenced by the module</p> <p>You can use the DumpDomain or DumpAssembly command to retrieve a module's address.</p>
DumpObj [-nofields] <object address> -or- DO <object address>	<p>Displays information about an object at the specified address. The DumpObj command displays the fields, the <code>EEClass</code> structure information, the method table, and the size of the object.</p> <p>You can use the DumpStackObjects command to retrieve an object's address.</p> <p>You can run the DumpObj command on fields of type <code>CLASS</code> because they are also objects.</p> <p>The <code>-nofields</code> option prevents fields of the object being displayed, it is useful for objects like String.</p>
DumpRuntimeTypes	<p>Displays the runtime type objects in the garbage collector heap and lists their associated type names and method tables.</p>

COMMAND	DESCRIPTION
DumpStack [-EE] [-n] [top <i>stack</i> [bottom <i>stack</i>]]	<p>Displays a stack trace.</p> <p>The -EE option causes the DumpStack command to display only managed functions. Use the top and bottom parameters to limit the stack frames displayed on x86 platforms.</p> <p>The -n option disables the display of source file names and line numbers. If the debugger has the option SYMOPT_LOAD_LINES specified, SOS will look up the symbols for every managed frame and if successful will display the corresponding source file name and line number. The -n (No line numbers) parameter can be specified to disable this behavior.</p>
DumpSig < <i>sigaddr</i> > < <i>moduleaddr</i> >	Displays information about a Sig structure at the specified address.
DumpSigElem < <i>sigaddr</i> > < <i>moduleaddr</i> >	Displays a single element of a signature object. In most cases, you should use DumpSig to look at individual signature objects. However, if a signature has been corrupted in some way, you can use DumpSigElem to read the valid portions of it.
DumpStackObjects [-verify] [top <i>stack</i> [bottom <i>stack</i>]] -or- DSO [-verify] [top <i>stack</i> [bottom <i>stack</i>]]	<p>Displays all managed objects found within the bounds of the current stack.</p> <p>The -verify option validates each non-static CLASS field of an object field.</p> <p>Use the DumpStackObject command with stack tracing commands such as K (windbg) or bt (lldb) along with the clrstack command to determine the values of local variables and parameters.</p>
DumpVC < <i>MethodTable address</i> > < <i>Address</i> >	<p>Displays information about the fields of a value class at the specified address.</p> <p>The MethodTable parameter allows the DumpVC command to correctly interpret fields. Value classes do not have a method table as their first field.</p>
EEHeap [-gc] [-loader]	<p>Displays information about process memory consumed by internal runtime data structures.</p> <p>The -gc and -loader options limit the output of this command to garbage collector or loader data structures.</p> <p>The information for the garbage collector lists the ranges of each segment in the managed heap. If the pointer falls within a segment range given by -gc, the pointer is an object pointer.</p>

COMMAND	DESCRIPTION
EEStack [-short] [-EE]	<p>Runs the DumpStack command on all threads in the process.</p> <p>The -EE option is passed directly to the DumpStack command. The -short parameter limits the output to the following kinds of threads:</p> <ul style="list-style-type: none"> Threads that have taken a lock. Threads that have been stalled in order to allow a garbage collection. Threads that are currently in managed code.
EHInfo [<MethodDesc address>] [<Code address>]	<p>Displays the exception handling blocks in a specified method. This command displays the code addresses and offsets for the clause block (the <code>try</code> block) and the handler block (the <code>catch</code> block).</p>
FAQ	<p>Displays frequently asked questions. Not supported in <code>dotnet-dump</code>.</p>
FinalizeQueue [-detail] [-allReady] [-short]	<p>Displays all objects registered for finalization.</p> <p>The -detail option displays extra information about any <code>SyncBlocks</code> that need to be cleaned up, and any <code>RuntimeCallableWrappers</code> (RCWs) that await cleanup. Both of these data structures are cached and cleaned up by the finalizer thread when it runs.</p> <p>The -allReady option displays all objects that are ready for finalization, regardless of whether they are already marked by the garbage collection as such, or will be marked by the next garbage collection. The objects that are in the "ready for finalization" list are finalizable objects that are no longer rooted. This option can be very expensive, because it verifies whether all the objects in the finalizable queues are still rooted.</p> <p>The -short option limits the output to the address of each object. If it is used in conjunction with -allReady, it enumerates all objects that have a finalizer that are no longer rooted. If it is used independently, it lists all objects in the finalizable and "ready for finalization" queues.</p>
FindAppDomain <Object address>	<p>Determines the application domain of an object at the specified address.</p>
FindRoots -gen <N> -gen any <object address>	<p>Causes the debugger to break in the debugger on the next collection of the specified generation. The effect is reset as soon as the break occurs. To break on the next collection, you have to reissue the command. The <code><object address></code> form of this command is used after the break caused by the -gen or -gen any has occurred. At that time, the debugger is in the right state for FindRoots to identify roots for objects from the current condemned generations. Only supported on Windows.</p>

COMMAND	DESCRIPTION
GCHandle s [-perdomain]	<p>Displays statistics about garbage collector handles in the process.</p> <p>The -perdomain option arranges the statistics by application domain.</p> <p>Use the GCHandles command to find memory leaks caused by garbage collector handle leaks. For example, a memory leak occurs when code retains a large array because a strong garbage collector handle still points to it, and the handle is discarded without freeing it.</p> <p>Only supported on Windows.</p>
GCHandleLeak s	<p>Searches memory for any references to strong and pinned garbage collector handles in the process and displays the results. If a handle is found, the GCHandleLeaks command displays the address of the reference. If a handle is not found in memory, this command displays a notification. Only supported on Windows.</p>
GCInfo <MethodDesc address> <Code address>	<p>Displays data that indicates when registers or stack locations contain managed objects. If a garbage collection occurs, the collector must know the locations of references to objects so it can update them with new object pointer values.</p>
GCRoot [-nostacks] [-all] <Object address>	<p>Displays information about references (or roots) to an object at the specified address.</p> <p>The GCRoot command examines the entire managed heap and the handle table for handles within other objects and handles on the stack. Each stack is then searched for pointers to objects, and the finalizer queue is also searched.</p> <p>This command does not determine whether a stack root is valid or is discarded. Use the clrstack and U commands to disassemble the frame that the local or argument value belongs to in order to determine if the stack root is still in use.</p> <p>The -nostacks option restricts the search to garbage collector handles and reachable objects.</p> <p>The -all option forces all roots to be displayed instead of just the unique roots.</p>
GCWhere <object address>	<p>Displays the location and size in the garbage collection heap of the argument passed in. When the argument lies in the managed heap but is not a valid object address, the size is displayed as 0 (zero).</p>
Help (soshelp) [<command>] [faq]	<p>Displays all available commands when no parameter is specified, or displays detailed help information about the specified command.</p> <p>The faq parameter displays answers to frequently asked questions.</p>

COMMAND	DESCRIPTION
HeapStat [-inclUnrooted -iu]	Displays the generation sizes for each heap and the total free space in each generation on each heap. If the -inclUnrooted option is specified, the report includes information about the managed objects from the garbage collection heap that is no longer rooted. Only supported on Windows.
HistClear	Releases any resources used by the family of Hist commands. Generally, you do not have to explicitly call HistClear , because each HistInit cleans up the previous resources.
HistInit	Initializes the SOS structures from the stress log saved in the debugger.
HistObj <obj_address>	Examines all stress log relocation records and displays the chain of garbage collection relocations that may have led to the address passed in as an argument.
HistObjFind <obj_address>	Displays all the log entries that reference an object at the specified address.
HistRoot <root>	Displays information related to both promotions and relocations of the specified root. The root value can be used to track the movement of an object through the garbage collections.
IP2MD (ip2md) <Code address>	Displays the MethodDesc structure at the specified address in code that has been JIT-compiled.
ListNearObj (lno) <obj_address>	Displays the objects preceding and following the specified address. The command looks for the address in the garbage collection heap that looks like a valid beginning of a managed object (based on a valid method table) and the object following the argument address. Only supported on Windows.
MinidumpMode [0] [1]	Prevents running unsafe commands when using a minidump. Pass 0 to disable this feature or 1 to enable this feature. By default, the MinidumpMode value is set to 0. Minidumps created with the .dump /m command or .dump command have limited CLR-specific data and allow you to run only a subset of SOS commands correctly. Some commands may fail with unexpected errors because required areas of memory are not mapped or are only partially mapped. This option protects you from running unsafe commands against minidumps. Only supported with Windbg.

COMMAND	DESCRIPTION
<p>Name2EE (<code>name2ee</code>) <module name> <type or method name></p> <p>-or-</p> <p>Name2EE <module name>!<type or method name></p>	<p>Displays the <code>MethodTable</code> structure and <code>EEClass</code> structure for the specified type or method in the specified module.</p> <p>The specified module must be loaded in the process.</p> <p>To get the proper type name, browse the module by using the Ildasm.exe (IL Disassembler). You can also pass <code>*</code> as the module name parameter to search all loaded managed modules. The <i>module name</i> parameter can also be the debugger's name for a module, such as <code>mscorlib</code> or <code>image00400000</code>.</p> <p>This command supports the Windows debugger syntax of <<code>module</code>> !<<code>type</code>>. The type must be fully qualified.</p>
<p>ObjSize [<Object address>] [-aggregate] [-stat]</p>	<p>Displays the size of the specified object. If you do not specify any parameters, the ObjSize command displays the size of all objects found on managed threads, displays all garbage collector handles in the process, and totals the size of any objects pointed to by those handles. The ObjSize command includes the size of all child objects in addition to the parent.</p> <p>The -aggregate option can be used in conjunction with the -stat argument to get a detailed view of the types that are still rooted. By using <code>!dumpheap -stat</code> and <code>!objsize -aggregate -stat</code>, you can determine which objects are no longer rooted and diagnose various memory issues.</p> <p>Only supported on Windows.</p>
<p>PrintException [-nested] [-lines] [<Exception object address>]</p> <p>-or-</p> <p>PE [-nested] [<Exception object address>]</p>	<p>Displays and formats fields of any object derived from the Exception class at the specified address. If you do not specify an address, the PrintException command displays the last exception thrown on the current thread.</p> <p>The -nested option displays details about nested exception objects.</p> <p>The -lines option displays source information, if available.</p> <p>You can use this command to format and view the <code>_stackTrace</code> field, which is a binary array.</p>
<p>ProcInfo [-env] [-time] [-mem]</p>	<p>Displays environment variables for the process, kernel CPU time, and memory usage statistics. Only supported with Windbg.</p>
<p>RCWCleanupList <RCWCleanupList address></p>	<p>Displays the list of runtime callable wrappers at the specified address that are awaiting cleanup. Only supported with Windbg.</p>
<p>SaveModule <Base address> <Filename></p>	<p>Writes an image, which is loaded in memory at the specified address, to the specified file. Only supported with Windbg.</p>

COMMAND	DESCRIPTION
SetHostRuntime [<runtime-directory>]	<p>This command sets the path to the .NET Core runtime to use to host the managed code that runs as part of SOS in the debugger (lldb). The runtime needs to be at least version 2.1.0 or greater. If there are spaces in directory, it needs to be single-quoted (').</p> <p>Normally, SOS attempts to find an installed .NET Core runtime to run its managed code automatically but this command is available if it fails. The default is to use the same runtime (libcoreclr) being debugged. Use this command if the default runtime being debugged isn't working enough to run the SOS code or if the version is less than 2.1.0.</p> <p>If you received the following error message when running a SOS command, use this command to set the path to 2.1.0 or greater .NET Core runtime.</p> <pre>(lldb) clrstack Error: Fail to initialize CoreCLR 80004005 ClrStack failed</pre> <pre>(lldb) sethostruntime /usr/share/dotnet/shared/Microsoft.NETCore.App/2.1.6</pre> <p>You can use the "dotnet --info" in a command shell to find the path of an installed .NET Core runtime.</p>
SetSymbolServer [-ms] [-disable] [-log] [-loadsymbols] [-cache <cache-path>] [-directory <search-directory>] [-sympath <windows-symbol-path>] [<symbol-server-URL>]	<p>Enables the symbol server downloading support.</p> <p>The -ms option enables downloading from the public Microsoft symbol server.</p> <p>The -disable option turns on the symbol download support.</p> <p>The -cache <cache-path> option specifies a symbol cache directory. The default is \$HOME/.dotnet/symbolcache if not specified.</p> <p>The -directory option adds a path to search for symbols. Can be more than one.</p> <p>The -sympath option adds server, cache, and directory paths in the Windows symbol path format.</p> <p>The -log option enables symbol download logging.</p> <p>The -loadsymbols option attempts to download the native .NET Core symbols for the runtime. Supported on lldb and dotnet-dump.</p>
SOSFlush	Flushes an internal SOS cache.
SOSStatus [-reset]	Displays internal SOS status or reset the internal cached state.

COMMAND	DESCRIPTION
StopOnException [-derived] [-create -create2] <Exception> <Pseudo-register number>	<p>Causes the debugger to stop when the specified exception is thrown, but to continue running when other exceptions are thrown.</p> <p>The -derived option catches the specified exception and every exception that derives from the specified exception.</p> <p>Only supported with Windbg.</p>
SyncBlk [-all <syncblk number>]	<p>Displays the specified <code>SyncBlock</code> structure or all <code>SyncBlock</code> structures. If you do not pass any arguments, the SyncBlk command displays the <code>SyncBlock</code> structure corresponding to objects that are owned by a thread.</p> <p>A <code>SyncBlock</code> structure is a container for extra information that does not need to be created for every object. It can hold COM interop data, hash codes, and locking information for thread-safe operations.</p>
ThreadPool	<p>Displays information about the managed thread pool, including the number of work requests in the queue, the number of completion port threads, and the number of timers.</p>
Threads (clrthreads) [-live] [-special]	<p>Displays all managed threads in the process.</p> <p>The Threads command displays the debugger shorthand ID, the CLR thread ID, and the operating system thread ID. Additionally, the Threads command displays a Domain column that indicates the application domain in which a thread is executing, an APT column that displays the COM apartment mode, and an Exception column that displays the last exception thrown in the thread.</p> <p>The -live option displays threads associated with a live thread.</p> <p>The -special option displays all special threads created by the CLR. Special threads include garbage collection threads (in concurrent and server garbage collection), debugger helper threads, finalizer threads, AppDomain unload threads, and thread pool timer threads.</p>
ThreadState < State value field >	<p>Displays the state of the thread. The <code>value</code> parameter is the value of the <code>state</code> field in the Threads report output.</p>
Token2EE <module name> <token>	<p>Turns the specified metadata token in the specified module into a <code>MethodTable</code> structure or <code>MethodDesc</code> structure.</p> <p>You can pass <code>*</code> for the module name parameter to find what that token maps to in every loaded managed module. You can also pass the debugger's name for a module, such as <code>mscorlib</code> or <code>image00400000</code>.</p>

COMMAND	DESCRIPTION
U [-gcinfo] [-ehinfo] [-n] <MethodDesc address> <Code address>	<p>Displays an annotated disassembly of a managed method specified either by a <code>MethodDesc</code> structure pointer for the method or by a code address within the method body. The U command displays the entire method from start to finish, with annotations that convert metadata tokens to names.</p> <p>The <code>-gcinfo</code> option causes the U command to display the <code>GCInfo</code> structure for the method.</p> <p>The <code>-ehinfo</code> option displays exception information for the method. You can also obtain this information with the EHInfo command.</p> <p>The <code>-n</code> option disables the display of source file names and line numbers. If the debugger has the option <code>SYMOPT_LOAD_LINES</code> specified, SOS looks up the symbols for every managed frame and, if successful, displays the corresponding source file name and line number. You can specify the <code>-n</code> option to disable this behavior.</p>
VerifyHeap	<p>Checks the garbage collector heap for signs of corruption and displays any errors found.</p> <p>Heap corruptions can be caused by platform invoke calls that are constructed incorrectly.</p>
VerifyObj <object address>	Checks the object that is passed as an argument for signs of corruption. Only supported on Windows.
VMMap	Traverses the virtual address space and displays the type of protection applied to each region. Only supported with Windbg.
VMStat	Provides a summary view of the virtual address space, ordered by each type of protection applied to that memory (free, reserved, committed, private, mapped, image). The TOTAL column displays the result of the AVERAGE column multiplied by the BLK COUNT column. Only supported with Windbg.

Dotnet-Dump

For a list of available SOS commands with `dotnet-dump analyze`, see [dotnet-dump](#).

Windows Debugger

You can also use the SOS Debugging Extension by loading it into the [WinDbg/dbg debugger](#) and executing commands within the Windows debugger. SOS commands can be used on live processes or dumps.

Windbg should load the SOS extension automatically whenever the process being debugged contains the .NET Core runtime (coreclr.dll or libcoreclr.so).

LLDB Debugger

For instructions on configuring SOS for LLDB, see [dotnet-sos](#). SOS commands can be used on live processes or dumps.

By default you can reach all the SOS commands by entering: `sos [command_name]`. However, the common commands have been aliased so that you don't need the `sos` prefix:

COMMAND	FUNCTION
<code>bpmd</code>	Creates a breakpoint at the specified managed method in the specified module.
<code>clrstack</code>	Provides a stack trace of managed code only.
<code>clrthreads</code>	List the managed threads that are running.
<code>clru</code>	Displays an annotated disassembly of a managed method.
<code>dso</code>	Displays all managed objects found within the bounds of the current stack.
<code>dumpasync</code>	Displays info about async state machines on the garbage-collected heap.
<code>dumpclass</code>	Displays information about the <code>EEClass</code> structure at the specified address.
<code>dumpdomain</code>	Displays information all the AppDomains and all assemblies within the specified domain.
<code>dumpheap</code>	Displays info about the garbage-collected heap and collection statistics about objects.
<code>dumpil</code>	Displays the Microsoft intermediate language (MSIL) that is associated with a managed method.
<code>dumplog</code>	Writes the contents of an in-memory stress log to the specified file.
<code>dumpmd</code>	Displays information about the <code>MethodDesc</code> structure at the specified address.
<code>dumpmodule</code>	Displays information about the module at the specified address.
<code>dumpmt</code>	Displays information about the method table at the specified address.
<code>dumpobj</code>	Displays info the object at the specified address.
<code>dumpstack</code>	Displays a native and managed stack trace.
<code>eeheap</code>	Displays info about process memory consumed by internal runtime data structures.
<code>eestack</code>	Runs <code>dumpstack</code> on all threads in the process.
<code>gcroot</code>	Displays info about references (or roots) to the object at the specified address.
<code>histclear</code>	Releases any resources used by the family of Hist commands.

COMMAND	FUNCTION
<code>histinit</code>	Initializes the SOS structures from the stress log saved in the debugger.
<code>histobj</code>	Examines all stress log relocation records and displays the chain of garbage collection relocations that may have led to the address passed in as an argument.
<code>histobjfind</code>	Displays all the log entries that reference the object at the specified address.
<code>histroot</code>	Displays information related to both promotions and relocations of the specified root.
<code>ip2md</code>	Displays the <code>MethodDesc</code> structure at the specified address in code that has been JIT-compiled.
<code>loadsymbols</code>	Load the .NET Core native module symbols.
<code>name2ee</code>	Displays the <code>MethodTable</code> and <code>EEClass</code> structures for the specified type or method in the specified module.
<code>pe</code>	Displays and formats fields of any object derived from the <code>Exception</code> class at the specified address.
<code>setclrpath</code>	Sets the path to load coreclr dac/dbi files. <code>setclrpath <path></code>
<code>sethostruntime</code>	Sets or displays the .NET Core runtime directory to use to run managed code in SOS.
<code>setsymbolserver</code>	Enables the symbol server support.
<code>setsostid</code>	Sets the current OS tid/thread index instead of using the one lldb provides. <code>setsostid <tid> <index></code>
<code>sos</code>	Various coreclr debugging commands. For more information, see 'soshelp'. <code>sos <command-name> <args></code>
<code>soshelp</code>	Displays all available commands when no parameter is specified, or displays detailed help information about the specified command: <code>soshelp <command></code>
<code>syncblk</code>	Displays the SyncBlock holder info.

Windbg/cdb example usage

COMMAND	DESCRIPTION
<code>!dumparray -start 2 -length 5 -detail 00ad28d0</code>	Displays the contents of an array at the address <code>00ad28d0</code> . The display starts from the second element and continues for five elements.

COMMAND	DESCRIPTION
<code>!dumpassembly 1ca248</code>	Displays the contents of an assembly at the address <code>1ca248</code> .
<code>!dumpheap</code>	Displays information about the garbage collector heap.
<code>!DumpLog</code>	Writes the contents of the in-memory stress log to a (default) file called <code>StressLog.txt</code> in the current directory.
<code>!dumpmd 902f40</code>	Displays the <code>MethodDesc</code> structure at the address <code>902f40</code> .
<code>!dumpmodule 1caa50</code>	Displays information about a module at the address <code>1caa50</code> .
<code>!DumpObj a79d40</code>	Displays information about an object at the address <code>a79d40</code> .
<code>!DumpVC 0090320c 00a79d9c</code>	Displays the fields of a value class at the address <code>00a79d9c</code> using the method table at the address <code>0090320c</code> .
<code>!eeheap -gc</code>	Displays the process memory used by the garbage collector.
<code>!finalizequeue</code>	Displays all objects scheduled for finalization.
<code>!findappdomain 00a79d98</code>	Determines the application domain of an object at the address <code>00a79d98</code> .
<code>!gcinfo 5b68dbb8</code>	Displays all garbage collector handles in the current process.
<code>!name2ee unittest.exe MainClass.Main</code>	Displays the <code>MethodTable</code> and <code>EEClass</code> structures for the <code>Main</code> method in the class <code>MainClass</code> in the module <code>unittest.exe</code> .
<code>!token2ee unittest.exe 02000003</code>	Displays information about the metadata token at the address <code>02000003</code> in the module <code>unittest.exe</code> .

LLDB example usage

COMMAND	DESCRIPTION
<code>sos DumpArray -start 2 -length 5 -detail 00ad28d0</code>	Displays the contents of an array at the address <code>00ad28d0</code> . The display starts from the second element and continues for five elements.
<code>sos DumpAssembly 1ca248</code>	Displays the contents of an assembly at the address <code>1ca248</code> .
<code>dumpheap</code>	Displays information about the garbage collector heap.
<code>dumplog</code>	Writes the contents of the in-memory stress log to a (default) file called <code>StressLog.txt</code> in the current directory.

COMMAND	DESCRIPTION
<code>dumpmd 902f40</code>	Displays the <code>MethodDesc</code> structure at the address <code>902f40</code> .
<code>dumpmodule 1caa50</code>	Displays information about a module at the address <code>1caa50</code> .
<code>dumpobj a79d40</code>	Displays information about an object at the address <code>a79d40</code> .
<code>sos DumpVC 0090320c 00a79d9c</code>	Displays the fields of a value class at the address <code>00a79d9c</code> using the method table at the address <code>0090320c</code> .
<code>eeheap -gc</code>	Displays the process memory used by the garbage collector.
<code>sos FindAppDomain 00a79d98</code>	Determines the application domain of an object at the address <code>00a79d98</code> .
<code>sos GCInfo 5b68dbb8</code>	Displays all garbage collector handles in the current process.
<code>name2ee unittest.exe MainClass.Main</code>	Displays the <code>MethodTable</code> and <code>EEClass</code> structures for the <code>Main</code> method in the class <code>MainClass</code> in the module <code>unittest.exe</code> .
<code>sos Token2EE unittest.exe 02000003</code>	Displays information about the metadata token at the address <code>02000003</code> in the module <code>unittest.exe</code> .
<code>clrthreads</code>	Displays the managed threads.

See also

- [An introduction to dumps in .NET](#)
- [Learn how to debug a memory leak in .NET Core](#)
- [Collecting and analyzing memory dumps blog](#)
- [Dump analysis tool \(dotnet-dump\)](#)
- [SOS Installation Tool \(dotnet-sos\)](#)
- [Heap analysis tool \(dotnet-gcdump\)](#)

.NET logging and tracing

9/20/2022 • 3 minutes to read • [Edit Online](#)

Code can be instrumented to produce a log, which serves as a record of interesting events that occurred while the program was running. To understand the application's behavior, logs are reviewed. Logging and tracing both encapsulate this technique. .NET has accumulated several different logging APIs over its history and this article will help you understand what options are available.

Key distinctions in logging APIs

Structured logging

Logging APIs can either be structured or unstructured:

- Unstructured: Log entries have free-form text content that is intended to be viewed by humans.
- Structured: Log entries have a well defined schema and can be encoded in different binary and textual formats. These logs are designed to be machine translatable and queryable so that both humans and automated systems can work with them easily.

APIs that support structured logging are preferable for non-trivial usage. They offer more functionality, flexibility, and performance with little difference in usability.

Configuration

For simple use cases, you might want to use APIs that directly write messages to the console or a file. But most software projects will find it useful to configure which log events get recorded and how they are persisted. For example, when running in a local dev environment, you might want to output plain text to the console for easy readability. Then when the application is deployed to a production environment, you might switch to have the logs stored in a dedicated database or a set of rolling files. APIs with good configuration options will make these transitions easy, whereas less configurable options would require updating the instrumentation code everywhere to make changes.

Sinks

Most logging APIs allow log messages to be sent to different destinations called sinks. Some APIs have a large number of pre-made sinks, whereas others only have a few. If no pre-made sink exists, there's usually an extensibility API that will let you author a custom sink, although this requires writing a bit more code.

.NET logging APIs

`ILogger`

For most cases, whether adding logging to an existing project or creating a new project, the `ILogger` interface is a good default choice. `ILogger` supports fast structured logging, flexible configuration, and a collection of [common sinks](#). Additionally, the `ILogger` interface can also serve as a facade over many [third party logging implementations](#) that offer more functionality and extensibility.

`EventSource`

`EventSource` is an older high performance structured logging API. It was originally designed to integrate well with [Event Tracing for Windows \(ETW\)](#), but was later extended to support [EventPipe](#) cross-platform tracing and [EventListener](#) for custom sinks. In comparison to `ILogger`, `EventSource` has relatively few pre-made logging sinks and there is no built-in support to configure via separate configuration files. `EventSource` is excellent if you want tighter control over [ETW](#) or [EventPipe](#) integration, but for general purpose logging, `ILogger` is more

flexible and easier to use.

Trace

`System.Diagnostics.Trace` and `System.Diagnostics.Debug` are .NET's oldest logging APIs. These classes have flexible configuration APIs and a large ecosystem of sinks, but only support unstructured logging. On .NET Framework they can be configured via an app.config file, but in .NET Core, there's no built-in, file-based configuration mechanism. The .NET team continues to support these APIs for backward-compatibility purposes, but no new functionality will be added. These APIs are a fine choice for applications that are already using them. For newer apps that haven't already committed to a logging API, `ILogger` may offer better functionality.

Specialized logging APIs

Console

The `System.Console` class has the `Write` and `WriteLine` methods that can be used in simple logging scenarios. These APIs are very easy to get started with but the solution won't be as flexible as a general purpose logging API. Console only allows unstructured logging and there is no configuration support to select which log messages are enabled or to retarget to a different sink. Using the `ILogger` or `Trace` APIs with a console sink doesn't take much additional effort and keeps the logging configurable.

DiagnosticSource

`System.Diagnostics.DiagnosticSource` is intended for logging where the log messages will be analyzed synchronously in-process rather than serialized to any storage. This allows the source and listener to exchange arbitrary .NET objects as messages, whereas most logging APIs require the log event to be serializable. This technique can also be extremely fast, handling log events in tens of nanoseconds if the listener is implemented efficiently. Tools that use these APIs often act more like in-process profilers, though the API doesn't impose any constraint here.

EventLog

`System.Diagnostics.EventLog` is a Windows only API that writes messages to the Windows EventLog. In many cases using `ILogger` with an optional EventLog sink when running on Windows may give similar functionality without coupling the app tightly to the Windows OS.

Well-known event providers in .NET

9/20/2022 • 4 minutes to read • [Edit Online](#)

The .NET runtime and libraries write diagnostic events through a number of different event providers. Depending on your diagnostic needs, you can choose the appropriate providers to enable. This article describes some of the most commonly used event providers in the .NET runtime and libraries.

CoreCLR

"Microsoft-Windows-DotNetRuntime" provider

This provider emits various events from the .NET runtime, including GC, loader, JIT, exception, and other events. Read more about each event from this provider in [Runtime Provider Events List](#).

"Microsoft-DotNetCore-SampleProfiler" provider

This provider is a .NET runtime event provider that is used for CPU sampling for managed callstacks. When enabled, it captures a snapshot of each thread's managed callstack every 10 milliseconds. To enable this capture, you must specify an [EventLevel](#) of [Informational](#) or higher.

Framework libraries

"Microsoft-Extensions-DependencyInjection" provider

This provider logs information from [DependencyInjection](#). The following table shows events logged by the [Microsoft-Extensions-DependencyInjection](#) provider:

EVENT NAME	KEYWORD	LEVEL	DESCRIPTION
<code>CallSiteBuilt</code>		Verbose (5)	A call site has been built.
<code>ServiceResolved</code>		Verbose (5)	A service has been resolved.
<code>ExpressionTreeGenerated</code>		Verbose (5)	An expression tree has been generated.
<code>DynamicMethodBuilt</code>		Verbose (5)	A DynamicMethod has been built.
<code>ScopeDisposed</code>		Verbose (5)	A scope has been disposed.
<code>ServiceRealizationFailed</code>		Verbose (5)	A service realization has failed.
<code>ServiceProviderBuilt</code>	<code>ServiceProviderInitialized(0)</code>	Verbose (5)	A ServiceProvider has been built.
<code>ServiceProviderDescriptors</code>	<code>ServiceProviderInitialized(0)</code>	Verbose (5)	A list of ServiceDescriptor that has been used during the ServiceProvider build.

"System.Buffers.ArrayPoolEventSource" provider

This provider logs information from the [ArrayPool](#). The following table shows the events logged by

ArrayPoolEventSource :

EVENT NAME	LEVEL	DESCRIPTION
BufferRented	Verbose (5)	A buffer is successfully rented.
BufferAllocated	Informational (4)	A buffer is allocated by the pool.
BufferReturned	Verbose (5)	A buffer is returned to the pool.
BufferTrimmed	Informational (4)	A buffer is attempted to be freed due to memory pressure or inactivity.
BufferTrimPoll	Informational (4)	A check is being made to trim buffers.

"System.Net.Http" provider

This provider logs information from the HTTP stack. The following table shows the events logged by `System.Net.Http` provider:

EVENT NAME	LEVEL	DESCRIPTION
RequestStart	Informational (4)	An HTTP request has started.
RequestStop	Informational (4)	An HTTP request has finished.
RequestFailed	Error (2)	An HTTP request has failed.
ConnectionEstablished	Informational (4)	An HTTP connection has been established.
ConnectionClosed	Informational (4)	An HTTP connection has been closed.
RequestLeftQueue	Informational (4)	An HTTP request has left the request queue.
RequestHeadersStart	Informational (4)	An HTTP request for header has started.
RequestHeaderStop	Informational (4)	An HTTP request for header has finished.
RequestContentStart	Informational (4)	An HTTP request for content has started.
RequestContentStop	Informational (4)	An HTTP request for content has finished.
ResponseHeadersStart	Informational (4)	An HTTP response for header has started.
ResponseHeaderStop	Informational (4)	An HTTP response for header has finished.

EVENT NAME	LEVEL	DESCRIPTION
ResponseContentStart	Informational (4)	An HTTP response for content has started.
ResponseContentStop	Informational (4)	An HTTP response for content has finished.

"System.Net.NameResolution" provider

This provider logs information related to domain name resolution. The following table shows the events logged by `System.Net.NameResolution` :

EVENT NAME	LEVEL	DESCRIPTION
<code>ResolutionStart</code>	Informational (4)	A domain name resolution has started.
<code>ResolutionStop</code>	Informational (4)	A domain name resolution has finished.
<code>ResolutionFailed</code>	Informational (4)	A domain name resolution has failed.

"System.Net.Sockets" provider

This provider logs information from `Socket`. The following table shows the events logged by `System.Net.Sockets` provider:

EVENT NAME	LEVEL	DESCRIPTION
<code>ConnectStart</code>	Informational (4)	An attempt to start a socket connection has started.
<code>ConnectStop</code>	Informational (4)	An attempt to start a socket connection has finished.
<code>ConnectFailed</code>	Informational (4)	An attempt to start a socket connection has failed.
<code>AcceptStart</code>	Informational (4)	An attempt to accept a socket connection has started.
<code>AcceptStop</code>	Informational (4)	An attempt to accept a socket connection has finished.
<code>AcceptFailed</code>	Informational (4)	An attempt to accept a socket connection has failed.

"System.Threading.Tasks.TplEventSource" provider

This provider logs information on the [Task Parallel Library](#), such as Task scheduler events. The following table shows the events logged by `TplEventSource` :

EVENT NAME	KEYWORD	LEVEL	DESCRIPTION

Event Name	Keyword	Level	Description
<code>TaskScheduled</code>	<code>TaskTransfer (0x1)</code> <code>Tasks (0x2)</code>	Informational (4)	A Task is queued to the Task scheduler.
<code>TaskStarted</code>	<code>Tasks (0x2)</code>	Informational (4)	A Task has started executing.
<code>TaskCompleted</code>	<code>TaskStops (0x40)</code>	Informational (4)	A Task has finished executing.
<code>TaskWaitBegin</code>	<code>TaskTransfer (0x1)</code> <code>TaskWait (0x2)</code>	Informational (4)	Fired when an implicit or an explicit wait on a Task completion has started.
<code>TaskWaitEnd</code>	<code>Tasks (0x2)</code>	Verbose (5)	Fired when the wait for a Task completion returns.
<code>TaskWaitContinuationStarted</code>	<code>Tasks (0x2)</code>	Verbose (5)	Fired when the work (method) associated with a <code>TaskWaitEnd</code> is started.
<code>TaskWaitContinuationComplete</code>	<code>TaskStops (0x40)</code>	Verbose (5)	Fired when the work (method) associated with a <code>TaskWaitEnd</code> is completed.
<code>AwaitTaskContinuationSchedul</code>	<code>TaskTransfer (0x1)</code> <code>Tasks (0x2)</code>	Informational (4)	Fired when the an asynchronous continuation for a Task is scheduled.

ASP.NET Core

ASP.NET Core also provides several events to help you diagnose issues in the ASP.NET Core stack.

To learn more about the events in ASP.NET Core and how to consume them, see [Logging in .NET Core and ASP.NET Core](#).

Entity Framework core

EF Core also provides events to help you diagnose issues in EF Core.

To learn more about the events in EF Core and how to consume them, see [.NET Events in EF Core](#).

EventSource

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 3.1 and later versions ✓ .NET Framework 4.5 and later versions

[System.Diagnostics.Tracing.EventSource](#) is a fast structured logging solution built into the .NET runtime. On .NET Framework EventSource can send events to [Event Tracing for Windows \(ETW\)](#) and [System.Diagnostics.Tracing.EventListener](#). On .NET Core EventSource additionally supports [EventPipe](#), a cross-platform tracing option. Most often developers use EventSource logs for performance analysis, but EventSource can be used for any diagnostic tasks where logs are useful. The .NET runtime is already instrumented with [built-in events](#) and you can log your own custom events.

NOTE

Many technologies that integrate with EventSource use the terms 'Tracing' and 'Traces' instead of 'Logging' and 'Logs'. The meaning is the same here.

- [Getting started](#)
- [Instrumenting code to create events](#)
- [Collecting and viewing event traces](#)
- [Understanding Activity IDs](#)

Getting Started with EventSource

9/20/2022 • 4 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 3.1 and later versions ✓ .NET Framework 4.5 and later versions

This walkthrough shows how to log a new event with [System.Diagnostics.Tracing.EventSource](#), collect events in a trace file, view the trace, and understand basic EventSource concepts.

NOTE

Many technologies that integrate with EventSource use the terms 'Tracing' and 'Traces' instead of 'Logging' and 'Logs'. The meaning is the same here.

Log an event

The goal of EventSource is to allow .NET developers to write code like this to log an event:

```
DemoEventSource.Log.AppStarted("Hello World!", 12);
```

This line of code has a logging object (`DemoEventSource.Log`), a method representing the event to log (`AppStarted`), and optionally some strongly typed event parameters (`HelloWorld!` and `12`). There are no verbosity levels, event IDs, message templates, or anything else that doesn't need to be at the call site. All of this other information about events is written by defining a new class derived from [System.Diagnostics.Tracing.EventSource](#).

Here's a complete minimal example:

```
using System.Diagnostics.Tracing;

namespace EventSourceDemo
{
    public static class Program
    {
        public static void Main(string[] args)
        {
            DemoEventSource.Log.AppStarted("Hello World!", 12);
        }
    }

    [EventSource(Name = "Demo")]
    class DemoEventSource : EventSource
    {
        public static DemoEventSource Log { get; } = new DemoEventSource();

        [Event(1)]
        public void AppStarted(string message, int favoriteNumber) => WriteEvent(1, message,
favoriteNumber);
    }
}
```

The `DemoEventSource` class declares a method for each type of event that you wish to log. In this case, a single event called 'AppStarted' is defined by the `AppStarted()` method. Each time the code invokes the `AppStarted` method another `AppStarted` event will be recorded in the trace if the event is enabled. This is some of the data

that can be captured with each event:

- Event name - A name that identifies the kind of event that was logged. The event name will be identical to the method name, 'AppStarted' in this case.
- Event ID - A numerical ID that identifies the kind of event that was logged. This serves a similar role to the name but can assist in fast automated log processing. The AppStarted event has an ID of 1, specified in the [EventAttribute](#).
- Source name - The name of the EventSource that contains the event. This is used as a namespace for events. Event names and IDs only need to be unique within the scope of their source. Here the source is named "Demo", specified in the [EventSourceAttribute](#) on the class definition. The source name is also commonly referred to as a provider name.
- Arguments - All the method argument values are serialized.
- Other information - Events can also contain timestamps, thread IDs, processor IDs, [Activity IDs](#), stack traces, and event metadata such as message templates, verbosity levels, and keywords.

For more information and best practices on creating events, see [Instrumenting code to create events](#).

Collect and view a trace file

There's no required configuration in code that describes which events should be enabled, where the logged data should be sent, or what format the data should be stored in. If you run the app now, it won't produce any trace file by default. EventSource uses the [Publish-subscribe pattern](#), which requires subscribers to indicate the events that should be enabled and to control all serialization for the subscribed events. EventSource has integrations for subscribing from [Event Tracing for Windows \(ETW\)](#) and [EventPipe](#) (.NET Core only). Custom subscribers can also be created using the [System.Diagnostics.Tracing.EventListener API](#).

This demo shows an [EventPipe](#) example for .NET Core apps. To learn about more options see [Collecting and viewing event traces](#). [EventPipe](#) is an open and cross-platform tracing technology built into the .NET Core runtime to give .NET developers trace collection tools and a portable compact trace format (*.nettrace files). [dotnet-trace](#) is a command-line tool that collects EventPipe traces.

1. Download and Install [dotnet-trace](#)
2. Build the console app above. This demo assumes the app is named EventSourceDemo.exe and it is in the current directory. At the command-line run:

```
>dotnet-trace collect --providers Demo -- EventSourceDemo.exe
```

This should show output similar to:

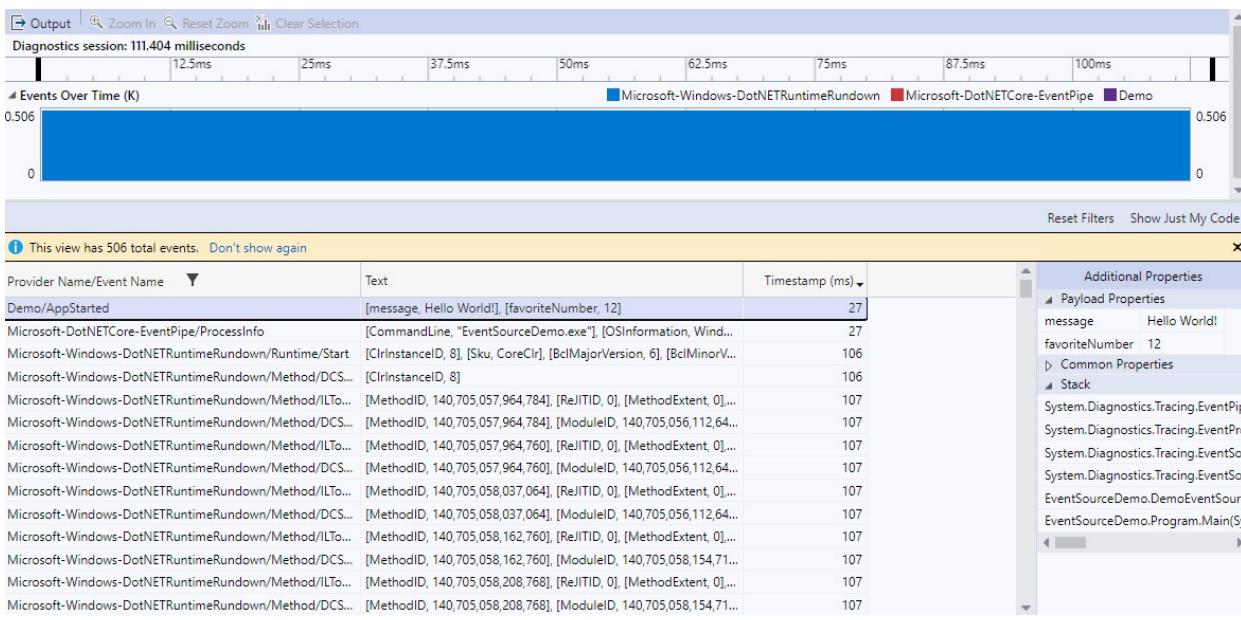
```
Provider Name          Keywords      Level      Enabled By
Demo                  0xFFFFFFFFFFFF  Verbose(5)  --providers

Launching: EventSourceDemo.exe
Process      : E:\temp\EventSourceDemo\bin\Debug\net6.0\EventSourceDemo.exe
Output File   : E:\temp\EventSourceDemo\bin\Debug\net6.0\EventSourceDemo.exe_20220303_001619.nettrace

[00:00:00:00]  Recording trace 0.00    (B)
Press <Enter> or <Ctrl+C> to exit...

Trace completed.
```

This command ran EventSourceDemo.exe with all events in the 'Demo' EventSource enabled and output the trace file `EventSourceDemo.exe_20220303_001619.nettrace`. Opening the file in Visual Studio shows the events that were logged.



In the list view, you can see the first event is the Demo/AppStarted event. The text column has the saved arguments, the timestamp column shows the event occurred 27 ms after logging started and to the right you can see the callstack. The other events are automatically enabled in every trace collected by dotnet-trace though they can be ignored and filtered from view in the UI if they're distracting. Those extra events capture some information about the process and jitted code, which allows Visual Studio to reconstruct the event stack traces.

Learning more about EventSource

- Instrumenting code to create events
- Collecting and viewing event traces

Instrument Code to Create EventSource Events

9/20/2022 • 12 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 3.1 and later versions ✓ .NET Framework 4.5 and later versions

The [Getting Started guide](#) showed you how to create a minimal EventSource and collect events in a trace file. This tutorial goes into more detail about creating events using [System.Diagnostics.Tracing.EventSource](#).

A minimal EventSource

```
[EventSource(Name = "Demo")]
class DemoEventSource : EventSource
{
    public static DemoEventSource Log { get; } = new DemoEventSource();

    [Event(1)]
    public void AppStarted(string message, int favoriteNumber) => WriteEvent(1, message, favoriteNumber);
}
```

The basic structure of a derived EventSource is always the same. In particular:

- The class inherits from [System.Diagnostics.Tracing.EventSource](#)
- For each different type of event you wish to generate, a method needs to be defined. This method should be named using the name of the event being created. If the event has additional data these should be passed using arguments. These event arguments need to be serialized so only [certain types](#) are allowed.
- Each method has a body that calls `WriteEvent` passing it an ID (a numeric value that represents the event) and the arguments of the event method. The ID needs to be unique within the EventSource. The ID is explicitly assigned using the [System.Diagnostics.Tracing.EventAttribute](#)
- EventSources are intended to be singleton instances. Thus it's convenient to define a static variable, by convention called `Log`, that represents this singleton.

Rules for defining event methods

1. Any instance, non-virtual, void returning method defined in an EventSource class is by default an event logging method.
2. Virtual or non-void-returning methods are included only if they're marked with the [System.Diagnostics.Tracing.EventAttribute](#)
3. To mark a qualifying method as non-logging you must decorate it with the [System.Diagnostics.Tracing.NonEventAttribute](#)
4. Event logging methods have event IDs associated with them. This can be done either explicitly by decorating the method with a [System.Diagnostics.Tracing.EventAttribute](#) or implicitly by the ordinal number of the method in the class. For example using implicit numbering the first method in the class has ID 1, the second has ID 2, and so on.
5. Event logging methods must call a [WriteEvent](#), [WriteEventCore](#), [WriteEventWithRelatedActivityId](#) or [WriteEventWithRelatedActivityIdCore](#) overload.
6. The event ID, whether implied or explicit, must match the first argument passed to the `WriteEvent*` API it calls.
7. The number, types and order of arguments passed to the EventSource method must align with how they're passed to the `WriteEvent*` APIs. For `WriteEvent` the arguments follow the Event ID, for

`WriteEventWithRelatedActivityId` the arguments follow the `relatedActivityId`. For the `WriteEvent*`Core methods, the arguments must be serialized manually into the `data` parameter.

8. Event names cannot contain `<` or `>` characters. While user-defined methods also cannot contain these characters, `async` methods will be rewritten by the compiler to contain them. To be sure these generated methods don't become events, mark all non-event methods on an [EventSource](#) with the [NonEventAttribute](#).

Best practices

1. Types that derive from `EventSource` usually don't have intermediate types in the hierarchy or implement interfaces. See [Advanced customizations](#) below for some exceptions where this may be useful.
2. Generally the name of the `EventSource` class is a bad public name for the `EventSource`. Public names, the names that will show up in logging configurations and log viewers, should be globally unique. Thus it's good practice to give your `EventSource` a public name using the [System.Diagnostics.Tracing.EventSourceAttribute](#). The name "Demo" used above is short and unlikely to be unique so not a good choice for production use. A common convention is to use a hierarchical name with `.` or `-` as a separator, such as "MyCompany-Samples-Demo", or the name of the Assembly or namespace for which the `EventSource` provides events. It's not recommended to include "EventSource" as part of the public name.
3. Assign Event IDs explicitly, this way seemingly benign changes to the code in the source class such as rearranging it or adding a method in the middle won't change the event ID associated with each method.
4. When authoring events that represent the start and end of a unit of work, by convention these methods are named with suffixes 'Start' and 'Stop'. For example, "RequestStart" and "RequestStop".
5. Do not specify an explicit value for `EventSourceAttribute`'s `Guid` property, unless you need it for backwards compatibility reasons. The default `Guid` value is derived from the source's name, which allows tools to accept the more human-readable name and derive the same `Guid`.
6. Call [IsEnabled\(\)](#) before performing any resource intensive work related to firing an event, such as computing an expensive event argument that won't be needed if the event is disabled.
7. Attempt to keep `EventSource` object back compatible and version them appropriately. The default version for an event is 0. The version can be changed by setting [EventAttribute.Version](#). Change the version of an event whenever you change the data that is serialized with it. Always add new serialized data to the end of the event declaration, that is, at the end of the list of method parameters. If this isn't possible, create a new event with a new ID to replace the old one.
8. When declaring events methods, specify fixed-size payload data before variably sized data.
9. Do not use strings containing null characters. When generating the manifest for ETW `EventSource` will declare all strings as null terminated, even though it is possible to have a null character in a C# String. If a string contains a null character the entire string will be written to the event payload, but any parser will treat the first null character as the end of the string. If there are payload arguments after the string, the remainder of the string will be parsed instead of the intended value.

Typical event customizations

Setting event verbosity levels

Each event has a verbosity level and event subscribers often enable all events on an `EventSource` up to a certain verbosity level. Events declare their verbosity level using the `Level` property. For example in this `EventSource` below a subscriber that requests events of level `Informational` and lower won't log the `Verbose DebugMessage` event.

```
[EventSource(Name = "MyCompany-Samples-Demo")]
class DemoEventSource : EventSource
{
    public static DemoEventSource Log { get; } = new DemoEventSource();

    [Event(1, Level = EventLevel.Informational)]
    public void AppStarted(string message, int favoriteNumber) => WriteEvent(1, message, favoriteNumber);
    [Event(2, Level = EventLevel.Verbose)]
    public void DebugMessage(string message) => WriteEvent(2, message);
}
```

If the verbosity level of an event is not specified in the `EventAttribute`, then it defaults to `Informational`.

Best practice

Use levels less than `Informational` for relatively rare warnings or errors. When in doubt, stick with the default of `Informational` and use `Verbose` for events that occur more frequently than 1000 events/sec.

Setting event keywords

Some event tracing systems support keywords as an additional filtering mechanism. Unlike verbosity that categorizes events by level of detail, keywords are intended to categorize events based on other criteria such as areas of code functionality or which would be useful for diagnosing certain problems. Keywords are named bit flags and each event can have any combination of keywords applied to it. For example the `EventSource` below defines some events that relate to request processing and other events that relate to startup. If a developer wanted to analyze the performance of startup, they might only enable logging the events marked with the `startup` keyword.

```
[EventSource(Name = "Demo")]
class DemoEventSource : EventSource
{
    public static DemoEventSource Log { get; } = new DemoEventSource();

    [Event(1, Keywords = Keywords.Startup)]
    public void AppStarted(string message, int favoriteNumber) => WriteEvent(1, message, favoriteNumber);
    [Event(2, Keywords = Keywords.Requests)]
    public void RequestStart(int requestId) => WriteEvent(2, requestId);
    [Event(3, Keywords = Keywords.Requests)]
    public void RequestStop(int requestId) => WriteEvent(3, requestId);

    public class Keywords // This is a bitvector
    {
        public const EventKeywords Startup = (EventKeywords)0x0001;
        public const EventKeywords Requests = (EventKeywords)0x0002;
    }
}
```

Keywords must be defined by using a nested class called `Keywords` and each individual keyword is defined by a member typed `public const EventKeywords`.

Best practice

Keywords are more important when distinguishing between high volume events. This allows an event consumer to raise the verbosity to a high level but manage the performance overhead and log size by only enabling narrow subsets of the events. Events that are triggered more than 1,000/sec are good candidates for a unique keyword.

Supported parameter types

`EventSource` requires that all event parameters can be serialized so it only accepts a limited set of types. These are:

- Primitives: bool, byte, sbyte, char, short, ushort, int, uint, long, ulong, float, double, IntPtr, and UIntPtr, Guid decimal, string, DateTime, DateTimeOffset, TimeSpan
- Enums
- Structures attributed with [System.Diagnostics.Tracing.EventDataAttribute](#). Only the public instance properties with serializable types will be serialized.
- Anonymous types where all public properties are serializable types
- Arrays of serializable types
- Nullable<T> where T is a serializable type
- KeyValuePair<T, U> where T and U are both serializable types
- Types that implement IEnumerable<T> for exactly one type T and where T is a serializable type

Troubleshooting

The EventSource class was designed so that it would never throw an Exception by default. This is a useful property, as logging is often treated as optional, and you usually don't want an error writing a log message to cause your application to fail. However, this makes finding any mistake in your EventSource difficult. Here are several techniques that can help troubleshoot:

1. The EventSource constructor has overloads that take [EventSourceSettings](#). Try enabling the ThrowOnEventWriteErrors flag temporarily.
2. The [EventSource.ConstructionException](#) property stores any Exception that was generated when validating the event logging methods. This can reveal various authoring errors.
3. EventSource logs errors using event ID 0, and this error event has a string describing the error.
4. When debugging, that same error string will also be logged using Debug.WriteLine() and show up in the debug output window.
5. EventSource internally throws and then catches exceptions when errors occur. To observe when these exceptions are occurring, enable first chance exceptions in a debugger, or use event tracing with the .NET runtime's [Exception events](#) enabled.

Advanced customizations

Setting OpCodes and Tasks

ETW has concepts of [Tasks and OpCodes](#) which are further mechanisms for tagging and filtering events. You can associate events with specific tasks and opcodes using the [Task](#) and [Opcode](#) properties. Here's an example:

```

[EventSource(Name = "Samples-EventSourceDemos-Customized")]
public sealed class CustomizedEventSource : EventSource
{
    static public CustomizedEventSource Log { get; } = new CustomizedEventSource();

    [Event(1, Task = Tasks.Request, Opcode=EventOpcode.Start)]
    public void RequestStart(int RequestID, string Url)
    {
        WriteEvent(1, RequestID, Url);
    }

    [Event(2, Task = Tasks.Request, Opcode=EventOpcode.Info)]
    public void RequestPhase(int RequestID, string PhaseName)
    {
        WriteEvent(2, RequestID, PhaseName);
    }

    [Event(3, Keywords = Keywords.Requests,
           Task = Tasks.Request, Opcode=EventOpcode.Stop)]
    public void RequestStop(int RequestID)
    {
        WriteEvent(3, RequestID);
    }

    public class Tasks
    {
        public const EventTask Request = (EventTask)0x1;
    }
}

```

You can implicitly create EventTask objects by declaring two event methods with subsequent event IDs that have the naming pattern <EventName>Start and <EventName>Stop. These events must be declared next to each other in the class definition and the <EventName>Start method must come first.

Self-describing (tracelogging) vs. manifest event formats

This concept only matters when subscribing to EventSource from ETW. ETW has two different ways that it can log events, manifest format and self-describing (sometimes called tracelogging) format. Manifest-based EventSource objects generate and log an XML document representing the events defined on the class upon initialization. This requires the EventSource to reflect over itself to generate the provider and event metadata. In the Self-describing format metadata for each event is transmitted inline with the event data rather than up-front. The self-describing approach supports the more flexible [Write](#) methods that can send arbitrary events without having created a pre-defined event logging method. It's also slightly faster at startup because it avoids eager reflection. However the extra metadata that's emitted with each event adds a small performance overhead, which may not be desirable when sending a high volume of events.

To use self-describing event format, construct your EventSource using the EventSource(String) constructor, the EventSource(String, EventSourceSettings) constructor, or by setting the EtwSelfDescribingEventFormat flag on EventSourceSettings.

EventSource types implementing interfaces

An EventSource type may implement an interface in order to integrate seamlessly in various advanced logging systems that use interfaces to define a common logging target. Here's an example of a possible use:

```

public interface IMyLogging
{
    void Error(int errorCode, string msg);
    void Warning(string msg);
}

[EventSource(Name = "Samples-EventSourceDemos-MyComponentLogging")]
public sealed class MyLoggingEventSource : EventSource, IMyLogging
{
    public static MyLoggingEventSource Log { get; } = new MyLoggingEventSource();

    [Event(1)]
    public void Error(int errorCode, string msg)
    { WriteEvent(1, errorCode, msg); }

    [Event(2)]
    public void Warning(string msg)
    { WriteEvent(2, msg); }
}

```

You must specify the `EventAttribute` on the interface methods, otherwise (for compatibility reasons) the method won't be treated as a logging method. Explicit interface method implementation is disallowed in order to prevent naming collisions.

EventSource class hierarchies

In most cases, you'll be able to write types that directly derive from the `EventSource` class. Sometimes however it's useful to define functionality that will be shared by multiple derived `EventSource` types, such as customized `WriteEvent` overloads (see [optimizing performance for high volume events](#) below).

Abstract base classes can be used as long as they don't define any keywords, tasks, opcodes, channels, or events. Here's an example where the `UtilBaseEventSource` class defines an optimized `WriteEvent` overload this is needed by multiple derived `EventSources` in the same component. One of these derived types is illustrated below as `OptimizedEventSource`.

```

public abstract class UtilBaseEventSource : EventSource
{
    protected UtilBaseEventSource()
        : base()
    { }
    protected UtilBaseEventSource(bool throwOnEventWriteErrors)
        : base(throwOnEventWriteErrors)
    { }

    protected unsafe void WriteEvent(int eventId, int arg1, short arg2, long arg3)
    {
        if (IsEnabled())
        {
            EventSource.EventData* descrs = stackalloc EventSource.EventData[2];
            descrs[0].DataPointer = (IntPtr)(&arg1);
            descrs[0].Size = 4;
            descrs[1].DataPointer = (IntPtr)(&arg2);
            descrs[1].Size = 2;
            descrs[2].DataPointer = (IntPtr)(&arg3);
            descrs[2].Size = 8;
            WriteEventCore(eventId, 3, descrs);
        }
    }
}

[EventSource(Name = "OptimizedEventSource")]
public sealed class OptimizedEventSource : UtilBaseEventSource
{
    public static OptimizedEventSource Log { get; } = new OptimizedEventSource();

    [Event(1, Keywords = Keywords.Kwd1, Level = EventLevel.Informational,
          Message = "LogElements called {0}/{1}/{2}.")]
    public void LogElements(int n, short sh, long l)
    {
        WriteEvent(1, n, sh, l); // Calls UtilBaseEventSource.WriteEvent
    }

    #region Keywords / Tasks /Opcodes / Channels
    public static class Keywords
    {
        public const EventKeywords Kwd1 = (EventKeywords)1;
    }
    #endregion
}

```

Optimizing performance for high volume events

The EventSource class has a number of overloads for WriteEvent, including one for variable number of arguments. When none of the other overloads match, the params method is called. Unfortunately, the params overload is relatively expensive. In particular it:

1. Allocates an array to hold the variable arguments.
2. Casts each parameter to an object, which causes allocations for value types.
3. Assigns these objects to the array.
4. Calls the function.
5. Figures out the type of each array element to determine how to serialize it.

This is probably 10 to 20 times as expensive as specialized types. This doesn't matter much for low volume cases, but for high volume events it can be important. There are two important cases for insuring that the params overload isn't used:

1. Ensure that enumerated types are cast to 'int' so that they match one of the fast overloads.

2. Create new fast WriteEvent overloads for high volume payloads.

Here's an example for adding a WriteEvent overload that takes four integer arguments

```
[NonEvent]
public unsafe void WriteEvent(int eventId, int arg1, int arg2,
                             int arg3, int arg4)
{
    EventData* descrs = stackalloc EventProvider.EventData[4];

    descrs[0].DataPointer = (IntPtr)(&arg1);
    descrs[0].Size = 4;
    descrs[1].DataPointer = (IntPtr)(&arg2);
    descrs[1].Size = 4;
    descrs[2].DataPointer = (IntPtr)(&arg3);
    descrs[2].Size = 4;
    descrs[3].DataPointer = (IntPtr)(&arg4);
    descrs[3].Size = 4;

    WriteEventCore(eventId, 4, (IntPtr)descrs);
}
```

Collect and View EventSource Traces

9/20/2022 • 7 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 3.1 and later versions ✓ .NET Framework 4.5 and later versions

The [Getting started guide](#) showed you how to create a minimal EventSource and collect events in a trace file. This tutorial shows how different tools can configure which events are collected in a trace and then view the traces.

Example app

You'll use the following sample app that generates events for this tutorial. Compile a .NET console application containing the following code:

```
using System.Diagnostics.Tracing;

namespace EventSourceDemo
{
    public static class Program
    {
        public static void Main(string[] args)
        {
            DemoEventSource.Log.AppStarted("Hello World!", 12);
            DemoEventSource.Log.DebugMessage("Got here");
            DemoEventSource.Log.DebugMessage("finishing startup");
            DemoEventSource.Log.RequestStart(3);
            DemoEventSource.Log.RequestStop(3);
        }
    }

    [EventSource(Name = "Demo")]
    class DemoEventSource : EventSource
    {
        public static DemoEventSource Log { get; } = new DemoEventSource();

        [Event(1, Keywords = Keywords.Startup)]
        public void AppStarted(string message, int favoriteNumber) => WriteEvent(1, message,
favoriteNumber);

        [Event(2, Keywords = Keywords.Requests)]
        public void RequestStart(int requestId) => WriteEvent(2, requestId);
        [Event(3, Keywords = Keywords.Requests)]
        public void RequestStop(int requestId) => WriteEvent(3, requestId);
        [Event(4, Keywords = Keywords.Startup, Level = EventLevel.Verbose)]
        public void DebugMessage(string message) => WriteEvent(4, message);

        public class Keywords
        {
            public const EventKeywords Startup = (EventKeywords)0x0001;
            public const EventKeywords Requests = (EventKeywords)0x0002;
        }
    }
}
```

Configure which events to collect

Most event collection tools use these configuration options to decide which events should be included in a trace:

- Provider names - This is a list of one or more EventSource names. Only events that are defined on EventSources in this list are eligible to be included. To collect events from the DemoEventSource class in the preceding example app, you would need to include the EventSource name "Demo" in the list of provider names.
- Event verbosity level - For each provider you can define a verbosity level and events with **verbosity** higher than that level will be excluded from the trace. If you specified that the "Demo" provider in the preceding example app should collect at the Informational verbosity level, then the DebugMessage event would be excluded because it has a higher level. Specifying **EventLevel LogAlways(0)** is a special case that indicates that events of any verbosity level should be included.
- Event keywords - For each provider you can define a set of keywords and only events tagged with at least one of the keywords will be included. In the example app above if you specified the Startup keyword then only the AppStarted and DebugMessage events would be included. If no keywords are specified this is a special case and means that events with any keyword should be included.

Conventions for describing provider configuration

Although each tool determines its own user interface for setting the trace configuration, there's a common convention many tools use when specifying the configuration as a text string. The list of providers is specified as a semi-colon delimited list, and each provider element in the list consists of name, keywords, and level separated by colons. For example, "Demo:3:5" identifies the EventSource named "Demo" with the keyword bitmask 3 (the `Startup` bit and the `Requests` bit) and `EventLevel` 5, which is `Verbose`. Many tools also let you omit the level and keywords if no level or keyword filtering is desired. For example, "Demo:5" only does level-based filtering, "Demo:3" only does keyword-based filtering, and "Demo" does no keyword or level filtering.

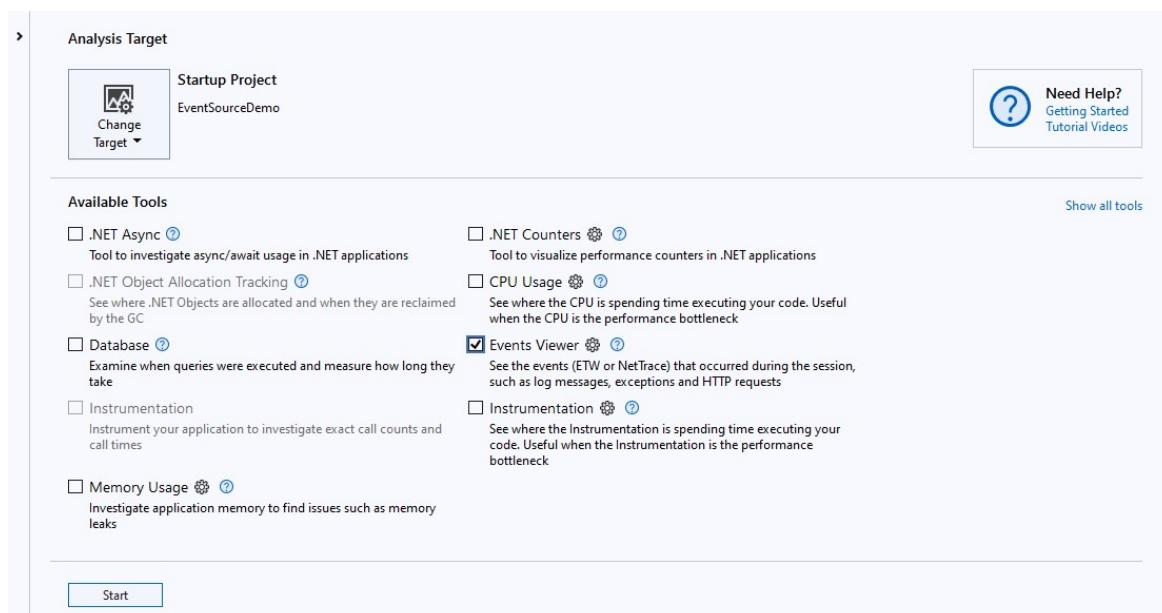
Visual Studio

The [Visual Studio profiler](#) supports both collecting and viewing traces. It also can view traces that have been collected in advance by other tools, such as [dotnet-trace](#).

Collect a trace

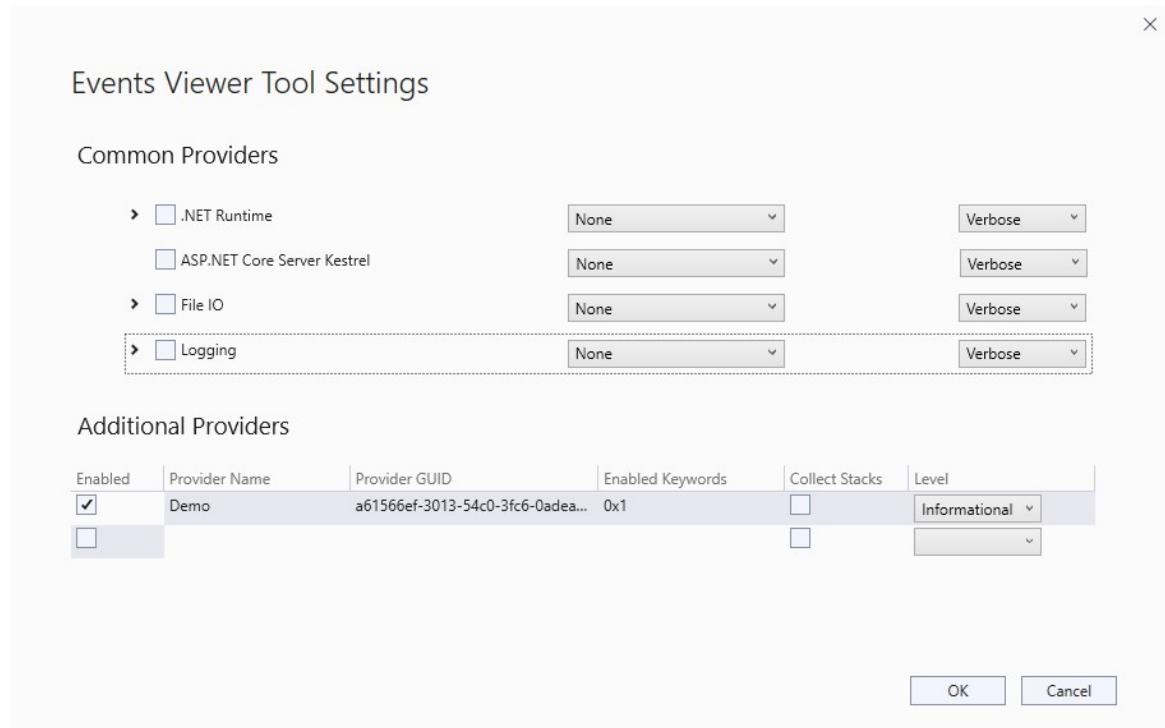
Most of Visual Studio's profiling tools use predefined sets of events that serve a particular purpose, such as analyzing CPU usage or allocations. To collect a trace with customized events, you'll use the [Events Viewer](#) tool.

- To open the Performance Profiler in Visual Studio, select Alt+F2.
- Select the **Events Viewer** check box.



- Select the small gear icon to the right of Events Viewer to open the configuration window.

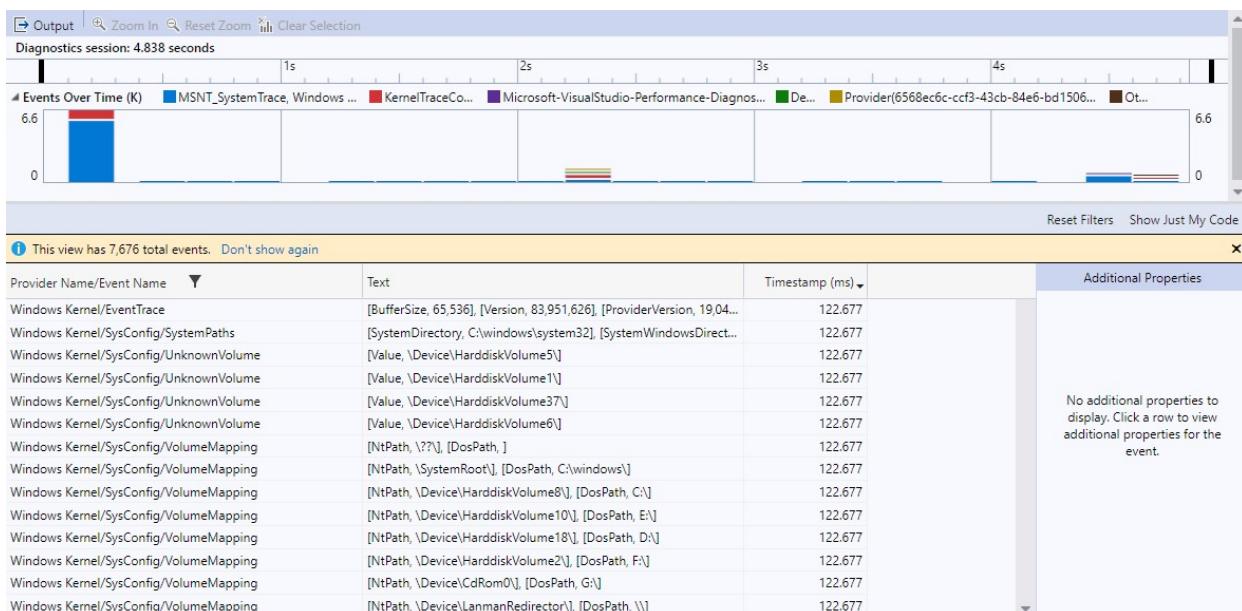
In the table below **Additional Providers**, add a row for each provider you wish to configure by clicking the **Enabled** checkbox, and then entering the provider name, keywords, and level. You don't need to enter the provider GUID; it's computed automatically.



4. Select **OK** to confirm the configuration settings.
5. Select **Start** to begin running the app and collecting logs.
6. Select **Stop Collection** or exit the app to stop collecting logs and show the collected data.

View a trace

Visual Studio can view traces it collected itself, or it can view traces collected in other tools. To view traces from other tools, use **File > Open** and select a trace file in the file picker. Visual Studio profiler supports **.etl** files (ETW's standard format), **.nettrace** files (EventPipe's standard format), and **.diagsession** files (Visual Studio's standard format). For information about working with trace files in Visual Studio, see the [Visual Studio documentation](#).



NOTE

Visual Studio collects some events automatically from ETW or EventPipe, even if they weren't explicitly configured. If you see events you don't recognize in the Provider Name or Event Name column and want to filter them out, use the filter icon to the right to select only the events you want to view.

PerfView

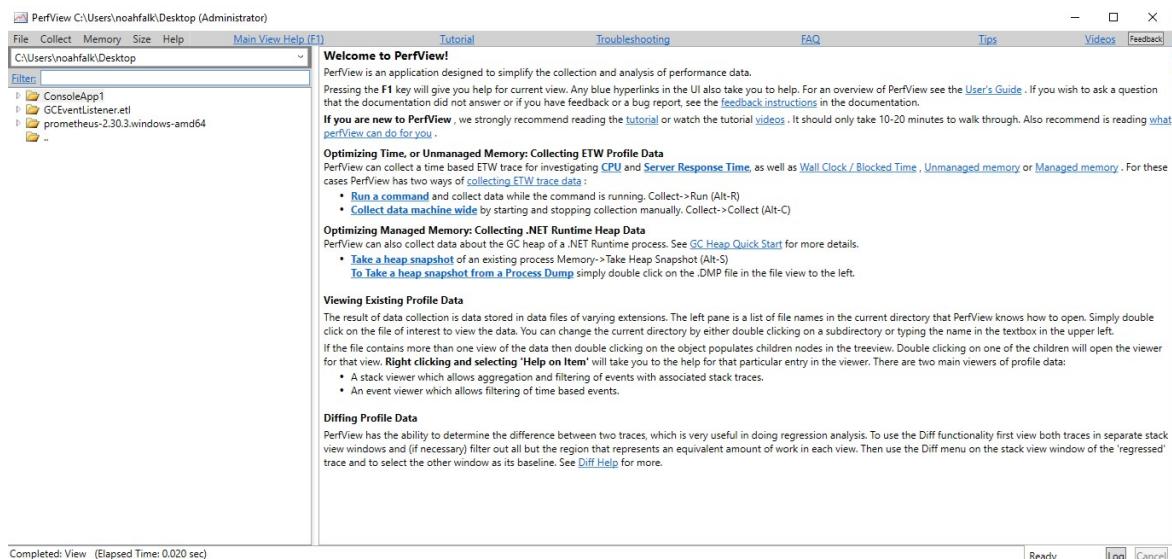
PerfView is a performance tool created by the .NET team that can collect and view ETW traces. It can also view trace files collected by other tools in various formats. In this tutorial, you'll collect an ETW trace of the [demo app](#) and then examine the collected events in PerfView's event viewer.

Collect a trace

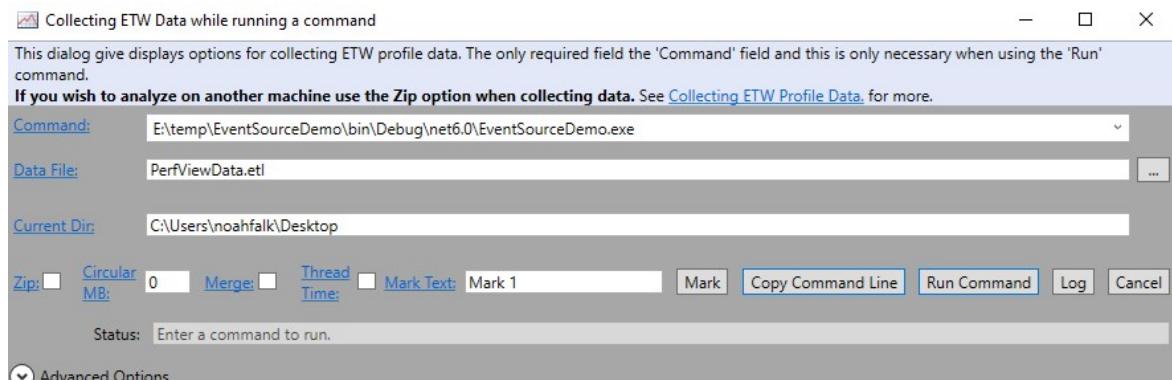
1. Download PerfView from the [releases page](#). This tutorial was done with [PerfView version 2.0.76](#), but any recent version should work.
2. Start PerfView.exe with administrator permissions.

NOTE

ETW trace collection always requires administrator permissions, however if you are only using PerfView to view a pre-existing trace, then no special permissions are needed.



3. From the **Collect** menu, select **Run**. This opens a new dialog where you'll enter the path to the [demo app](#).

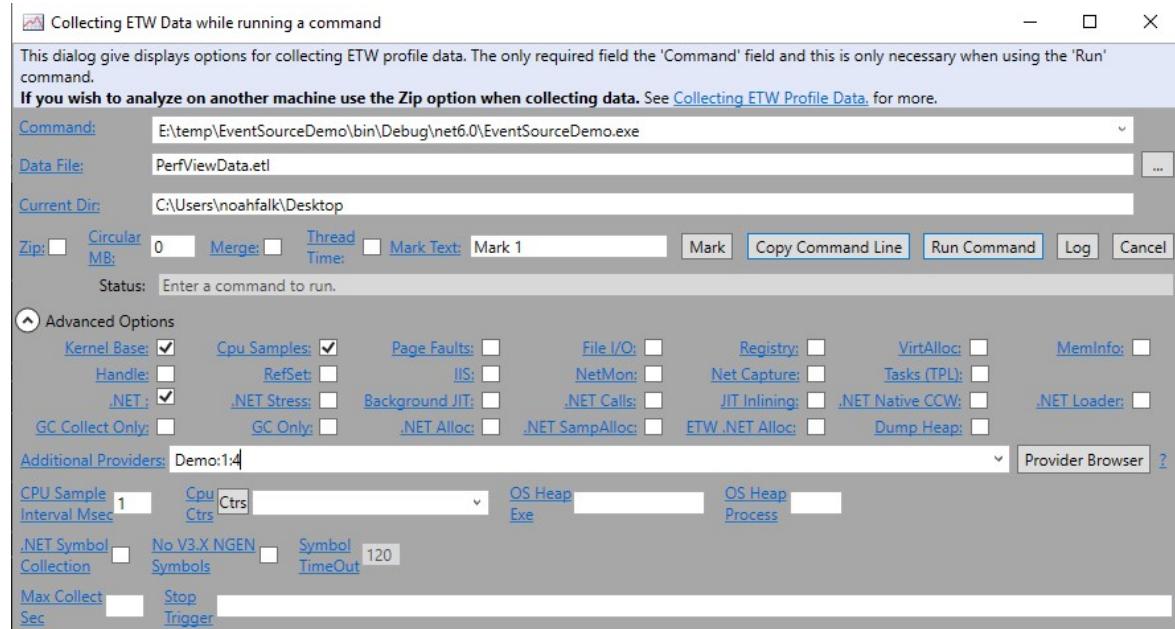


4. To configure which events get collected, expand **Advanced Options** at the bottom of the dialog. In the

Additional Providers text box, enter providers using the [conventional text format](#) described previously.

In this case, you're entering "Demo:1:4", which means keyword bit 1 ([Startup](#) events) and verbosity 4 (

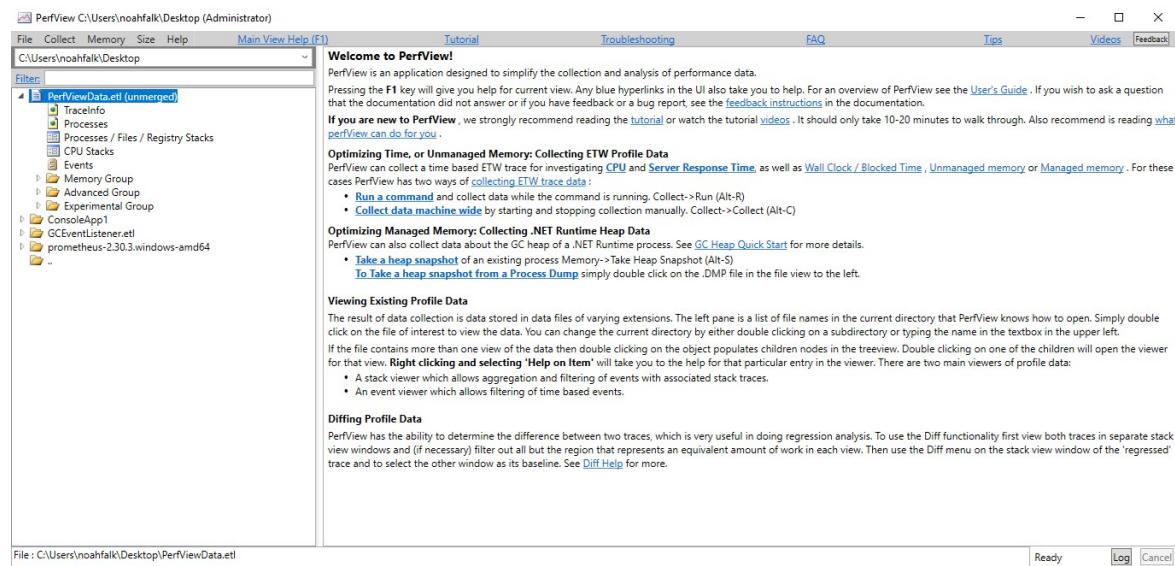
[Informational](#)).



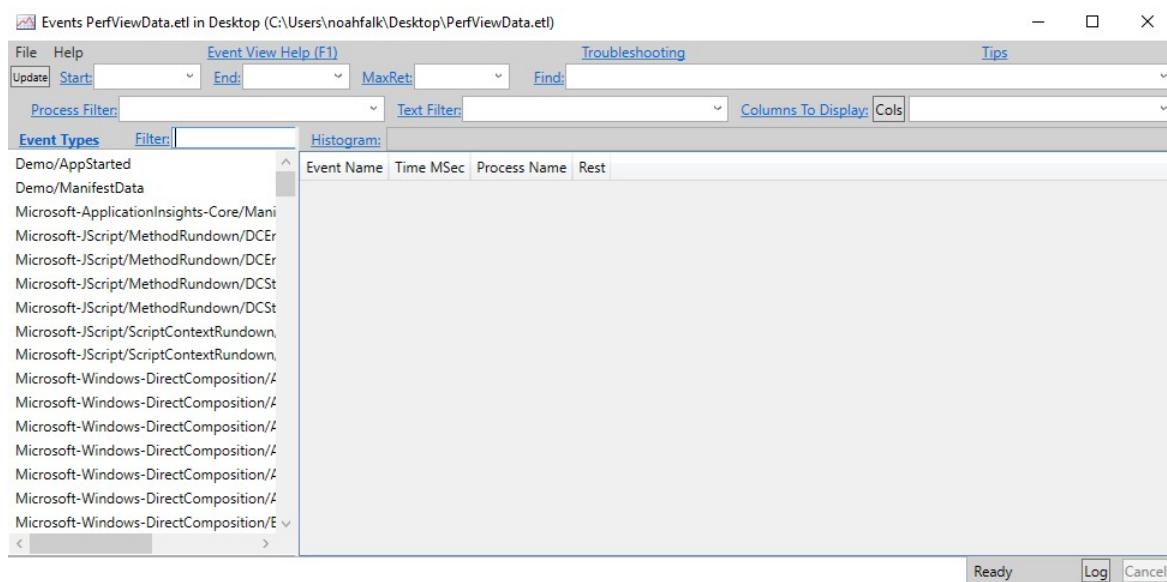
5. To launch the app and begin collecting the trace, select the **Run Command** button. When the app exits, the trace *PerfViewData.etl* is saved in the current directory.

View a trace

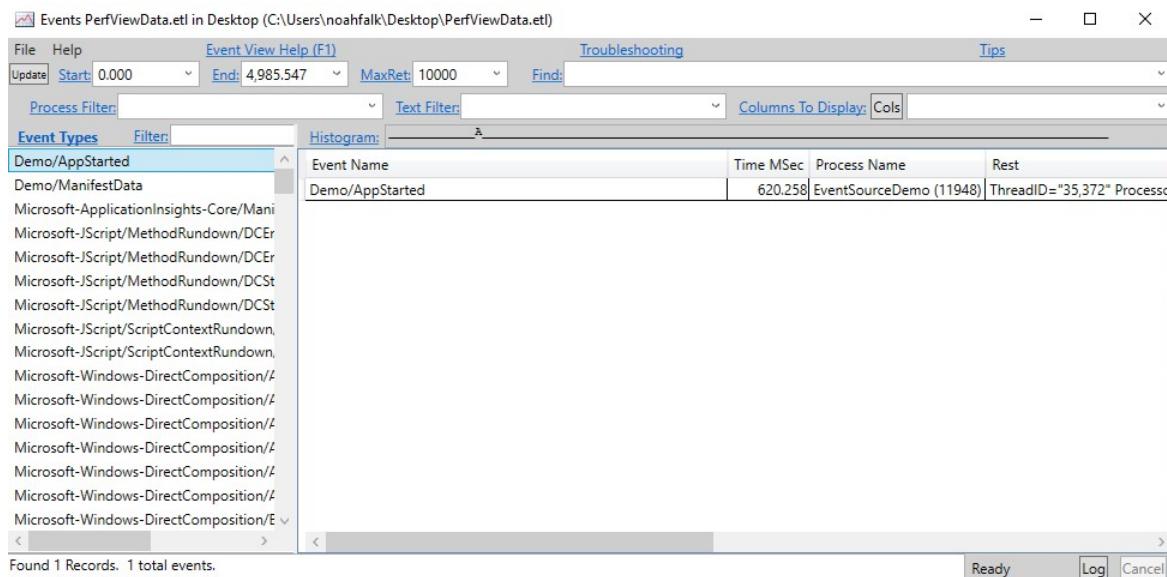
1. In the main window drop-down text box at the upper left, select the directory containing the trace file. Then double click the trace file in the tree view below.



2. To bring up the Events viewer, double click the **Events** item that appears in the tree view below the trace file.



3. All event types in the trace are shown in the list at the left. Double click an event type, such as Demo\AppStarted, to show all events of that type in the table on the right.



Learn more

To learn more about using PerfView, see the [PerfView video tutorials](#).

dotnet-trace

[dotnet-trace](#) is a cross-platform command-line tool that can collect traces from .NET Core apps using [EventPipe](#) tracing. It doesn't support viewing trace data, but the traces it collects can be viewed by other tools such as [PerfView](#) or [Visual Studio](#). dotnet-trace also supports converting its default `.nettrace` format traces into other formats, such as Chromium or [Speedscope](#).

Collect a trace

1. Download and Install [dotnet-trace](#).
2. At the command line, run the `dotnet-trace collect` command:

```
E:\temp\EventSourceDemo\bin\Debug\net6.0>dotnet-trace collect --providers Demo:1:4 --
EventSourceDemo.exe
```

This should show output similar to:

```
E:\temp\EventSourceDemo\bin\Debug\net6.0> dotnet-trace collect --providers Demo:1:4 --
EventSourceDemo.exe

Provider Name          Keywords          Level          Enabled By
Demo                  0x0000000000000001  Informational(4)  --providers

Launching: EventSourceDemo.exe
Process      : E:\temp\EventSourceDemo\bin\Debug\net6.0\EventSourceDemo.exe
Output File   :
E:\temp\EventSourceDemo\bin\Debug\net6.0\EventSourceDemo.exe_20220317_021512.nettrace

[00:00:00:00]  Recording trace 0.00    (B)
Press <Enter> or <Ctrl+C> to exit...

Trace completed.
```

dotnet-trace uses the [conventional text format](#) for describing provider configuration in the `--providers` argument. For more options on how to take traces using dotnet-trace, see the [dotnet-trace docs](#).

EventListener

[System.Diagnostics.Tracing.EventListener](#) is an .NET API that can be used from in-process to receive callbacks for events generated by an [System.Diagnostics.Tracing.EventSource](#). This API can be used to create custom logging tools or to analyze the events in memory without ever serializing them.

To use `EventListener`, declare a type that derives from `EventListener`, invoke `EnableEvents` to subscribe to the events from any `EventSource` of interest, and override the `OnEventWritten`, which will be called whenever a new event is available. It's often useful to override `OnEventSourceCreated` to discover which `EventSource` objects exist, but this isn't required. Following is an example `EventListener` implementation that prints to console when messages are received:

1. Add this code to the [demo app](#).

```
class ConsoleWriterEventListener : EventListener
{
    protected override void OnEventSourceCreated(EventSource eventSource)
    {
        if(eventSource.Name == "Demo")
        {
            EnableEvents(eventSource, EventLevel.Informational);
        }
    }

    protected override void OnEventWritten(EventWrittenEventArgs eventData)
    {
        Console.WriteLine(eventData.TimeStamp + " " + eventData.EventName);
    }
}
```

2. Modify the `Main` method to create an instance of the new listener.

```
public static void Main(string[] args)
{
    ConsoleWriterEventLister listener = new ConsoleWriterEventLister();

    DemoEventSource.Log.AppStarted("Hello World!", 12);
    DemoEventSource.Log.DebugMessage("Got here");
    DemoEventSource.Log.DebugMessage("finishing startup");
    DemoEventSource.Log.RequestStart(3);
    DemoEventSource.Log.RequestStop(3);
}
```

3. Build and run the app. Previously, it had no output, but now it writes the events to the console:

```
3/24/2022 9:23:35 AM AppStarted
3/24/2022 9:23:35 AM RequestStart
3/24/2022 9:23:35 AM RequestStop
```

EventSource Activity IDs

9/20/2022 • 7 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 3.1 and later versions ✓ .NET Framework 4.5 and later versions

This guide explains Activity IDs, an optional identifier that can be logged with each event generated using [System.Diagnostics.Tracing.EventSource](#). For an introduction, see [Getting Started with EventSource](#).

The challenge of logging concurrent work

Long ago, a typical application may have been simple and single-threaded, which makes logging straightforward. You could write each step to a log file in order and then read the log back exactly in the order it was written to understand what happened. If the app handled requests then only one request was handled at a time. All log messages for request A would be printed in order, then all the messages for B, and so on. When apps become multi-threaded that strategy no longer works because multiple requests are being handled at the same time. However if each request is assigned to a single thread that processes it entirely, you can solve the problem by recording a thread ID for each log message. For example, a multi-threaded app might log:

Thread Id	Message
-----	-----
12	BeginRequest A
12	Doing some work
190	BeginRequest B
12	uh-oh error happened
190	Doing some work

By reading the thread IDs you know that thread 12 was processing request A and thread 190 was processing request B, therefore the 'uh-oh error happened' message related to request A. However application concurrency has continued to grow ever more sophisticated. It's now common to use `async` and `await` so that a single request could be handled partially on many different threads before the work is complete. Thread IDs are no longer sufficient to correlate together all the messages produced for one request. Activity IDs solve this problem. They provide a finer grain identifier that can track individual requests, or portions of requests, regardless of if the work is spread across different threads.

NOTE

The Activity ID concept referred to here is not the same as the `System.Diagnostics.Tracing.Activity`, despite the similar naming.

Tracking work using an Activity ID

You can run the code below to see Activity tracking in action.

```
using System.Diagnostics.Tracing;

public static class Program
{
    public static async Task Main(string[] args)
    {
        ConsoleWriterEventListener listener = new ConsoleWriterEventListener();

        Task a = ProcessWorkItem("A");
    }
}
```

```

        Task b = ProcessWorkItem("B");
        await Task.WhenAll(a, b);
    }

    private static async Task ProcessWorkItem(string requestName)
    {
        DemoEventSource.Log.WorkStart(requestName);
        await HelperA();
        await HelperB();
        DemoEventSource.Log.WorkStop();
    }

    private static async Task HelperA()
    {
        DemoEventSource.Log.DebugMessage("HelperA");
        await Task.Delay(100); // pretend to do some work
    }

    private static async Task HelperB()
    {
        DemoEventSource.Log.DebugMessage("HelperB");
        await Task.Delay(100); // pretend to do some work
    }
}

[EventSource(Name ="Demo")]
class DemoEventSource : EventSource
{
    public static DemoEventSource Log = new DemoEventSource();

    [Event(1)]
    public void WorkStart(string requestName) => WriteEvent(1, requestName);
    [Event(2)]
    public void WorkStop() => WriteEvent(2);

    [Event(3)]
    public void DebugMessage(string message) => WriteEvent(3, message);
}

class ConsoleWriterEventListener : EventListener
{
    protected override void OnEventSourceCreated(EventSource eventSource)
    {
        if(eventSource.Name == "Demo")
        {
            Console.WriteLine("{0,-5} {1,-40} {2,-15} {3}", "TID", "Activity ID", "Event", "Arguments");
            EnableEvents(eventSource, EventLevel.Verbose);
        }
        else if(eventSource.Name == "System.Threading.Tasks.TplEventSource")
        {
            // Activity IDs aren't enabled by default.
            // Enabling Keyword 0x80 on the TplEventSource turns them on
            EnableEvents(eventSource, EventLevel.LogAlways, (EventKeywords)0x80);
        }
    }

    protected override void OnEventWritten(EventWrittenEventArgs eventData)
    {
        lock (this)
        {
            Console.Write("{0,-5} {1,-40} {2,-15} ", eventData.OSThreadId, eventData.ActivityId,
eventData.EventName);
            if (eventData.Payload.Count == 1)
            {
                Console.WriteLine(eventData.Payload[0]);
            }
            else
            {
                Console.WriteLine();
            }
        }
    }
}

```

```

        }
    }
}

```

When run, this code prints output:

TID	Activity ID	Event	Arguments
21256	00000011-0000-0000-0000-00006ab99d59	WorkStart	A
21256	00000011-0000-0000-0000-00006ab99d59	DebugMessage	HelperA
21256	00000012-0000-0000-0000-00006bb99d59	WorkStart	B
21256	00000012-0000-0000-0000-00006bb99d59	DebugMessage	HelperA
14728	00000011-0000-0000-0000-00006ab99d59	DebugMessage	HelperB
11348	00000012-0000-0000-0000-00006bb99d59	DebugMessage	HelperB
11348	00000012-0000-0000-0000-00006bb99d59	WorkStop	
14728	00000011-0000-0000-0000-00006ab99d59	WorkStop	

NOTE

There is a known issue where Visual Studio debugger may cause invalid Activity IDs to be generated. Either don't run this sample under the debugger or set a breakpoint at the beginning of Main and evaluate the expression 'System.Threading.Tasks.TplEventSource.Log.TasksSetActivityIds = false' in the immediate window before continuing to work around the issue.

Using the Activity IDs you can see that all of the messages for work item A have ID `00000011-...` and all the messages for work item B have ID `00000012-...`. Both work items first did some work on thread 21256, but then each of them continued their work on separate threadpool threads 11348 and 14728 so trying to track the request only with thread IDs wouldn't have worked.

EventSource has an automatic heuristic where defining an event named `_Something_Start` followed immediately by another event named `_Something_Stop` is considered the start and stop of a unit of work. Logging the start event for a new unit of work creates a new Activity ID and begins logging all events on the same thread with that Activity ID until the stop event is logged. The ID also automatically follows async control flow when using `async` and `await`. Although we recommend that you use the Start/Stop naming suffixes, you may name the events anything you like by explicitly annotating them using the `EventAttribute.Opcode` property. Set the first event to `EventOpcode.Start` and the second to `EventOpcode.Stop`.

Log requests that do parallel work

Sometimes a single request might do different parts of its work in parallel, and you want to group all the log events and the subparts. The example below simulates a request that does two database queries in parallel and then does some processing on the result of each query. You want to isolate the work for each query, but also understand which queries belong to the same request when many concurrent requests could be running. This is modeled as a tree where each top-level request is a root and then subportions of work are branches. Each node in the tree gets its own Activity ID, and the first event logged with the new child Activity ID logs an extra field called Related Activity ID to describe its parent.

Run the following code:

```

using System.Diagnostics.Tracing;

public static class Program
{
    public static async Task Main(string[] args)
    {
        ConsoleWriterEventListener listener = new ConsoleWriterEventListener();

```

```

        await ProcessWorkItem("A");
    }

    private static async Task ProcessWorkItem(string requestName)
    {
        DemoEventSource.Log.WorkStart(requestName);
        Task query1 = Query("SELECT bowls");
        Task query2 = Query("SELECT spoons");
        await Task.WhenAll(query1, query2);
        DemoEventSource.Log.WorkStop();
    }

    private static async Task Query(string query)
    {
        DemoEventSource.Log.QueryStart(query);
        await Task.Delay(100); // pretend to send a query
        DemoEventSource.Log.DebugMessage("processing query");
        await Task.Delay(100); // pretend to do some work
        DemoEventSource.Log.QueryStop();
    }
}

[EventSource(Name = "Demo")]
class DemoEventSource : EventSource
{
    public static DemoEventSource Log = new DemoEventSource();

    [Event(1)]
    public void WorkStart(string requestName) => WriteEvent(1, requestName);
    [Event(2)]
    public void WorkStop() => WriteEvent(2);
    [Event(3)]
    public void DebugMessage(string message) => WriteEvent(3, message);
    [Event(4)]
    public void QueryStart(string query) => WriteEvent(4, query);
    [Event(5)]
    public void QueryStop() => WriteEvent(5);
}

class ConsoleWriterEventListener : EventListener
{
    protected override void OnEventSourceCreated(EventSource eventSource)
    {
        if (eventSource.Name == "Demo")
        {
            Console.WriteLine("{0,-5} {1,-40} {2,-40} {3,-15} {4}", "TID", "Activity ID", "Related Activity ID", "Event", "Arguments");
            EnableEvents(eventSource, EventLevel.Verbose);
        }
        else if (eventSource.Name == "System.Threading.Tasks.TplEventSource")
        {
            // Activity IDs aren't enabled by default.
            // Enabling Keyword 0x80 on the TplEventSource turns them on
            EnableEvents(eventSource, EventLevel.LogAlways, (EventKeywords)0x80);
        }
    }

    protected override void OnEventWritten(EventWrittenEventArgs eventData)
    {
        lock (this)
        {
            Console.Write("{0,-5} {1,-40} {2, -40} {3,-15} ", eventData.OSThreadId, eventData.ActivityId,
            eventData.RelatedActivityId, eventData.EventName);
            if (eventData.Payload.Count == 1)
            {
                Console.WriteLine(eventData.Payload[0]);
            }
            else
            {
                Console.WriteLine("Multiple payload items found!");
            }
        }
    }
}

```

```
        {
            Console.WriteLine();
        }
    }
}
```

This example prints output such as:

TID	Activity ID	Related Activity ID	Event
Arguments			
34276	0000011-0000-0000-0000-000086af9d59	0000000-0000-0000-0000-000000000000	WorkStart
34276	00001011-0000-0000-0000-000086af9d59	0000011-0000-0000-0000-000086af9d59	QueryStart
SELECT bowls			
34276	00002011-0000-0000-0000-0000868f9d59	0000011-0000-0000-0000-000086af9d59	QueryStart
SELECT spoons			
32684	00002011-0000-0000-0000-0000868f9d59	0000000-0000-0000-0000-000000000000	DebugMessage
processing query			
18624	00001011-0000-0000-0000-0000869f9d59	0000000-0000-0000-0000-000000000000	DebugMessage
processing query			
18624	00002011-0000-0000-0000-0000868f9d59	0000000-0000-0000-0000-000000000000	QueryStop
32684	00001011-0000-0000-0000-0000869f9d59	0000000-0000-0000-0000-000000000000	QueryStop
32684	0000011-0000-0000-0000-000086af9d59	0000000-0000-0000-0000-000000000000	WorkStop

This example only ran one top-level request, which was assigned Activity ID `00000011-....`. Then each `QueryStart` event began a new branch of the request with Activity IDs `00001011-....` and `00002011-....` respectively. You can identify these IDs are children of the original request because both of the start events logged their parent `00000011-....` in the Related Activity ID field.

NOTE

You may have noticed the numerical values of the IDs have some clear patterns between parent and child and aren't random. Although it can assist in spotting the relationship visually in simple cases, it's best for tools not to rely on this and treat the IDs as opaque identifiers. As the nesting level grows deeper, the byte pattern will change. Using the Related Activity ID field is the best way to ensure tools work reliably regardless of nesting level.

Because requests with complex trees of subwork items will generate many different Activity IDs, these IDs are usually best parsed by tools rather than trying to reconstruct the tree by hand. [PerfView](#) is one tool that knows how to correlate events annotated with these IDs.

DiagnosticSource and DiagnosticListener

9/20/2022 • 11 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 3.1 and later versions ✓ .NET Framework 4.5 and later versions

[System.Diagnostics.DiagnosticSource](#) is a module that allows code to be instrumented for production-time logging of rich data payloads for consumption within the process that was instrumented. At run time, consumers can dynamically discover data sources and subscribe to the ones of interest.

[System.Diagnostics.DiagnosticSource](#) was designed to allow in-process tools to access rich data. When using [System.Diagnostics.DiagnosticSource](#), the consumer is assumed to be within the same process and as a result, non-serializable types (for example, `HttpResponseMessage` or `HttpContext`) can be passed, giving customers plenty of data to work with.

Getting Started with DiagnosticSource

This walkthrough shows how to create a `DiagnosticSource` event and instrument code with [System.Diagnostics.DiagnosticSource](#). It then explains how to consume the event by finding interesting `DiagnosticListeners`, subscribing to their events, and decoding event data payloads. It finishes by describing *filtering*, which allows only specific events to pass through the system.

DiagnosticSource Implementation

You will work with the following code. This code is an `HttpClient` class with a `SendWebRequest` method that sends an HTTP request to the URL and receives a reply.

```
using System.Diagnostics;
MyListener TheListener = new MyListener();
TheListener.Listening();
HTTPClient Client = new HTTPClient();
Client.SendWebRequest("https://docs.microsoft.com/dotnet/core/diagnostics/");

class HTTPClient
{
    private static DiagnosticSource httpLogger = new DiagnosticListener("System.Net.Http");
    public byte[] SendWebRequest(string url)
    {
        if (httpLogger.IsEnabled("RequestStart"))
        {
            httpLogger.Write("RequestStart", new { Url = url });
        }
        //Pretend this sends an HTTP request to the url and gets back a reply.
        byte[] reply = new byte[] { };
        return reply;
    }
}
class Observer<T> : IObserver<T>
{
    public Observer(Action<T> onNext, Action onCompleted)
    {
        _onNext = onNext ?? new Action<T>(_ => { });
        _onCompleted = onCompleted ?? new Action(() => { });
    }
    public void OnCompleted() { _onCompleted(); }
    public void OnError(Exception error) { }
    public void OnNext(T value) { _onNext(value); }
    private Action<T> _onNext;
    private Action _onCompleted;
```

```

}
class MyListener
{
    IDisposable networkSubscription;
    IDisposable listenerSubscription;
    private readonly object allListeners = new();
    public void Listening()
    {
        Action<KeyValuePair<string, object>> whenHeard = delegate (KeyValuePair<string, object> data)
        {
            Console.WriteLine($"Data received: {data.Key}: {data.Value}");
        };
        Action<DiagnosticListener> onNewListener = delegate (DiagnosticListener listener)
        {
            Console.WriteLine($"New Listener discovered: {listener.Name}");
            //Subscribe to the specific DiagnosticListener of interest.
            if (listener.Name == "System.Net.Http")
            {
                //Use lock to ensure the callback code is thread safe.
                lock (allListeners)
                {
                    if (networkSubscription != null)
                    {
                        networkSubscription.Dispose();
                    }
                    IObserver<KeyValuePair<string, object>> iobserver = new Observer<KeyValuePair<string, object>>(whenHeard, null);
                    networkSubscription = listener.Subscribe(iobserver);
                }
            }
        };
        //Subscribe to discover all DiagnosticListeners
        IObserver<DiagnosticListener> observer = new Observer<DiagnosticListener>(onNewListener, null);
        //When a listener is created, invoke the onNext function which calls the delegate.
        listenerSubscription = DiagnosticListener.AllListeners.Subscribe(observer);
    }
    // Typically you leave the listenerSubscription subscription active forever.
    // However when you no longer want your callback to be called, you can
    // call listenerSubscription.Dispose() to cancel your subscription to the IObservable.
}

```

Running the provided implementation prints to the console.

```

New Listener discovered: System.Net.Http
Data received: RequestStart: { Url = https://docs.microsoft.com/dotnet/core/diagnostics/ }

```

Log an event

The `DiagnosticSource` type is an abstract base class that defines the methods needed to log events. The class that holds the implementation is `DiagnosticListener`. The first step in instrumenting code with `DiagnosticSource` is to create a `DiagnosticListener`. For example:

```
private static DiagnosticSource httpLogger = new DiagnosticListener("System.Net.Http");
```

Notice that `httpLogger` is typed as a `DiagnosticSource`. That's because this code only writes events and thus is only concerned with the `DiagnosticSource` methods that the `DiagnosticListener` implements. `DiagnosticListeners` are given names when they are created, and this name should be the name of a logical grouping of related events (typically the component). Later, this name is used to find the Listener and subscribe to any of its events. Thus the event names only need to be unique within a component.

The `DiagnosticSource` logging interface consists of two methods:

```
bool IsEnabled(string name)
void Write(string name, object value);
```

This is instrument site specific. You need to check the instrumentation site to see what types are passed into `IsEnabled`. This provides you with the information to know what to cast the payload to.

A typical call site will look like:

```
if (httpLogger.IsEnabled("RequestStart"))
{
    httpLogger.Write("RequestStart", new { Url = url });
}
```

Every event has a `string` name (for example, `RequestStart`), and exactly one `object` as a payload. If you need to send more than one item, you can do so by creating an `object` with properties to represent all its information. C#'s [anonymous type](#) feature is typically used to create a type to pass 'on the fly', and makes this scheme very convenient. At the instrumentation site, you must guard the call to `Write()` with an `.IsEnabled()` check on the same event name. Without this check, even when the instrumentation is inactive, the rules of the C# language require all the work of creating the payload `object` and calling `Write()` to be done, even though nothing is actually listening for the data. By guarding the `Write()` call, you make it efficient when the source is not enabled.

Combining everything you have:

```
class HttpClient
{
    private static DiagnosticSource httpLogger = new DiagnosticListener("System.Net.Http");
    public byte[] SendWebRequest(string url)
    {
        if (httpLogger.IsEnabled("RequestStart"))
        {
            httpLogger.Write("RequestStart", new { Url = url });
        }
        //Pretend this sends an HTTP request to the url and gets back a reply.
        byte[] reply = new byte[] { };
        return reply;
    }
}
```

Discovery of `DiagnosticListeners`

The first step in receiving events is to discover which `DiagnosticListeners` you are interested in.

`DiagnosticListener` supports a way of discovering `DiagnosticListeners` that are active in the system at run time. The API to accomplish this is the [AllListeners](#) property.

Implement an `Observer<T>` class that inherits from the `IEnumerable` interface, which is the 'callback' version of the `IEnumerable` interface. You can learn more about it at the [Reactive Extensions](#) site. An `IEnumerable` has three callbacks, `OnNext`, `OnComplete`, and `OnError`. An `IEnumerable` has a single method called `Subscribe` that gets passed one of these Observers. Once connected, the Observer gets callbacks (mostly `OnNext` callbacks) when things happen.

A typical use of the `AllListeners` static property looks like this:

```

class Observer<T> : IObserver<T>
{
    public Observer(Action<T> onNext, Action onCompleted)
    {
        _onNext = onNext ?? new Action<T>(_ => { });
        _onCompleted = onCompleted ?? new Action(() => { });
    }
    public void OnCompleted() { _onCompleted(); }
    public void OnError(Exception error) { }
    public void OnNext(T value) { _onNext(value); }
    private Action<T> _onNext;
    private Action _onCompleted;
}
class MyListener
{
    IDisposable networkSubscription;
    IDisposable listenerSubscription;
    private readonly object allListeners = new();
    public void Listening()
    {
        Action<KeyValuePair<string, object>> whenHeard = delegate (KeyValuePair<string, object> data)
        {
            Console.WriteLine($"Data received: {data.Key}: {data.Value}");
        };
        Action<DiagnosticListener> onNewListener = delegate (DiagnosticListener listener)
        {
            Console.WriteLine($"New Listener discovered: {listener.Name}");
            //Subscribe to the specific DiagnosticListener of interest.
            if (listener.Name == "System.Net.Http")
            {
                //Use lock to ensure the callback code is thread safe.
                lock (allListeners)
                {
                    if (networkSubscription != null)
                    {
                        networkSubscription.Dispose();
                    }
                    IObserver<KeyValuePair<string, object>> iobserver = new Observer<KeyValuePair<string,
object>>(whenHeard, null);
                    networkSubscription = listener.Subscribe(iobserver);
                }
            }
        };
        //Subscribe to discover all DiagnosticListeners
        IObserver<DiagnosticListener> observer = new Observer<DiagnosticListener>(onNewListener, null);
        //When a listener is created, invoke the onNext function which calls the delegate.
        listenerSubscription = DiagnosticListener.AllListeners.Subscribe(observer);
    }
    // Typically you leave the listenerSubscription subscription active forever.
    // However when you no longer want your callback to be called, you can
    // call listenerSubscription.Dispose() to cancel your subscription to the IObservable.
}

```

This code creates a callback delegate and, using the `AllListeners.Subscribe` method, requests that the delegate be called for every active `DiagnosticListener` in the system. The decision of whether or not to subscribe to the listener is made by inspecting its name. The code above is looking for the 'System.Net.Http' listener that you created previously.

Like all calls to `Subscribe()`, this one returns an `IDisposable` that represents the subscription itself. Callbacks will continue to happen as long as nothing calls `Dispose()` on this subscription object. The code example never calls `Dispose()`, so it will receive callbacks forever.

When you subscribe to `AllListeners`, you get a callback for ALL ACTIVE `DiagnosticListeners`. Thus, upon

subscribing, you get a flurry of callbacks for all existing `DiagnosticListeners`, and as new ones are created, you receive a callback for those as well. You receive a complete list of everything it's possible to subscribe to.

Subscribe to `DiagnosticListeners`

A `DiagnosticListener` implements the `IEnumerable<KeyValuePair<string, object>>` interface, so you can call `Subscribe()` on it as well. The following code shows how to fill out the previous example:

```
Disposable networkSubscription;
Disposable listenerSubscription;
private readonly object allListeners = new();
public void Listening()
{
    Action<KeyValuePair<string, object>> whenHeard = delegate (KeyValuePair<string, object> data)
    {
        Console.WriteLine($"Data received: {data.Key}: {data.Value}");
    };
    Action<DiagnosticListener> onNewListener = delegate (DiagnosticListener listener)
    {
        Console.WriteLine($"New Listener discovered: {listener.Name}");
        //Subscribe to the specific DiagnosticListener of interest.
        if (listener.Name == "System.Net.Http")
        {
            //Use lock to ensure the callback code is thread safe.
            lock (allListeners)
            {
                if (networkSubscription != null)
                {
                    networkSubscription.Dispose();
                }
                IObserver<KeyValuePair<string, object>> iobserver = new Observer<KeyValuePair<string, object>>(whenHeard, null);
                networkSubscription = listener.Subscribe(iobserver);
            }
        }
    };
    //Subscribe to discover all DiagnosticListeners
    IObserver<DiagnosticListener> observer = new Observer<DiagnosticListener>(onNewListener, null);
    //When a listener is created, invoke the onNext function which calls the delegate.
    listenerSubscription = DiagnosticListener.AllListeners.Subscribe(observer);
}
```

In this example, after finding the 'System.Net.Http' `DiagnosticListener`, an action is created that prints out the name of the listener event, and `payload.ToString()`.

NOTE

`DiagnosticListener` implements `IEnumerable<KeyValuePair<string, object>>`. This means on each callback we get a `KeyValuePair`. The key of this pair is the name of the event and the value is the payload `object`. The example simply logs this information to the console.

It's important to keep track of subscriptions to the `DiagnosticListener`. In the previous code, the `networkSubscription` variable remembers this. If you form another `creation`, you must unsubscribe the previous listener and subscribe to the new one.

The `DiagnosticSource / DiagnosticListener` code is thread safe, but the callback code also needs to be thread safe. To ensure the callback code is thread safe, locks are used. It is possible to create two `DiagnosticListeners` with the same name at the same time. To avoid race conditions, updates of shared variables are performed under the protection of a lock.

Once the previous code is run, the next time a `Write()` is done on 'System.Net.Http' `DiagnosticListener` the information will be logged to the console.

Subscriptions are independent of one another. As a result, other code can do exactly the same thing as the code example and generate two 'pipes' of the logging information.

Decode Payloads

The `KeyValuePair` that is passed to the callback has the event name and payload, but the payload is typed simply as an `object`. There are two ways of getting more specific data:

If the payload is a well known type (for example, a `string`, or an `HttpMessageRequest`), then you can simply cast the `object` to the expected type (using the `as` operator so as not to cause an exception if you are wrong) and then access the fields. This is very efficient.

Use reflection API. For example, assume the following method is present.

```
/// Define a shortcut method that fetches a field of a particular name.
static class PropertyExtensions
{
    static object GetProperty(this object _this, string propertyName)
    {
        return _this.GetType().GetTypeInfo().GetDeclaredProperty(propertyName)?.GetValue(_this);
    }
}
```

To decode the payload more fully, you could replace the `listener.Subscribe()` call with the following code.

```
networkSubscription = listener.Subscribe(delegate(KeyValuePair<string, object> evnt) {
    var eventName = evnt.Key;
    var payload = evnt.Value;
    if (eventName == "RequestStart")
    {
        var url = payload.GetProperty("Url") as string;
        var request = payload.GetProperty("Request");
        Console.WriteLine("Got RequestStart with URL {0} and Request {1}", url, request);
    }
});
```

Note that using reflection is relatively expensive. However, using reflection is the only option if the payloads were generated using anonymous types. This overhead can be reduced by making fast, specialized property fetchers using either `PropertyInfo.GetMethod.CreateDelegate()` or `xref<System.Reflection.Emit>` namespace, but that's beyond the scope of this article. (For an example of a fast, delegate-based property fetcher, see the `PropertySpec` class used in the `DiagnosticSourceEventSource`.)

Filtering

In the previous example, the code uses the `IEnumerable.Subscribe()` method to hook up the callback. This causes all events to be given to the callback. However, `DiagnosticListener` has overloads of `Subscribe()` that allow the controller to control which events are given.

The `listener.Subscribe()` call in the previous example can be replaced with the following code to demonstrate.

```

// Create the callback delegate.
Action<KeyValuePair<string, object>> callback = (KeyValuePair<string, object> evnt) =>
    Console.WriteLine("From Listener {0} Received Event {1} with payload {2}", networkListener.Name,
evnt.Key, evnt.Value.ToString());

// Turn it into an observer (using the Observer<T> Class above).
Observer<KeyValuePair<string, object>> observer = new Observer<KeyValuePair<string, object>>(callback);

// Create a predicate (asks only for one kind of event).
Predicate<string> predicate = (string eventName) => eventName == "RequestStart";

// Subscribe with a filter predicate.
IDisposable subscription = listener.Subscribe(observer, predicate);

// subscription.Dispose() to stop the callbacks.

```

This efficiently subscribes to only the 'RequestStart' events. All other events will cause the `DiagnosticSource.IsEnabled()` method to return `false` and thus be efficiently filtered out.

Context-based filtering

Some scenarios require advanced filtering based on extended context. Producers can call `DiagnosticSource.IsEnabled` overloads and supply additional event properties as shown in the following code.

```

//aRequest and anActivity are the current request and activity about to be logged.
if (httpLogger.IsEnabled("RequestStart", aRequest, anActivity))
    httpLogger.Write("RequestStart", new { Url="http://clr", Request=aRequest });

```

The next code example demonstrates that consumers can use such properties to filter events more precisely.

```

// Create a predicate (asks only for Requests for certains URIs)
Func<string, object, object, bool> predicate = (string eventName, object context, object activity) =>
{
    if (eventName == "RequestStart")
    {
        if (context is HttpRequestMessage request)
        {
            return IsUriEnabled(request.RequestUri);
        }
    }
    return false;
}

// Subscribe with a filter predicate
IDisposable subscription = listener.Subscribe(observer, predicate);

```

Producers are not aware of the filter a consumer has provided. `DiagnosticListener` will invoke the provided filter, omitting additional arguments if necessary, thus the filter should expect to receive a `null` context. If a producer calls `.IsEnabled()` with event name and context, those calls are enclosed in an overload that takes only the event name. Consumers must ensure that their filter allows events without context to pass through.

EventPipe

9/20/2022 • 4 minutes to read • [Edit Online](#)

EventPipe is a runtime component that can be used to collect tracing data, similar to ETW or LTTng. The goal of EventPipe is to allow .NET developers to easily trace their .NET applications without having to rely on platform-specific OS-native components such as ETW or LTTng.

EventPipe is the mechanism behind many of the diagnostic tools and can be used for consuming events emitted by the runtime as well as custom events written with [EventSource](#).

This article is a high-level overview of EventPipe. It describes when and how to use EventPipe, and how to configure it to best suit your needs.

EventPipe basics

EventPipe aggregates events emitted by runtime components - for example, the Just-In-Time compiler or the garbage collector - and events written from [EventSource](#) instances in the libraries and user code.

The events are then serialized in the `.nettrace` file format and can be written directly to a file or streamed through a [diagnostic port](#) for out-of-process consumption.

To learn more about the EventPipe serialization format, refer to the [EventPipe format documentation](#).

EventPipe vs. ETW/LTTng

EventPipe is part of the .NET runtime (CoreCLR) and is designed to work the same way across all the platforms .NET Core supports. This allows tracing tools based on EventPipe, such as `dotnet-counters`, `dotnet-gcdump`, and `dotnet-trace`, to work seamlessly across platforms.

However, because EventPipe is a runtime built-in component, its scope is limited to managed code and the runtime itself. EventPipe cannot be used for tracking some lower-level events, such as resolving native code stack or getting various kernel events. If you use C/C++ interop in your app or you want to trace the runtime itself (which is written in C++), or want deeper diagnostics into the behavior of the app that requires kernel events (that is, native-thread context-switching events) you should use ETW or [perf/LTTng](#).

Another major difference between EventPipe and ETW/LTTng is admin/root privilege requirement. To trace an application using ETW or LTTng you need to be an admin/root. Using EventPipe, you can trace applications as long as the tracer (for example, `dotnet-trace`) is run as the same user as the user that launched the application.

The following table is a summary of the differences between EventPipe and ETW/LTTng.

FEATURE	EVENTPIPE	ETW	LTTNG
Cross-platform	Yes	No (only on Windows)	No (only on supported Linux distros)
Require admin/root privilege	No	Yes	Yes
Can get OS/kernel events	No	Yes	Yes
Can resolve native callstacks	No	Yes	Yes

Use EventPipe to trace your .NET application

You can use EventPipe to trace your .NET application in many ways:

- Use one of the [diagnostics tools](#) that are built on top of EventPipe.
- Use [Microsoft.Diagnostics.NETCore.Client](#) library to write your own tool to configure and start EventPipe sessions.
- Use [environment variables](#) to start EventPipe.

After you've produced a `nettrace` file that contains your EventPipe events, you can view the file in [PerfView](#) or Visual Studio. On non-Windows platforms, you can convert the `nettrace` file to a `speedscope` or `Chromium` trace format by using `dotnet-trace convert` command and view it with `speedscope` or Chrome DevTools.

You can also analyze EventPipe traces programmatically with [TraceEvent](#).

Tools that use EventPipe

This is the easiest way to use EventPipe to trace your application. To learn more about how to use each of these tools, refer to each tool's documentation.

- [dotnet-counters](#) lets you monitor and collect various metrics emitted by the .NET runtime and core libraries, as well as custom metrics you can write.
- [dotnet-gcdump](#) lets you collect GC heap dumps of live processes for analyzing an application's managed heap.
- [dotnet-trace](#) lets you collect traces of applications to analyze for performance.

Trace using environment variables

The preferred mechanism for using EventPipe is to use [dotnet-trace](#) or the [Microsoft.Diagnostics.NETCore.Client](#) library.

However, you can use the following environment variables to set up an EventPipe session on an app and have it write the trace directly to a file. To stop tracing, exit the application.

- `DOTNET_EnableEventPipe` : Set this to `1` to start an EventPipe session that writes directly to a file. The default value is `0`.
- `DOTNET_EventPipeOutputPath` : The path to the output EventPipe trace file when it's configured to run via `DOTNET_EnableEventPipe`. The default value is `trace.nettrace`, which will be created in the same directory that the app is running from.

NOTE

Since .NET 6, instances of the string `{pid}` in `DOTNET_EventPipeOutputPath` are replaced with the process id of the process being traced.

- `DOTNET_EventPipeCircularMB` : A hexadecimal value that represents the size of EventPipe's internal buffer in megabytes. This configuration value is only used when EventPipe is configured to run via `DOTNET_EnableEventPipe`. The default buffer size is 1024MB which translates to this environment variable being set to `400`, since `0x400 == 1024`.

NOTE

If the target process writes events too frequently, it can overflow this buffer and some events might be dropped. If too many events are getting dropped, increase the buffer size to see if the number of dropped events reduces. If the number of dropped events does not decrease with a larger buffer size, it may be due to a slow reader preventing the target process' buffers from being flushed.

- `DOTNET_EventPipeProcNumbers` : Set this to `1` to enable capturing processor numbers in EventPipe event headers. The default value is `0`.
- `DOTNET_EventPipeConfig` : Sets up the EventPipe session configuration when starting an EventPipe session with `DOTNET_EnableEventPipe`. The syntax is as follows:

```
<provider>:<keyword>:<level>
```

You can also specify multiple providers by concatenating them with a comma:

```
<provider1>:<keyword1>:<level1>,<provider2>:<keyword2>:<level2>
```

If this environment variable is not set but EventPipe is enabled by `DOTNET_EnableEventPipe`, it will start tracing by enabling the following providers with the following keywords and levels:

- `Microsoft-Windows-DotNETRuntime:4c14fccbd:5`
- `Microsoft-Windows-DotNETRuntimePrivate:4002000b:5`
- `Microsoft-DotNETCore-SampleProfiler:0:5`

To learn more about some of the well-known providers in .NET, refer to [Well-known Event Providers](#).

NOTE

.NET 6 standardizes on the prefix `DOTNET_` instead of `COMPlus_` for environment variables that configure .NET run-time behavior. However, the `COMPlus_` prefix will continue to work. If you're using a previous version of the .NET runtime, you should still use the `COMPlus_` prefix for environment variables.

.NET metrics

9/20/2022 • 2 minutes to read • [Edit Online](#)

Metrics are numerical measurements reported over time, most often used to monitor the health of an application and generate alerts. For example, a web service might track how many requests it receives each second, how many milliseconds it took to respond, and how many of the responses sent an error back to the user. These metrics can be reported to a monitoring system at regular intervals. If the example web service is intended to respond to requests within 400 ms and then one day the response time slows to 600 ms, the monitoring system can notify engineers that the application is not operating as expected.

Getting started

There are two parts to using metrics in a .NET app:

- **Instrumentation:** Code in .NET libraries takes measurements and associates these measurements with a metric name.
- **Collection:** A .NET app developer configures which named metrics need to be transmitted from the app for external storage and analysis. Some tools may also let engineers configure this outside the app using config files or separate UI.

.NET library developers are primarily interested in the instrumentation step. App developers or operational engineers usually focus on the collection step, leveraging the pre-existing instrumentation within libraries they are using. However, if you're an app developer and none of the existing metrics meet your needs, you can also create new metrics.

Next steps

- [Instrumentation tutorial](#) - How to create new metrics in code
- [Collection tutorial](#) - How to store and view metric data for your app
- [Built-in metrics](#) - Discover metrics that are ready for use in .NET runtime libraries
- [Compare metric APIs](#)
- [EventCounters](#) - Learn what EventCounters are, how to implement them, and how to consume them

Creating Metrics

9/20/2022 • 16 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 3.1 and later versions ✓ .NET Framework 4.6.1 and later versions

.NET applications can be instrumented using the [System.Diagnostics.Metrics](#) APIs to track important metrics. Some metrics are included in standard .NET libraries, but you may want to add new custom metrics that are relevant for your applications and libraries. In this tutorial, you will add new metrics and understand what types of metrics are available.

NOTE

.NET has some older metric APIs, namely [EventCounters](#) and [System.Diagnostics.PerformanceCounter](#), that are not covered here. To learn more about these alternatives, see [Compare metric APIs](#).

Create a custom metric

Prerequisites: [.NET Core 3.1 SDK](#) or a later version

Create a new console application that references the [System.Diagnostics.DiagnosticSource NuGet package](#) version 6 or greater. Applications that target .NET 6+ include this reference by default. Then, update the code in [Program.cs](#) to match:

```
> dotnet new console  
> dotnet add package System.Diagnostics.DiagnosticSource
```

```
using System;  
using System.Diagnostics.Metrics;  
using System.Threading;  
  
class Program  
{  
    static Meter s_meter = new Meter("HatCo.HatStore", "1.0.0");  
    static Counter<int> s_hatsSold = s_meter.CreateCounter<int>("hats-sold");  
  
    static void Main(string[] args)  
    {  
        Console.WriteLine("Press any key to exit");  
        while(!Console.KeyAvailable)  
        {  
            // Pretend our store has a transaction each second that sells 4 hats  
            Thread.Sleep(1000);  
            s_hatsSold.Add(4);  
        }  
    }  
}
```

The [System.Diagnostics.Metrics.Meter](#) type is the entry point for a library to create a named group of instruments. Instruments record the numeric measurements that are needed to calculate metrics. Here we used [CreateCounter](#) to create a Counter instrument named "hats-sold". During each pretend transaction, the code calls [Add](#) to record the measurement of hats that were sold, 4 in this case. The "hats-sold" instrument implicitly defines some metrics that could be computed from these measurements, such as the total number of hats sold or hats sold/sec. Ultimately it is up to metric collection tools to determine which metrics to compute and how to

perform those computations, but each instrument has some default conventions that convey the developer's intent. For Counter instruments, the convention is that collection tools show the total count and/or the rate at which the count is increasing.

The generic parameter `int` on `Counter<int>` and `CreateCounter<int>(...)` defines that this counter must be able to store values up to `Int32.MaxValue`. You can use any of `byte`, `short`, `int`, `long`, `float`, `double`, or `decimal` depending on the size of data you need to store and whether fractional values are needed.

Run the app and leave it running for now. We will view the metrics next.

```
> dotnet run  
Press any key to exit
```

Best practices

- Create the Meter once, store it in a static variable or DI container, and use that instance as long as needed. Each library or library subcomponent can (and often should) create its own [Meter](#). Consider creating a new Meter rather than reusing an existing one if you anticipate app developers would appreciate being able to enable and disable the groups of metrics separately.
- The name passed to the [Meter](#) constructor has to be unique to avoid conflicts with any other Meters. Use a dotted hierarchical name that contains the assembly name and optionally a subcomponent name. If an assembly is adding instrumentation for code in a second, independent assembly, the name should be based on the assembly that defines the Meter, not the assembly whose code is being instrumented.
- The [Meter](#) constructor version parameter is optional. We recommend that you provide a version in case you release multiple versions of the library and make changes to the instruments.
- .NET doesn't enforce any naming scheme for metrics, but by convention all the .NET runtime libraries have metric names using '-' if a separator is needed. Other metric ecosystems have encouraged using '!' or '_' as the separator. Microsoft's suggestion is to use '-' in code and let the metric consumer such as OpenTelemetry or Prometheus convert to an alternate separator if needed.
- The APIs to create instruments and record measurements are thread-safe. In .NET libraries, most instance methods require synchronization when invoked on the same object from multiple threads, but that's not needed in this case.
- The Instrument APIs to record measurements ([Add](#) in this example) typically run in <10 ns when no data is being collected, or tens to hundreds of nanoseconds when measurements are being collected by a high-performance collection library or tool. This allows these APIs to be used liberally in most cases, but take care for code that is extremely performance sensitive.

View the new metric

There are many options to store and view metrics. This tutorial uses the [dotnet-counters](#) tool, which is useful for ad-hoc analysis. You can also see the [metrics collection tutorial](#) for other alternatives. If the [dotnet-counters](#) tool is not already installed, use the SDK to install it:

```
> dotnet tool update -g dotnet-counters  
You can invoke the tool using the following command: dotnet-counters  
Tool 'dotnet-counters' (version '5.0.251802') was successfully installed.
```

While the example app is still running, list the running processes in a second shell to determine the process ID:

```
> dotnet-counters ps
  10180 dotnet      C:\Program Files\dotnet\dotnet.exe
  19964 metric-instr E:\temp\metric-instr\bin\Debug\netcoreapp3.1\metric-instr.exe
```

Find the ID for the process name that matches the example app and have dotnet-counters monitor the new counter:

```
> dotnet-counters monitor -p 19964 HatCo.HatStore
Press p to pause, r to resume, q to quit.
  Status: Running

[HatCo.HatStore]
  hats-sold (Count / 1 sec)        4
```

As expected, you can see that HatCo store is steadily selling 4 hats each second.

Types of instruments

In the previous example, we've only demonstrated a [Counter<T>](#) instrument, but there are more instrument types available. Instruments differ in two ways:

- **Default metric computations** - Tools that collect and analyze the instrument measurements will compute different default metrics depending on the instrument.
- **Storage of aggregated data** - Most useful metrics need data to be aggregated from many measurements. One option is the caller provides individual measurements at arbitrary times and the collection tool manages the aggregation. Alternatively, the caller can manage the aggregate measurements and provide them on-demand in a callback.

Types of instruments currently available:

- **Counter** ([CreateCounter](#)) - This instrument tracks a value that increases over time and the caller reports the increments using [Add](#). Most tools will calculate the total and the rate of change in the total. For tools that only show one thing, the rate of change is recommended. For example, assume that the caller invokes [Add\(\)](#) once each second with successive values 1, 2, 4, 5, 4, 3. If the collection tool updates every three seconds, then the total after three seconds is $1+2+4=7$ and the total after six seconds is $1+2+4+5+4+3=19$. The rate of change is the (`current_total - previous_total`), so at three seconds the tool reports $7-0=7$, and after six seconds, it reports $19-7=12$.
- **UpDownCounter** ([CreateCounter](#)) - This instrument tracks a value that may increase or decrease over time. The caller reports the increments and decrements using [Add](#). For example, assume that the caller invokes [Add\(\)](#) once each second with successive values 1, 5, -2, 3, -1, -3. If the collection tool updates every three seconds, then the total after three seconds is $1+5-2=4$ and the total after six seconds is $1+5-2+3-1-3=3$.
- **ObservableCounter** ([CreateObservableCounter](#)) - This instrument is similar to Counter except that the caller is now responsible for maintaining the aggregated total. The caller provides a callback delegate when the ObservableCounter is created and the callback is invoked whenever tools need to observe the current total. For example, if a collection tool updates every three seconds, then the callback function will also be invoked every three seconds. Most tools will have both the total and rate of change in the total available. If only one can be shown, rate of change is recommended. If the callback returns 0 on the initial call, 7 when it is called again after three seconds, and 19 when called after six seconds, then the tool will report those values unchanged as the totals. For rate of change, the tool will show $7-0=7$ after three seconds and $19-7=12$ after six seconds.
- **ObservableUpDownCounter** ([CreateObservableUpDownCounter](#)) - This instrument is similar to

`UpDownCounter` except that the caller is now responsible for maintaining the aggregated total. The caller provides a callback delegate when the `ObservableUpDownCounter` is created and the callback is invoked whenever tools need to observe the current total. For example, if a collection tool updates every three seconds, then the callback function will also be invoked every three seconds. Whatever value is returned by the callback will be shown in the collection tool unchanged as the total.

- **ObservableGauge** ([CreateObservableGauge](#)) - This instrument allows the caller to provide a callback where the measured value is passed through directly as the metric. Each time the collection tool updates, the callback is invoked, and whatever value is returned by the callback is displayed in the tool.
- **Histogram** ([CreateHistogram](#)) - This instrument tracks the distribution of measurements. There isn't a single canonical way to describe a set of measurements, but tools are recommended to use histograms or computed percentiles. For example, assume the caller invoked `Record` to record these measurements during the collection tool's update interval: 1,5,2,3,10,9,7,4,6,8. A collection tool might report that the 50th, 90th, and 95th percentiles of these measurements are 5, 9, and 9 respectively.

Best practices when selecting an instrument type

- For counting things, or any other value that solely increases over time, use Counter or `ObservableCounter`. Choose between Counter and `ObservableCounter` depending on which is easier to add to the existing code: either an API call for each increment operation, or a callback that will read the current total from a variable the code maintains. In extremely hot code paths where performance is important and using `Add` would create more than one million calls per second per thread, using `ObservableCounter` may offer more opportunity for optimization.
- For timing things, Histogram is usually preferred. Often it's useful to understand the tail of these distributions (90th, 95th, 99th percentile) rather than averages or totals.
- Other common cases, such as cache hit rates or sizes of caches, queues, and files are usually well suited for `UpDownCounter` or `ObservableUpDownCounter`. Choose between them depending on which is easier to add to the existing code: either an API call for each increment and decrement operation or a callback that will read the current value from a variable the code maintains.

NOTE

If you're using an older version of .NET or a DiagnosticSource NuGet package that doesn't support `UpDownCounter` and `ObservableUpDownCounter` (before version 7), `observableGauge` is often a good substitute.

Example of different instrument types

Stop the example process started previously, and replace the example code in `Program.cs` with:

```

using System;
using System.Diagnostics.Metrics;
using System.Threading;

class Program
{
    static Meter s_meter = new Meter("HatCo.HatStore", "1.0.0");
    static Counter<int> s_hatsSold = s_meter.CreateCounter<int>("hats-sold");
    static Histogram<int> s_orderProcessingTimeMs = s_meter.CreateHistogram<int>("order-processing-time");
    static int s_coatsSold;
    static int s_ordersPending;

    static Random s_rand = new Random();

    static void Main(string[] args)
    {
        s_meter.CreateObservableCounter<int>("coats-sold", () => s_coatsSold);
        s_meter.CreateObservableGauge<int>("orders-pending", () => s_ordersPending);

        Console.WriteLine("Press any key to exit");
        while(!Console.KeyAvailable)
        {
            // Pretend our store has one transaction each 100ms that each sell 4 hats
            Thread.Sleep(100);
            s_hatsSold.Add(4);

            // Pretend we also sold 3 coats. For an ObservableCounter we track the value in our variable and
            // report it
            // on demand in the callback
            s_coatsSold += 3;

            // Pretend we have some queue of orders that varies over time. The callback for the "orders-
            pending" gauge will report
            // this value on-demand.
            s_ordersPending = s_rand.Next(0, 20);

            // Last we pretend that we measured how long it took to do the transaction (for example we could
            // time it with Stopwatch)
            s_orderProcessingTimeMs.Record(s_rand.Next(5, 15));
        }
    }
}

```

Run the new process and use dotnet-counters as before in a second shell to view the metrics:

```

> dotnet-counters ps
  2992 dotnet      C:\Program Files\dotnet\dotnet.exe
  20508 metric-instr E:\temp\metric-instr\bin\Debug\netcoreapp3.1\metric-instr.exe
> dotnet-counters monitor -p 20508 HatCo.HatStore
Press p to pause, r to resume, q to quit.
  Status: Running

[HatCo.HatStore]
  coats-sold (Count / 1 sec)          30
  hats-sold (Count / 1 sec)           40
  order-processing-time
    Percentile=50                   125
    Percentile=95                   146
    Percentile=99                   146
  orders-pending                   3

```

This example uses some randomly generated numbers so your values will vary a bit. You can see that `hats-sold` (the Counter) and `coats-sold` (the ObservableCounter) both show up as a rate. The ObservableGauge, `orders-pending`, appears as an absolute value. Dotnet-counters renders Histogram

instruments as three percentile statistics (50th, 95th, and 99th) but other tools may summarize the distribution differently or offer more configuration options.

Best practices

- Histograms tend to store a lot more data in memory than other metric types, however, the exact memory usage is determined by the collection tool being used. If you're defining a large number (>100) of Histogram metrics, you may need to give users guidance not to enable them all at the same time, or to configure their tools to save memory by reducing precision. Some collection tools may have hard limits on the number of concurrent Histograms they will monitor to prevent excessive memory use.
- Callbacks for all observable instruments are invoked in sequence, so any callback that takes a long time can delay or prevent all metrics from being collected. Favor quickly reading a cached value, returning no measurements, or throwing an exception over performing any potentially long-running or blocking operation.
- The [CreateObservableGauge](#) and [CreateObservableCounter](#) functions do return an instrument object, but in most cases you don't need to save it in a variable because no further interaction with the object is needed. Assigning it to a static variable as we did for the other instruments is legal but error prone, because C# static initialization is lazy and the variable is usually never referenced. Here is an example of the problem:

```
using System;
using System.Diagnostics.Metrics;

class Program
{
    // BEWARE! Static initializers only run when code in a running method refers to a static variable.
    // These statics will never be initialized because none of them were referenced in Main().
    //
    static Meter s_meter = new Meter("HatCo.HatStore", "1.0.0");
    static ObservableCounter<int> s_coatsSold = s_meter.CreateObservableCounter<int>("coats-sold", () =>
s_rand.Next(1,10));
    static Random s_rand = new Random();

    static void Main(string[] args)
    {
        Console.ReadLine();
    }
}
```

Descriptions and units

Instruments can specify optional descriptions and units. These values are opaque to all metric calculations but can be shown in collection tool UI to help engineers understand how to interpret the data. Stop the example process you started previously, and replace the example code in [Program.cs](#) with:

```

using System;
using System.Diagnostics.Metrics;
using System.Threading;

class Program
{
    static Meter s_meter = new Meter("HatCo.HatStore", "1.0.0");
    static Counter<int> s_hatsSold = s_meter.CreateCounter<int>(name: "hats-sold",
                                                               unit: "Hats",
                                                               description: "The number of hats sold in our
store");

    static void Main(string[] args)
    {
        Console.WriteLine("Press any key to exit");
        while(!Console.KeyAvailable)
        {
            // Pretend our store has a transaction each 100ms that sells 4 hats
            Thread.Sleep(100);
            s_hatsSold.Add(4);
        }
    }
}

```

Run the new process and use dotnet-counters as before in a second shell to view the metrics:

```

Press p to pause, r to resume, q to quit.
Status: Running

[HatCo.HatStore]
hats-sold (Hats / 1 sec)          40

```

dotnet-counters doesn't currently use the description text in the UI, but it does show the unit when it is provided. In this case, you see "Hats" has replaced the generic term "Count" that is visible in previous descriptions.

Best practices

The unit specified in the constructor should describe the units appropriate for an individual measurement. This will sometimes differ from the units on the final metric. In this example, each measurement is a number of hats, so "Hats" is the appropriate unit to pass in the constructor. The collection tool calculated a rate and derived on its own that the appropriate unit for the calculated metric is Hats/sec.

Multi-dimensional metrics

Measurements can also be associated with key-value pairs called tags that allow data to be categorized for analysis. For example, HatCo might want to record not only the number of hats that were sold, but also which size and color they were. When analyzing the data later, HatCo engineers can break out the totals by size, color, or any combination of both.

Counter and Histogram tags can be specified in overloads of the [Add](#) and [Record](#) that take one or more `KeyValuePair` arguments. For example:

```

s_hatsSold.Add(2,
               new KeyValuePair<string, object>("Color", "Red"),
               new KeyValuePair<string, object>("Size", 12));

```

Replace the code of `Program.cs` and rerun the app and dotnet-counters as before:

```

using System;
using System.Collections.Generic;
using System.Diagnostics.Metrics;
using System.Threading;

class Program
{
    static Meter s_meter = new Meter("HatCo.HatStore", "1.0.0");
    static Counter<int> s_hatsSold = s_meter.CreateCounter<int>("hats-sold");

    static void Main(string[] args)
    {
        Console.WriteLine("Press any key to exit");
        while(!Console.KeyAvailable)
        {
            // Pretend our store has a transaction, every 100ms, that sells 2 (size 12) red hats, and 1
            (size 19) blue hat.
            Thread.Sleep(100);
            s_hatsSold.Add(2,
                new KeyValuePair<string,object>("Color", "Red"),
                new KeyValuePair<string,object>("Size", 12));
            s_hatsSold.Add(1,
                new KeyValuePair<string,object>("Color", "Blue"),
                new KeyValuePair<string,object>("Size", 19));
        }
    }
}

```

Dotnet-counters now shows a basic categorization:

```

Press p to pause, r to resume, q to quit.
Status: Running

[HatCo.HatStore]
hats-sold (Count / 1 sec)
Color=Blue,Size=19                      9
Color=Red,Size=12                        18

```

For ObservableCounter and ObservableGauge, tagged measurements can be provided in the callback passed to the constructor:

```

using System;
using System.Collections.Generic;
using System.Diagnostics.Metrics;
using System.Threading;

class Program
{
    static Meter s_meter = new Meter("HatCo.HatStore", "1.0.0");

    static void Main(string[] args)
    {
        s_meter.CreateObservableGauge<int>("orders-pending", GetOrdersPending);
        Console.WriteLine("Press any key to exit");
        Console.ReadLine();
    }

    static IEnumerable<Measurement<int>> GetOrdersPending()
    {
        return new Measurement<int>[]
        {
            // pretend these measurements were read from a real queue somewhere
            new Measurement<int>(6, new KeyValuePair<string,object>("Country", "Italy")),
            new Measurement<int>(3, new KeyValuePair<string,object>("Country", "Spain")),
            new Measurement<int>(1, new KeyValuePair<string,object>("Country", "Mexico")),
        };
    }
}

```

When run with dotnet-counters as before, the result is:

```

Press p to pause, r to resume, q to quit.
Status: Running

[HatCo.HatStore]
  orders-pending
    Country=Italy           6
    Country=Mexico          1
    Country=Spain            3

```

Best practices

- Although the API allows any object to be used as the tag value, numeric types and strings are anticipated by collection tools. Other types may or may not be supported by a given collection tool.
- Beware of having very large or unbounded combinations of tag values being recorded in practice. Although the .NET API implementation can handle it, collection tools will likely allocate storage for metric data associated with each tag combination and this could become very large. For example, it's fine if HatCo has 10 different hat colors and 25 hat sizes for up to $10 \times 25 = 250$ sales totals to track. However, if HatCo added a third tag that's a CustomerID for the sale and they sell to 100 million customers worldwide, now there are now likely to be billions of different tag combinations being recorded. Most metric collection tools will either drop data to stay within technical limits or there can be large monetary costs to cover the data storage and processing. The implementation of each collection tool will determine its limits, but likely less than 1000 combinations for one instrument is safe. Anything above 1000 combinations will require the collection tool to apply filtering or be engineered to operate at high scale. Histogram implementations tend to use far more memory than other metrics, so safe limits could be 10-100 times lower. If you anticipate large number of unique tag combinations, then logs, transactional databases, or big data processing systems may be more appropriate solutions to operate at the needed scale.
- For instruments that will have very large numbers of tag combinations, prefer using a smaller storage

type to help reduce memory overhead. For example, storing the `short` for a `Counter<short>` only occupies 2 bytes per tag combination, whereas a `double` for `Counter<double>` occupies 8 bytes per tag combination.

- Collection tools are encouraged to optimize for code that specifies the same set of tag names in the same order for each call to record measurements on the same instrument. For high-performance code that needs to call [Add](#) and [Record](#) frequently, prefer using the same sequence of tag names for each call.
- The .NET API is optimized to be allocation-free for [Add](#) and [Record](#) calls with three or fewer tags specified individually. To avoid allocations with larger numbers of tags, use [TagList](#). In general, the performance overhead of these calls increases as more tags are used.

NOTE

OpenTelemetry refers to tags as 'attributes'. These are two different names for the same functionality.

Collect metrics

9/20/2022 • 11 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 3.1 and later versions ✓ .NET Framework 4.6.1 and later versions

Instrumented code can record numeric measurements, but the measurements usually need to be aggregated, transmitted, and stored to create useful metrics for monitoring. This process of aggregating, transmitting, and storing the data is called collection. In this tutorial, we will show several examples on how to collect metrics:

- Populating metrics in Grafana with OpenTelemetry and Prometheus.
- Viewing metrics in real time with the `dotnet-counters` command-line tool.
- Creating a custom collection tool using the underlying .NET [MeterListener API](#).

For more information about custom metric instrumentation and an overview of instrumentation options, see [Compare metric APIs](#).

Create an example application

Prerequisites: [.NET Core 3.1 SDK](#) or a later version

Before metrics can be collected, we need to produce some measurements. For simplicity, we will create a small app that has some trivial metric instrumentation. The .NET runtime also has [various metrics built-in](#). For more information about creating new metrics using the `System.Diagnostics.Metrics.Meter` API shown here, see [the instrumentation tutorial](#).

```
dotnet new console
dotnet add package System.Diagnostics.DiagnosticSource
```

Replace the code of `Program.cs` with:

```
using System;
using System.Diagnostics.Metrics;
using System.Threading;

class Program
{
    static Meter s_meter = new Meter("HatCo.HatStore", "1.0.0");
    static Counter<int> s_hatsSold = s_meter.CreateCounter<int>("hats-sold");

    static void Main(string[] args)
    {
        Console.WriteLine("Press any key to exit");
        while(!Console.KeyAvailable)
        {
            // Pretend our store has a transaction each second that sells 4 hats
            Thread.Sleep(1000);
            s_hatsSold.Add(4);
        }
    }
}
```

View metrics with `dotnet-counters`

`dotnet-counters` is a simple command-line tool that can view live metrics for any .NET Core application on

demand. It doesn't require any advance setup, which can make it useful for ad-hoc investigations or to verify that metric instrumentation is working correctly. It works with both [System.Diagnostics.Metrics](#) based APIs and [EventCounters](#).

If the [dotnet-counters](#) tool is not already installed, use the SDK to install it:

```
> dotnet tool update -g dotnet-counters
You can invoke the tool using the following command: dotnet-counters
Tool 'dotnet-counters' (version '5.0.251802') was successfully installed.
```

While the example app is still running, list the running processes in a second shell to determine the process ID:

```
> dotnet-counters ps
10180 dotnet      C:\Program Files\dotnet\dotnet.exe
19964 metric-instr E:\temp\metric-instr\bin\Debug\netcoreapp3.1\metric-instr.exe
```

Find the ID for the process name that matches the example app and have dotnet-counters monitor all metrics from the "HatCo.HatStore" meter. The meter name is case-sensitive.

```
> dotnet-counters monitor -p 19964 HatCo.HatStore
Press p to pause, r to resume, q to quit.
Status: Running

[HatCo.HatStore]
hats-sold (Count / 1 sec)          4
```

We can also run dotnet-counters specifying a different set of metrics to see some of the built-in instrumentation from the .NET runtime:

```
> dotnet-counters monitor -p 19964 System.Runtime
Press p to pause, r to resume, q to quit.
Status: Running

[System.Runtime]
% Time in GC since last GC (%)                      0
Allocation Rate (B / 1 sec)                          8,168
CPU Usage (%)                                       0
Exception Count (Count / 1 sec)                     0
GC Heap Size (MB)                                    2
Gen 0 GC Count (Count / 1 sec)                     0
Gen 0 Size (B)                                     2,216,256
Gen 1 GC Count (Count / 1 sec)                     0
Gen 1 Size (B)                                     423,392
Gen 2 GC Count (Count / 1 sec)                     0
Gen 2 Size (B)                                     203,248
LOH Size (B)                                       933,216
Monitor Lock Contention Count (Count / 1 sec)     0
Number of Active Timers                           1
Number of Assemblies Loaded                      39
ThreadPool Completed Work Item Count (Count / 1 sec) 0
ThreadPool Queue Length                           0
ThreadPool Thread Count                           3
Working Set (MB)                                    30
```

For more information about the tool, see the [dotnet-counters](#). To learn more about metrics that are available out of the box in .NET, see [built-in metrics](#).

View metrics in Grafana with OpenTelemetry and Prometheus

Prerequisites

- [.NET Core 3.1 SDK](#) or a later version

Overview

[OpenTelemetry](#) is a vendor-neutral open-source project supported by the [Cloud Native Computing Foundation](#) that aims to standardize generating and collecting telemetry for cloud-native software. The built-in platform metric APIs are designed to be compatible with this standard to make integration straightforward for any .NET developers that wish to use it. At the time of writing, support for OpenTelemetry metrics is relatively new, but [Azure Monitor](#) and many major APM vendors have endorsed it and have integration plans underway.

This example shows one of the integrations available now for OpenTelemetry metrics using the popular OSS [Prometheus](#) and [Grafana](#) projects. The metrics data will flow like this:

1. The .NET metric APIs collect measurements from our example application.
2. The OpenTelemetry library running inside the same process aggregates these measurements.
3. The Prometheus exporter library makes the aggregated data available via an HTTP metrics endpoint. 'Exporter' is what OpenTelemetry calls the libraries that transmit telemetry to vendor-specific backends.
4. A Prometheus server, potentially running on a different machine, polls the metrics endpoint, reads the data, and stores it in a database for long-term persistence. Prometheus refers to this as 'scraping' an endpoint.
5. The Grafana server, potentially running on a different machine, queries the data stored in Prometheus and displays it to engineers on a web-based monitoring dashboard.

Configure the example application to use OpenTelemetry's Prometheus exporter

Add a reference to the OpenTelemetry Prometheus exporter to the example application:

```
dotnet add package OpenTelemetry.Exporter.Prometheus --version 1.2.0-beta1
```

NOTE

The Prometheus exporter library includes a reference to OpenTelemetry's shared library so this command implicitly adds both libraries to the application.

NOTE

This tutorial is using a pre-release build of OpenTelemetry's Prometheus support available at the time of writing. The OpenTelemetry project maintainers might make changes prior to the official release.

Modify the code of [Program.cs](#) so that it contains the extra code to configure OpenTelemetry at the beginning of Main():

```

using System;
using System.Diagnostics.Metrics;
using System.Threading;
using OpenTelemetry;
using OpenTelemetry.Metrics;

class Program
{
    static Meter s_meter = new Meter("HatCo.HatStore", "1.0.0");
    static Counter<int> s_hatsSold = s_meter.CreateCounter<int>(name: "hats-sold",
                                                               unit: "Hats",
                                                               description: "The number of hats sold in our
store");

    static void Main(string[] args)
    {
        using MeterProvider meterProvider = Sdk.CreateMeterProviderBuilder()
            .AddMeter("HatCo.HatStore")
            .AddPrometheusExporter(opt =>
            {
                opt.StartHttpListener = true;
                opt.HttpListenerPrefixes = new string[] { $"http://localhost:9184/" };
            })
            .Build();

        Console.WriteLine("Press any key to exit");
        while(!Console.KeyAvailable)
        {
            // Pretend our store has a transaction each second that sells 4 hats
            Thread.Sleep(1000);
            s_hatsSold.Add(4);
        }
    }
}

```

`AddMeter("HatCo.HatStore")` configures OpenTelemetry to transmit all the metrics collected by the Meter our app defined. `AddPrometheusExporter(...)` configures OpenTelemetry to expose Prometheus' metrics endpoint on port 9184 and to use the HttpListener. See the [OpenTelemetry documentation](#) for more information about OpenTelemetry configuration options, in particular, alternative hosting options that are useful for ASP.NET applications.

NOTE

At the time of writing, OpenTelemetry only supports metrics emitted using the [System.Diagnostics.Metrics](#) APIs. However, support for [EventCounters](#) is planned.

Run the example app and leave it running in the background.

```

> dotnet run
Press any key to exit

```

Set up and configure Prometheus

Follow the [Prometheus first steps](#) to set up your Prometheus server and confirm it is working.

Modify the `prometheus.yml` configuration file so that Prometheus will scrape the metrics endpoint that our example app is exposing. Add this text in the `scrape_configs` section:

```

- job_name: 'OpenTelemetryTest'
scrape_interval: 1s # poll very quickly for a more responsive demo
static_configs:
  - targets: ['localhost:9184']

```

If you are starting from the default configuration, then `scrape_configs` should now look like this:

```

scrape_configs:
# The job name is added as a label `job=<job_name>` to any timeseries scraped from this config.
- job_name: "prometheus"

# metrics_path defaults to '/metrics'
# scheme defaults to 'http'.

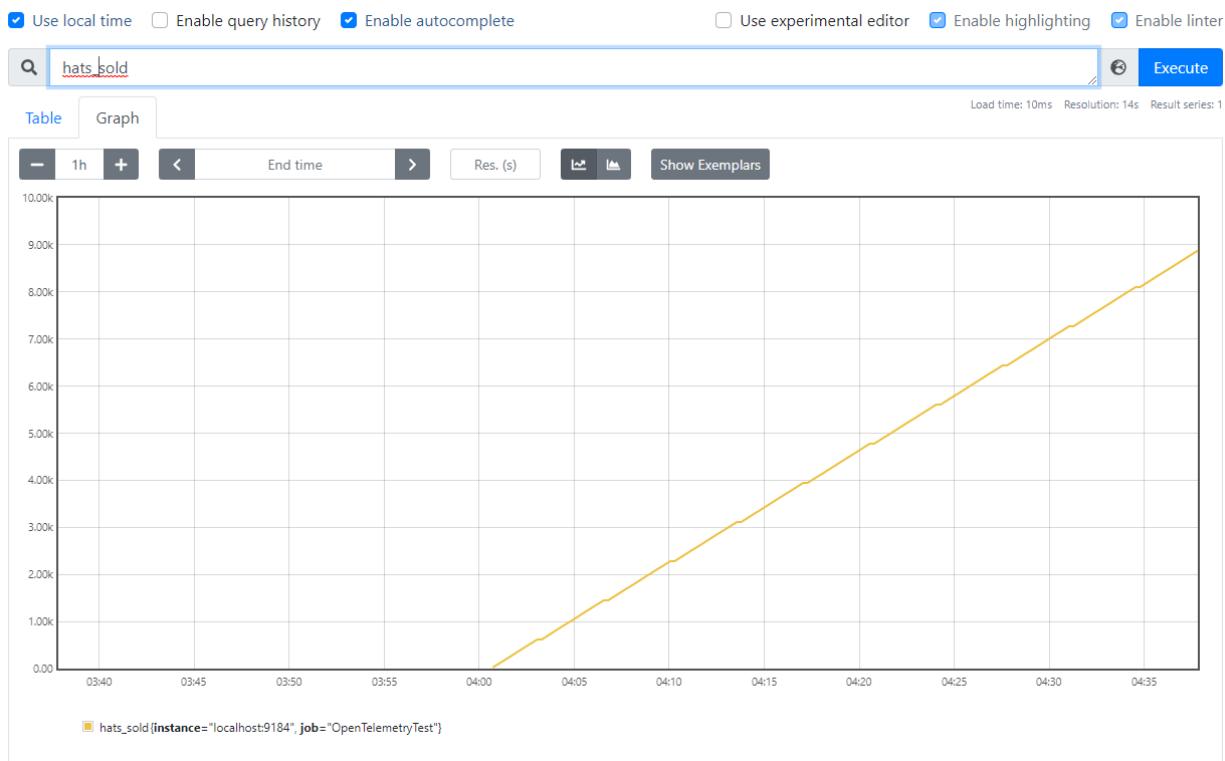
static_configs:
  - targets: ["localhost:9090"]

- job_name: 'OpenTelemetryTest'
scrape_interval: 1s # poll very quickly for a more responsive demo
static_configs:
  - targets: ['localhost:9184']

```

Reload the configuration or restart the Prometheus server, then confirm that OpenTelemetryTest is in the UP state in the **Status > Targets** page of the Prometheus web portal.

On the Graph page of the Prometheus web portal, enter `hats_sold` in the expression text box. In the graph tab, Prometheus should show the steadily increasing value of the "hats-sold" Counter that is being emitted by our example application.

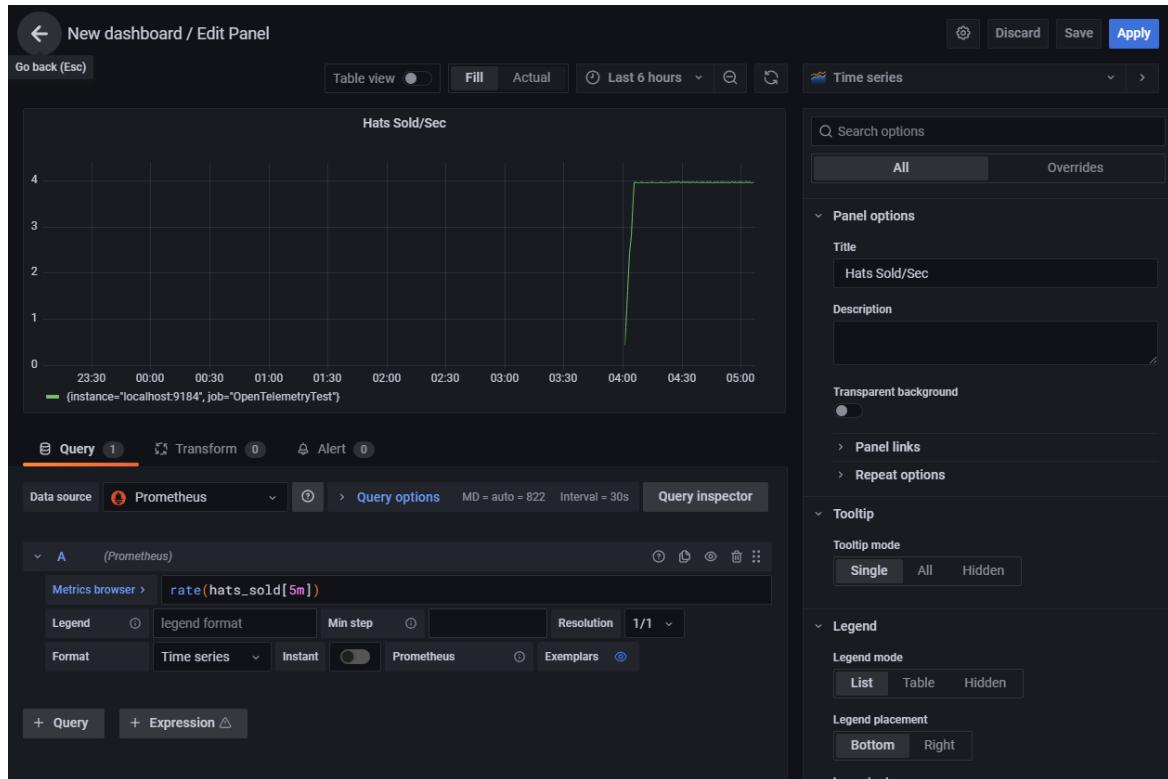


If the Prometheus server hasn't been scraping the example app for long, you may need to wait a short while for data to accumulate. You can also adjust the time range control in the upper left to "1m" (1 minute) to get a better view of very recent data.

Show metrics on a Grafana dashboard

1. Follow [the standard instructions](#) to install Grafana and connect it to a Prometheus data source.

2. Create a Grafana dashboard by clicking the + icon on the left toolbar in the Grafana web portal, then select **Dashboard**. In the dashboard editor that appears, enter 'Hats Sold/Sec' as the Title and 'rate(hats_sold[5m])' in the PromQL expression field. It should look like this:



3. Click **Apply** to save and view the simple new dashboard.



Create a custom collection tool using the .NET MeterListener API

The .NET [MeterListener](#) API allows creating custom in-process logic to observe the measurements being recorded by [System.Diagnostics.Metrics.Meter](#). For guidance creating custom logic compatible with the older EventCounters instrumentation, see [EventCounters](#).

Modify the code of `Program.cs` to use [MeterListener](#) like this:

```

using System;
using System.Collections.Generic;
using System.Diagnostics.Metrics;
using System.Threading;

class Program
{
    static Meter s_meter = new Meter("HatCo.HatStore", "1.0.0");
    static Counter<int> s_hatsSold = s_meter.CreateCounter<int>(name: "hats-sold",
                                                               unit: "Hats",
                                                               description: "The number of hats sold in our
store");

    static void Main(string[] args)
    {
        using MeterListener meterListener = new MeterListener();
        meterListener.InstrumentPublished = (instrument, listener) =>
        {
            if(instrument.Meter.Name == "HatCo.HatStore")
            {
                listener.EnableMeasurementEvents(instrument);
            }
        };
        meterListener.SetMeasurementEventCallback<int>(OnMeasurementRecorded);
        meterListener.Start();

        Console.WriteLine("Press any key to exit");
        while(!Console.KeyAvailable)
        {
            // Pretend our store has a transaction each second that sells 4 hats
            Thread.Sleep(1000);
            s_hatsSold.Add(4);
        }
    }

    static void OnMeasurementRecorded<T>(Instrument instrument, T measurement,
    ReadOnlySpan<KeyValuePair<string,object>> tags, object state)
    {
        Console.WriteLine($"{instrument.Name} recorded measurement {measurement}");
    }
}

```

When run, the application now runs our custom callback on each measurement:

```

> dotnet run
Press any key to exit
hats-sold recorded measurement 4
hats-sold recorded measurement 4
hats-sold recorded measurement 4
hats-sold recorded measurement 4
...

```

Let's break down what happens in the example above.

```
using MeterListener meterListener = new MeterListener();
```

First we created an instance of the [MeterListener](#), which we will use to receive measurements.

```
meterListener.InstrumentPublished = (instrument, listener) =>
{
    if(instrument.Meter.Name == "HatCo.HatStore")
    {
        listener.EnableMeasurementEvents(instrument);
    }
};
```

Here we configured which instruments the listener will receive measurements from. `InstrumentPublished` is a delegate that will be invoked anytime a new instrument is created within the app. Our delegate can examine the instrument, such as checking the name, the Meter, or any other public property to decide whether to subscribe. If we do want to receive measurements from this instrument, then we invoke `EnableMeasurementEvents` to indicate that. If your code has another way to obtain a reference to an instrument, it's legal to invoke `EnableMeasurementEvents()` at any time with that reference, but this is probably uncommon.

```
meterListener.SetMeasurementEventCallback<int>(OnMeasurementRecorded);
...
static void OnMeasurementRecorded<T>(Instrument instrument, T measurement,
ReadOnlySpan<KeyValuePair<string,object>> tags, object state)
{
    Console.WriteLine($"{instrument.Name} recorded measurement {measurement}");
}
```

Next we configured the delegate that is invoked when measurements are received from an instrument by calling `SetMeasurementEventCallback`. The generic parameter controls which data type of measurement will be received by the callback. For example, a `Counter<int>` generates `int` measurements whereas a `Counter<double>` generates `double` measurements. Instruments can be created with `byte`, `short`, `int`, `long`, `float`, `double`, and `decimal` types. We recommend registering a callback for every data type unless you have scenario-specific knowledge that not all data types will be needed, such as in this example. Making repeated calls to `SetMeasurementEventCallback()` with different generic arguments may appear a little unusual. The API is designed this way to allow MeterListeners to receive measurements with extremely low performance overhead, typically just a few nanoseconds.

When `MeterListener.EnableMeasurementEvents()` was called initially, there was an opportunity to provide a `state` object as one of the parameters. That object can be anything you want. If you provide a state object in that call, then it will be stored with that instrument and returned to you as the `state` parameter in the callback. This is intended both as a convenience and as a performance optimization. Often listeners need to create an object for each instrument that will store measurements in memory and have code to do calculations on those measurements. Although you could create a Dictionary that maps from the instrument to the storage object and look it up on every measurement, that would be much slower than accessing it from `state`.

```
meterListener.Start();
```

Once the `MeterListener` is configured, we need to start it to trigger callbacks to begin. The `InstrumentPublished` delegate will be invoked for every pre-existing Instrument in the process. In the future, any newly created Instrument will also trigger `InstrumentPublished` to be invoked.

```
using MeterListener meterListener = new MeterListener();
```

Once we are done listening, disposing the listener stops the flow of callbacks and releases any internal references to the listener object. The `using` keyword we used when declaring `meterListener` causes `Dispose()` to be called automatically when the variable goes out of scope. Be aware that `Dispose()` is only promising that it won't initiate new callbacks. Because callbacks occur on different threads, there may still be callbacks in

progress after the call to `Dispose()` returns. If you need a guarantee that a certain region of code in your callback isn't currently executing and will never execute again in the future, then you will need some additional thread synchronization to enforce that. `Dispose()` doesn't include the synchronization by default because it adds performance overhead in every measurement callback—and `MeterListener` is designed as a highly performance conscious API.

EventCounters in .NET

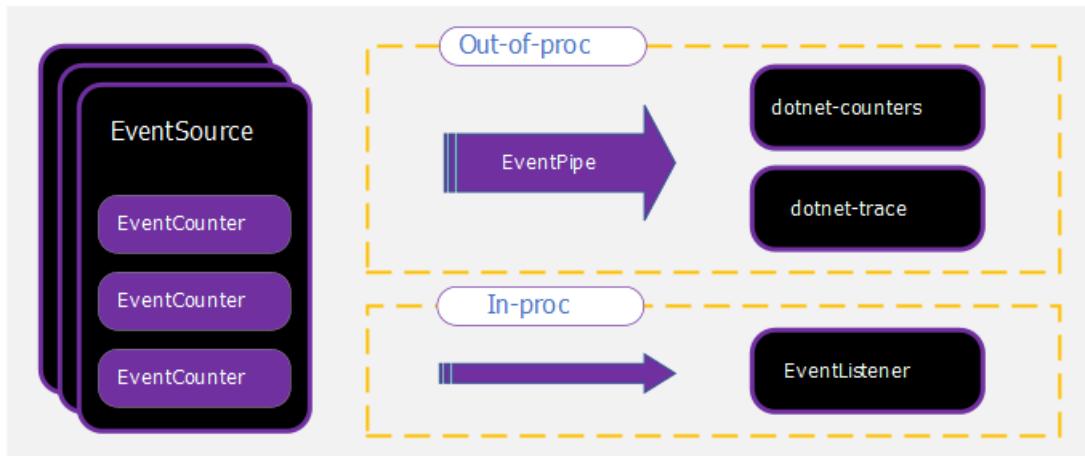
9/20/2022 • 10 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 3.0 SDK and later versions

EventCounters are .NET APIs used for lightweight, cross-platform, and near real-time performance metric collection. EventCounters were added as a cross-platform alternative to the "performance counters" of .NET Framework on Windows. In this article, you'll learn what EventCounters are, how to implement them, and how to consume them.

The .NET runtime and a few .NET libraries publish basic diagnostics information using EventCounters starting in .NET Core 3.0. Apart from the EventCounters that are provided by the .NET runtime, you may choose to implement your own EventCounters. EventCounters can be used to track various metrics. Learn more about them in [well-known EventCounters in .NET](#)

EventCounters live as a part of an [EventSource](#), and are automatically pushed to listener tools on a regular basis. Like all other events on an [EventSource](#), they can be consumed both in-proc and out-of-proc via [EventListener](#) and [EventPipe](#). This article focuses on the cross-platform capabilities of EventCounters, and intentionally excludes PerfView and ETW (Event Tracing for Windows) - although both can be used with EventCounters.



EventCounter API overview

There are two primary categories of EventCounters. Some counters are for "rate" values, such as total number of exceptions, total number of GCs, and total number of requests. Other counters are "snapshot" values, such as heap usage, CPU usage, and working set size. Within each of these categories of counters, there are two types of counters that vary by how they get their value. Polling counters retrieve their value via a callback, and non-polling counters have their values directly set on the counter instance.

The counters are represented by the following implementations:

- [EventCounter](#)
- [IncrementingEventCounter](#)
- [PollingCounter](#)
- [IncrementingPollingCounter](#)

An event listener specifies how long measurement intervals are. At the end of each interval a value is transmitted to the listener for each counter. The implementations of a counter determine what APIs and

calculations are used to produce the value each interval.

- The [EventCounter](#) records a set of values. The [EventCounter.WriteMetric](#) method adds a new value to the set. With each interval, a statistical summary for the set is computed, such as the min, max, and mean. The [dotnet-counters](#) tool will always display the mean value. The [EventCounter](#) is useful to describe a discrete set of operations. Common usage may include monitoring the average size in bytes of recent IO operations, or the average monetary value of a set of financial transactions.
- The [IncrementingEventCounter](#) records a running total for each time interval. The [IncrementingEventCounter.Increment](#) method adds to the total. For example, if `Increment()` is called three times during one interval with values `1`, `2`, and `5`, then the running total of `8` will be reported as the counter value for this interval. The [dotnet-counters](#) tool will display the rate as the recorded total / time. The [IncrementingEventCounter](#) is useful to measure how frequently an action is occurring, such as the number of requests processed per second.
- The [PollingCounter](#) uses a callback to determine the value that is reported. With each time interval, the user provided callback function is invoked and the return value is used as the counter value. A [PollingCounter](#) can be used to query a metric from an external source, for example getting the current free bytes on a disk. It can also be used to report custom statistics that can be computed on demand by an application. Examples include reporting the 95th percentile of recent request latencies, or the current hit or miss ratio of a cache.
- The [IncrementingPollingCounter](#) uses a callback to determine the reported increment value. With each time interval, the callback is invoked, and then the difference between the current invocation, and the last invocation is the reported value. The [dotnet-counters](#) tool will always display the difference as a rate, the reported value / time. This counter is useful when it isn't feasible to call an API on each occurrence, but it's possible to query the total number of occurrences. For example, you could report the number of bytes written to a file per second, even without a notification each time a byte is written.

Implement an EventSource

The following code implements a sample [EventSource](#) exposed as the named `"Sample.EventCounter.Minimal"` provider. This source contains an [EventCounter](#) representing request-processing time. Such a counter has a name (that is, its unique ID in the source) and a display name, both used by listener tools such as [dotnet-counter](#).

```

using System.Diagnostics.Tracing;

[EventSource(Name = "Sample.EventCounter.Minimal")]
public sealed class MinimalEventCounterSource : EventSource
{
    public static readonly MinimalEventCounterSource Log = new MinimalEventCounterSource();

    private EventCounter _requestCounter;

    private MinimalEventCounterSource() =>
        _requestCounter = new EventCounter("request-time", this)
        {
            DisplayName = "Request Processing Time",
            DisplayUnits = "ms"
        };

    public void Request(string url, long elapsedMilliseconds)
    {
        WriteEvent(1, url, elapsedMilliseconds);
        _requestCounter?.WriteMetric(elapsedMilliseconds);
    }

    protected override void Dispose(bool disposing)
    {
        _requestCounter?.Dispose();
        _requestCounter = null;

        base.Dispose(disposing);
    }
}

```

You use `dotnet-counters ps` to display a list of .NET processes that can be monitored:

```

dotnet-counters ps
 1398652 dotnet      C:\Program Files\dotnet\dotnet.exe
 1399072 dotnet      C:\Program Files\dotnet\dotnet.exe
 1399112 dotnet      C:\Program Files\dotnet\dotnet.exe
 1401880 dotnet      C:\Program Files\dotnet\dotnet.exe
 1400180 sample-counters C:\sample-counters\bin\Debug\netcoreapp3.1\sample-counters.exe

```

Pass the `EventSource` name to the `--counters` option to start monitoring your counter:

```
dotnet-counters monitor --process-id 1400180 --counters Sample.EventCounter.Minimal
```

The following example shows monitor output:

```

Press p to pause, r to resume, q to quit.
Status: Running

[Samples-EventCounterDemos-Minimal]
Request Processing Time (ms)          0.445

```

Press `q` to stop the monitoring command.

Conditional counters

When implementing an `EventSource`, the containing counters can be conditionally instantiated when the `EventSource.OnEventCommand` method is called with a `Command` value of `EventCommand.Enable`. To safely instantiate a counter instance only if it is `null`, use the [null-coalescing assignment operator](#). Additionally, custom methods can evaluate the `IsEnabled` method to determine whether or not the current event source is

enabled.

```
using System.Diagnostics.Tracing;

[EventSource(Name = "Sample.EventCounter.Conditional")]
public sealed class ConditionalEventCounterSource : EventSource
{
    public static readonly ConditionalEventCounterSource Log = new ConditionalEventCounterSource();

    private EventCounter _requestCounter;

    private ConditionalEventCounterSource() { }

    protected override void OnEventCommand(EventCommandEventEventArgs args)
    {
        if (args.Command == EventCommand.Enable)
        {
            _requestCounter ??= new EventCounter("request-time", this)
            {
                DisplayName = "Request Processing Time",
                DisplayUnits = "ms"
            };
        }
    }

    public void Request(string url, float elapsedMilliseconds)
    {
        if (IsEnabled())
        {
            _requestCounter?.WriteMetric(elapsedMilliseconds);
        }
    }

    protected override void Dispose(bool disposing)
    {
        _requestCounter?.Dispose();
        _requestCounter = null;

        base.Dispose(disposing);
    }
}
```

TIP

Conditional counters are counters that are conditionally instantiated, a micro-optimization. The runtime adopts this pattern for scenarios where counters are normally not used, to save a fraction of a millisecond.

.NET Core runtime example counters

There are many great example implementations in the .NET Core runtime. Here is the runtime implementation for the counter that tracks the working set size of the application.

```
var workingSetCounter = new PollingCounter(
    "working-set",
    this,
    () => (double)(Environment.WorkingSet / 1_000_000))
{
    DisplayName = "Working Set",
    DisplayUnits = "MB"
};
```

The [PollingCounter](#) reports the current amount of physical memory mapped to the process (working set) of the

app, since it captures a metric at a moment in time. The callback for polling a value is the provided lambda expression, which is just a call to the `System.Environment.WorkingSet` API. `DisplayName` and `DisplayUnits` are optional properties that can be set to help the consumer side of the counter to display the value more clearly. For example, `dotnet-counters` uses these properties to display the more display-friendly version of the counter names.

IMPORTANT

The `DisplayName` properties are not localized.

For the `PollingCounter`, and the `IncrementingPollingCounter`, nothing else needs to be done. They both poll the values themselves at an interval requested by the consumer.

Here is an example of a runtime counter implemented using `IncrementingPollingCounter`.

```
var monitorContentionCounter = new IncrementingPollingCounter(
    "monitor-lock-contention-count",
    this,
    () => Monitor.LockContentionCount
)
{
    DisplayName = "Monitor Lock Contention Count",
    DisplayRateTimeScale = TimeSpan.FromSeconds(1)
};
```

The `IncrementingPollingCounter` uses the `Monitor.LockContentionCount` API to report the increment of the total lock contention count. The `DisplayRateTimeScale` property is optional, but when used it can provide a hint for what time interval the counter is best displayed at. For example, the lock contention count is best displayed as *count per second*, so its `DisplayRateTimeScale` is set to one second. The display rate can be adjusted for different types of rate counters.

NOTE

The `DisplayRateTimeScale` is *not* used by `dotnet-counters`, and event listeners are not required to use it.

There are more counter implementations to use as a reference in the [.NET runtime](#) repo.

Concurrency

TIP

The EventCounters API does not guarantee thread safety. When the delegates passed to `PollingCounter` or `IncrementingPollingCounter` instances are called by multiple threads, it's your responsibility to guarantee the delegates' thread-safety.

For example, consider the following `EventSource` to keep track of requests.

```

using System;
using System.Diagnostics.Tracing;

public class RequestEventSource : EventSource
{
    public static readonly RequestEventSource Log = new RequestEventSource();

    private IncrementingPollingCounter _requestRateCounter;
    private long _requestCount = 0;

    private RequestEventSource() =>
        _requestRateCounter = new IncrementingPollingCounter("request-rate", this, () => _requestCount)
    {
        DisplayName = "Request Rate",
        DisplayRateTimeScale = TimeSpan.FromSeconds(1)
    };

    public void AddRequest() => ++_requestCount;

    protected override void Dispose(bool disposing)
    {
        _requestRateCounter?.Dispose();
        _requestRateCounter = null;

        base.Dispose(disposing);
    }
}

```

The `AddRequest()` method can be called from a request handler, and the `RequestRateCounter` polls the value at the interval specified by the consumer of the counter. However, the `AddRequest()` method can be called by multiple threads at once, putting a race condition on `_requestCount`. A thread-safe alternative way to increment the `_requestCount` is to use [Interlocked.Increment](#).

```
public void AddRequest() => Interlocked.Increment(ref _requestCount);
```

To prevent torn reads (on 32-bit architectures) of the `long`-field `_requestCount` use [Interlocked.Read](#).

```

_requestRateCounter = new IncrementingPollingCounter("request-rate", this, () => Interlocked.Read(ref
_requestCount))
{
    DisplayName = "Request Rate",
    DisplayRateTimeScale = TimeSpan.FromSeconds(1)
};

```

Consume EventCounters

There are two primary ways of consuming EventCounters: in-proc and out-of-proc. The consumption of EventCounters can be distinguished into three layers of various consuming technologies.

- Transporting events in a raw stream via ETW or EventPipe:

ETW APIs come with the Windows OS, and EventPipe is accessible as a [.NET API](#), or the diagnostic [IPC protocol](#).

- Decoding the binary event stream into events:

The [TraceEvent library](#) handles both ETW and EventPipe stream formats.

- Command-line and GUI tools:

Tools like PerfView (ETW or EventPipe), dotnet-counters (EventPipe only), and dotnet-monitor (EventPipe only).

Consume out-of-proc

Consuming EventCounters out-of-proc is a common approach. You can use [dotnet-counters](#) to consume them in a cross-platform manner via an EventPipe. The `dotnet-counters` tool is a cross-platform dotnet CLI global tool that can be used to monitor the counter values. To find out how to use `dotnet-counters` to monitor your counters, see [dotnet-counters](#), or work through the [Measure performance using EventCounters](#) tutorial.

dotnet-trace

The `dotnet-trace` tool can be used to consume the counter data through an EventPipe. Here is an example using `dotnet-trace` to collect counter data.

```
dotnet-trace collect --process-id <pid> Sample.EventCounter.Minimal:0:0:EventCounterIntervalSec=1
```

For more information on how to collect counter values over time, see the [dotnet-trace](#) documentation.

Azure Application Insights

EventCounters can be consumed by Azure Monitor, specifically Azure Application Insights. Counters can be added and removed, and you're free to specify custom counters, or well-known counters. For more information, see [Customizing counters to be collected](#).

dotnet-monitor

The `dotnet-monitor` tool makes it easier to access diagnostics from a .NET process in a remote and automated fashion. In addition to traces, it can monitor metrics, collect memory dumps, and collect GC dumps. It's distributed as both a CLI tool and a docker image. It exposes a REST API, and the collection of diagnostic artifacts occurs through REST calls.

For more information, see [dotnet-monitor](#).

Consume in-proc

You can consume the counter values via the [EventListener](#) API. An [EventListener](#) is an in-proc way of consuming any events written by all instances of an [EventSource](#) in your application. For more information on how to use the `EventListener` API, see [EventListener](#).

First, the [EventSource](#) that produces the counter value needs to be enabled. Override the `EventListener.OnEventSourceCreated` method to get a notification when an [EventSource](#) is created, and if this is the correct [EventSource](#) with your EventCounters, then you can call `EventListener.EnableEvents` on it. Here is an example override:

```
protected override void OnEventSourceCreated(EventSource source)
{
    if (!source.Name.Equals("System.Runtime"))
    {
        return;
    }

    EnableEvents(source, EventLevel.Verbose, EventKeywords.All, new Dictionary<string, string>()
    {
        ["EventCounterIntervalSec"] = "1"
    });
}
```

Sample code

Here is a sample [EventListener](#) class that prints all the counter names and values from the .NET runtime's [EventSource](#), for publishing its internal counters (`System.Runtime`) every second.

```

using System;
using System.Collections.Generic;
using System.Diagnostics.Tracing;

public class SimpleEventListener : EventListener
{
    public SimpleEventListener()
    {
    }

    protected override void OnEventSourceCreated(EventSource source)
    {
        if (!source.Name.Equals("System.Runtime"))
        {
            return;
        }

        EnableEvents(source, EventLevel.Verbose, EventKeywords.All, new Dictionary<string, string>()
        {
            ["EventCounterIntervalSec"] = "1"
        });
    }

    protected override void OnEventWritten(EventWrittenEventArgs eventData)
    {
        if (!eventData.EventName.Equals("EventCounters"))
        {
            return;
        }

        for (int i = 0; i < eventData.Payload.Count; ++ i)
        {
            if (eventData.Payload[i] is IDictionary<string, object> eventPayload)
            {
                var (counterName, counterValue) = GetRelevantMetric(eventPayload);
                Console.WriteLine($"{counterName} : {counterValue}");
            }
        }
    }

    private static (string counterName, string counterValue) GetRelevantMetric(
        IDictionary<string, object> eventPayload)
    {
        var counterName = "";
        var counterValue = "";

        if (eventPayload.TryGetValue("DisplayName", out object displayValue))
        {
            counterName = displayValue.ToString();
        }
        if (eventPayload.TryGetValue("Mean", out object value) ||
            eventPayload.TryGetValue("Increment", out value))
        {
            counterValue = value.ToString();
        }

        return (counterName, counterValue);
    }
}

```

As shown above, you *must* make sure the `"EventCounterIntervalSec"` argument is set in the `filterPayload` argument when calling `EnableEvents`. Otherwise the counters will not be able to flush out values since it doesn't know at which interval it should be getting flushed out.

See also

- [dotnet-counters](#)
- [dotnet-trace](#)
- [EventCounter](#)
- [EventListener](#)
- [EventSource](#)

Well-known EventCounters in .NET

9/20/2022 • 7 minutes to read • [Edit Online](#)

The .NET runtime and libraries implement and publish several [EventCounters](#) that can be used to identify and diagnose various performance issues. This article is a reference on the providers that can be used to monitor these counters and their descriptions.

System.Runtime counters

The following counters are published as part of .NET runtime (CoreCLR) and are maintained in the

[RuntimeEventSource.cs](#).

COUNTER	DESCRIPTION	FIRST AVAILABLE IN
% Time in GC since last GC (<code>time-in-gc</code>)	The percent of time in GC since the last GC	.NET Core 3.1
Allocation Rate (<code>alloc-rate</code>)	The number of bytes allocated per update interval	.NET Core 3.1
CPU Usage (<code>cpu-usage</code>)	The percent of the process's CPU usage relative to all of the system CPU resources	.NET Core 3.1
Exception Count (<code>exception-count</code>)	The number of exceptions that have occurred	.NET Core 3.1
GC Heap Size (<code>gc-heap-size</code>)	The number of megabytes thought to be allocated based on GC.GetTotalMemory(Boolean)	.NET Core 3.1
Gen 0 GC Count (<code>gen-0-gc-count</code>)	The number of times GC has occurred for Gen 0 per update interval	.NET Core 3.1
Gen 0 Size (<code>gen-0-size</code>)	The number of bytes for Gen 0 GC	.NET Core 3.1
Gen 1 GC Count (<code>gen-1-gc-count</code>)	The number of times GC has occurred for Gen 1 per update interval	.NET Core 3.1
Gen 1 Size (<code>gen-1-size</code>)	The number of bytes for Gen 1 GC	.NET Core 3.1
Gen 2 GC Count (<code>gen-2-gc-count</code>)	The number of times GC has occurred for Gen 2 per update interval	.NET Core 3.1
Gen 2 Size (<code>gen-2-size</code>)	The number of bytes for Gen 2 GC	.NET Core 3.1
LOH Size (<code>loh-size</code>)	The number of bytes for the large object heap	.NET Core 3.1

COUNTER	DESCRIPTION	FIRST AVAILABLE IN
POH Size (<code>poh-size</code>)	The number of bytes for the pinned object heap (available on .NET 5 and later versions)	.NET Core 3.1
GC Fragmentation (<code>gc-fragmentation</code>)	The GC Heap Fragmentation (available on .NET 5 and later versions)	.NET Core 3.1
Monitor Lock Contention Count (<code>monitor-lock-contention-count</code>)	The number of times there was contention when trying to take the monitor's lock, based on Monitor.LockContentionCount	.NET Core 3.1
Number of Active Timers (<code>active-timer-count</code>)	The number of Timer instances that are currently active, based on Timer.ActiveCount	.NET Core 3.1
Number of Assemblies Loaded (<code>assembly-count</code>)	The number of Assembly instances loaded into a process at a point in time	.NET Core 3.1
ThreadPool Completed Work Item Count (<code>threadpool-completed-items-count</code>)	The number of work items that have been processed so far in the ThreadPool	.NET Core 3.1
ThreadPool Queue Length (<code>threadpool-queue-length</code>)	The number of work items that are currently queued to be processed in the ThreadPool	.NET Core 3.1
ThreadPool Thread Count (<code>threadpool-thread-count</code>)	The number of thread pool threads that currently exist in the ThreadPool , based on ThreadPool.ThreadCount	.NET Core 3.1
Working Set (<code>working-set</code>)	The number of megabytes of physical memory mapped to the process context at a point in time base on Environment.WorkingSet	.NET Core 3.1
IL Bytes Jitted (<code>il-bytes-jitted</code>)	The total size of ILs that are JIT-compiled, in bytes	.NET 5
Method Jitted Count (<code>method-jitted-count</code>)	The number of methods that are JIT-compiled	.NET 5
GC Committed Bytes (<code>gc-committed</code>)	The number of bytes committed by the GC	.NET 6

Microsoft.AspNetCore.Hosting counters

The following counters are published as part of [ASP.NET Core](#) and are maintained in [HostingEventSource.cs](#).

COUNTER	DESCRIPTION	FIRST AVAILABLE IN
Current Requests (<code>current-requests</code>)	The total number of requests that have started, but not yet stopped	.NET Core 3.1

COUNTER	DESCRIPTION	FIRST AVAILABLE IN
Failed Requests (<code>failed-requests</code>)	The total number of failed requests that have occurred for the life of the app	.NET Core 3.1
Request Rate (<code>requests-per-second</code>)	The number of requests that occur per update interval	.NET Core 3.1
Total Requests (<code>total-requests</code>)	The total number of requests that have occurred for the life of the app	.NET Core 3.1

Microsoft.AspNetCore.Http.Connections counters

The following counters are published as part of [ASP.NET Core SignalR](#) and are maintained in [`HttpConnectionsEventSource.cs`](#).

COUNTER	DESCRIPTION	FIRST AVAILABLE IN
Average Connection Duration (<code>connections-duration</code>)	The average duration of a connection in milliseconds	.NET Core 3.1
Current Connections (<code>current-connections</code>)	The number of active connections that have started, but not yet stopped	.NET Core 3.1
Total Connections Started (<code>connections-started</code>)	The total number of connections that have started	.NET Core 3.1
Total Connections Stopped (<code>connections-stopped</code>)	The total number of connections that have stopped	.NET Core 3.1
Total Connections Timed Out (<code>connections-timed-out</code>)	The total number of connections that have timed out	.NET Core 3.1

Microsoft-AspNetCore-Server-Kestrel counters

The following counters are published as part of the [ASP.NET Core Kestrel web server](#) and are maintained in [`KestrelEventSource.cs`](#).

COUNTER	DESCRIPTION	FIRST AVAILABLE IN
Connection Queue Length (<code>connection-queue-length</code>)	The current length of the connection queue	.NET 5
Connection Rate (<code>connections-per-second</code>)	The number of connections per update interval to the web server	.NET 5
Current Connections (<code>current-connections</code>)	The current number of active connections to the web server	.NET 5
Current TLS Handshakes (<code>current-tls-handshakes</code>)	The current number of TLS handshakes	.NET 5

COUNTER	DESCRIPTION	FIRST AVAILABLE IN
Current Upgraded Requests (WebSockets) (<code>current-upgraded-requests</code>)	The current number of upgraded requests (WebSockets)	.NET 5
Failed TLS Handshakes (<code>failed-tls-handshakes</code>)	The total number of failed TLS handshakes	.NET 5
Request Queue Length (<code>request-queue-length</code>)	The current length of the request queue	.NET 5
TLS Handshake Rate (<code>tls-handshakes-per-second</code>)	The number of TLS handshakes per update interval	.NET 5
Total Connections (<code>total-connections</code>)	The total number of connections to the web server	.NET 5
Total TLS Handshakes (<code>total-tls-handshakes</code>)	The total number of TLS handshakes with the web server	.NET 5

System.Net.Http counters

The following counters are published by the HTTP stack.

COUNTER	DESCRIPTION	FIRST AVAILABLE IN
Requests Started (<code>requests-started</code>)	The number of requests started since the process started	.NET 5
Requests Started Rate (<code>requests-started-rate</code>)	The number of requests started per update interval	.NET 5
Requests Failed (<code>requests-failed</code>)	The number of failed requests since the process started	.NET 5
Requests Failed Rate (<code>requests-failed-rate</code>)	The number of failed requests per update interval	.NET 5
Current Requests (<code>current-requests</code>)	Current number of active HTTP requests that have started but not yet completed or failed	.NET 5
Current HTTP 1.1 Connections (<code>http11-connections-current-total</code>)	The current number of HTTP 1.1 connections that have started but not yet completed or failed	.NET 5
Current HTTP 2.0 Connections (<code>http20-connections-current-total</code>)	The current number of HTTP 2.0 connections that have started but not yet completed or failed	.NET 5
HTTP 1.1 Requests Queue Duration (<code>http11-requests-queue-duration</code>)	The average duration of the time HTTP 1.1 requests spent in the request queue	.NET 5

COUNTER	DESCRIPTION	FIRST AVAILABLE IN
HTTP 2.0 Requests Queue Duration (<code>http20-requests-queue-duration</code>)	The average duration of the time HTTP 2.0 requests spent in the request queue	.NET 5

System.Net.NameResolution counters

The following counters track metrics related to DNS lookups.

COUNTER	DESCRIPTION	FIRST AVAILABLE IN
DNS Lookups Requested (<code>dns-lookups-requested</code>)	The number of DNS lookups requested since the process started	.NET 5
Average DNS Lookup Duration (<code>dns-lookups-duration</code>)	The average time taken for a DNS lookup	.NET 5

System.Net.Security counters

The following counters track metrics related to the Transport Layer Security protocol.

COUNTER	DESCRIPTION	FIRST AVAILABLE IN
TLS handshakes completed (<code>tls-handshake-rate</code>)	The number of TLS handshakes completed per update interval	.NET 5
Total TLS handshakes completed (<code>total-tls-handshakes</code>)	The total number of TLS handshakes completed since the process started	.NET 5
Current TLS handshakes (<code>current-tls-handshakes</code>)	The current number of TLS handshakes that have started but not yet completed	.NET 5
Total TLS handshakes failed (<code>failed-tls-handshakes</code>)	The total number of failed TLS handshakes since the process started	.NET 5
All TLS Sessions Active (<code>all-tls-sessions-open</code>)	The number of active TLS sessions of any version	.NET 5
TLS 1.0 Sessions Active (<code>tls10-sessions-open</code>)	The number of active TLS 1.0 sessions	.NET 5
TLS 1.1 Sessions Active (<code>tls11-sessions-open</code>)	The number of active TLS 1.1 sessions	.NET 5
TLS 1.2 Sessions Active (<code>tls12-sessions-open</code>)	The number of active TLS 1.2 sessions	.NET 5
TLS 1.3 Sessions Active (<code>tls13-sessions-open</code>)	The number of active TLS 1.3 sessions	.NET 5

COUNTER	DESCRIPTION	FIRST AVAILABLE IN
TLS Handshake Duration (<code>all-tls-handshake-duration</code>)	The average duration of all TLS handshakes	.NET 5
TLS 1.0 Handshake Duration (<code>tls10-handshake-duration</code>)	The average duration of TLS 1.0 handshakes	.NET 5
TLS 1.1 Handshake Duration (<code>tls11-handshake-duration</code>)	The average duration of TLS 1.1 handshakes	.NET 5
TLS 1.2 Handshake Duration (<code>tls12-handshake-duration</code>)	The average duration of TLS 1.2 handshakes	.NET 5
TLS 1.3 Handshake Duration (<code>tls13-handshake-duration</code>)	The average duration of TLS 1.3 handshakes	.NET 5

System.Net.Sockets counters

The following counters track metrics related to [Socket](#).

COUNTER	DESCRIPTION	FIRST AVAILABLE IN
Outgoing Connections Established (<code>outgoing-connections-established</code>)	The total number of outgoing connections established since the process started	.NET 5
Incoming Connections Established (<code>incoming-connections-established</code>)	The total number of incoming connections established since the process started	.NET 5
Bytes Received (<code>bytes-received</code>)	The total number of bytes received since the process started	.NET 5
Bytes Sent (<code>bytes-sent</code>)	The total number of bytes sent since the process started	.NET 5
Datagrams Received (<code>datagrams-received</code>)	The total number of datagrams received since the process started	.NET 5
Datagrams Sent (<code>datagrams-sent</code>)	The total number of datagrams sent since the process started	.NET 5

Tutorial: Measure performance using EventCounters in .NET Core

9/20/2022 • 5 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 3.0 SDK and later versions

In this tutorial, you'll learn how an [EventCounter](#) can be used to measure performance with a high frequency of events. You can use the [available counters](#) published by various official .NET Core packages, third-party providers, or create your own metrics for monitoring.

In this tutorial, you will:

- Implement an [EventSource](#).
- Monitor counters with [dotnet-counters](#).

Prerequisites

The tutorial uses:

- [.NET Core 3.1 SDK](#) or a later version.
- [dotnet-counters](#) to monitor event counters.
- A [sample debug target app](#) to diagnose.

Get the source

The sample application will be used as a basis for monitoring. The [sample ASP.NET Core repository](#) is available from the samples browser. You download the zip file, extract it once downloaded, and open it in your favorite IDE. Build and run the application to ensure that it works properly, then stop the application.

Implement an EventSource

For events that happen every few milliseconds, you'll want the overhead per event to be low (less than a millisecond). Otherwise, the impact on performance will be significant. Logging an event means you're going to write something to disk. If the disk is not fast enough, you will lose events. You need a solution other than logging the event itself.

When dealing with a large number of events, knowing the measure per event is not useful either. Most of the time all you need is just some statistics out of it. So you could get the statistics within the process itself and then write an event once in a while to report the statistics, that's what [EventCounter](#) will do.

Below is an example of how to implement an [System.Diagnostics.Tracing.EventSource](#). Create a new file named *MinimalEventCounterSource.cs* and use the code snippet as its source:

```

using System.Diagnostics.Tracing;

[EventSource(Name = "Sample.EventCounter.Minimal")]
public sealed class MinimalEventCounterSource : EventSource
{
    public static readonly MinimalEventCounterSource Log = new MinimalEventCounterSource();

    private EventCounter _requestCounter;

    private MinimalEventCounterSource() =>
        _requestCounter = new EventCounter("request-time", this)
    {
        DisplayName = "Request Processing Time",
        DisplayUnits = "ms"
    };

    public void Request(string url, long elapsedMilliseconds)
    {
        WriteEvent(1, url, elapsedMilliseconds);
        _requestCounter?.WriteMetric(elapsedMilliseconds);
    }

    protected override void Dispose(bool disposing)
    {
        _requestCounter?.Dispose();
        _requestCounter = null;

        base.Dispose(disposing);
    }
}

```

The `EventSource.WriteEvent` line is the `EventSource` part and is not part of `EventCounter`, it was written to show that you can log a message together with the event counter.

Add an action filter

The sample source code is an ASP.NET Core project. You can add an [action filter](#) globally that will log the total request time. Create a new file named `LogRequestTimeFilterAttribute.cs`, and use the following code:

```

using System.Diagnostics;
using Microsoft.AspNetCore.Http.Extensions;
using Microsoft.AspNetCore.Mvc.Filters;

namespace DiagnosticScenarios
{
    public class LogRequestTimeFilterAttribute : ActionFilterAttribute
    {
        readonly Stopwatch _stopwatch = new Stopwatch();

        public override void OnActionExecuting(ActionExecutingContext context) => _stopwatch.Start();

        public override void OnActionExecuted(ActionExecutedContext context)
        {
            _stopwatch.Stop();

            MinimalEventCounterSource.Log.Request(
                context.HttpContext.Request.GetDisplayUrl(), _stopwatch.ElapsedMilliseconds);
        }
    }
}

```

The action filter starts a `Stopwatch` as the request begins, and stops after it has completed, capturing the elapsed time. The total milliseconds is logged to the `MinimalEventCounterSource` singleton instance. In order for this filter

to be applied, you need to add it to the filters collection. In the `Startup.cs` file, update the `ConfigureServices` method to include this filter.

```
public void ConfigureServices(IServiceCollection services) =>
    services.AddControllers(options => options.Filters.Add<LogRequestTimeFilterAttribute>())
        .AddNewtonsoftJson();
```

Monitor event counter

With the implementation on an [EventSource](#) and the custom action filter, build and launch the application. You logged the metric to the [EventCounter](#), but unless you access the statistics from it, it is not useful. To get the statistics, you need to enable the [EventCounter](#) by creating a timer that fires as frequently as you want the events, as well as a listener to capture the events. To do that, you can use [dotnet-counters](#).

Use the `dotnet-counters ps` command to display a list of .NET processes that can be monitored.

```
dotnet-counters ps
```

Using the process identifier from the output of the `dotnet-counters ps` command, you can start monitoring the event counter with the following `dotnet-counters monitor` command:

```
dotnet-counters monitor --process-id 2196 --counters
Sample.EventCounter.Minimal,Microsoft.AspNetCore.Hosting[total-requests,requests-per-second],System.Runtime[cpu-usage]
```

While the `dotnet-counters monitor` command is running, hold F5 on the browser to start issuing continuous requests to the `https://localhost:5001/api/values` endpoint. After a few seconds stop by pressing q

```
Press p to pause, r to resume, q to quit.
Status: Running

[Microsoft.AspNetCore.Hosting]
  Request Rate / 1 sec          9
  Total Requests                134
[System.Runtime]
  CPU Usage (%)                 13
[Sample.EventCounter.Minimal]
  Request Processing Time (ms)  34.5
```

The `dotnet-counters monitor` command is great for active monitoring. However, you may want to collect these diagnostic metrics for post processing and analysis. For that, use the `dotnet-counters collect` command. The `collect` switch command is similar to the `monitor` command, but accepts a few additional parameters. You can specify the desired output file name and format. For a JSON file named `diagnostics.json` use the following command:

```
dotnet-counters collect --process-id 2196 --format json -o diagnostics.json --counters
Sample.EventCounter.Minimal,Microsoft.AspNetCore.Hosting[total-requests,requests-per-second],System.Runtime[cpu-usage]
```

Again, while the command is running, hold F5 on the browser to start issuing continuous requests to the `https://localhost:5001/api/values` endpoint. After a few seconds stop by pressing q. The `diagnostics.json` file is written. The JSON file that is written is not indented, however; for readability it is indented here.

```
{
  "TargetProcess": "DiagnosticScenarios",
  "StartTime": "8/5/2020 3:02:45 PM",
  "Events": [
    {
      "timestamp": "2020-08-05 15:02:47Z",
      "provider": "System.Runtime",
      "name": "CPU Usage (%)",
      "counterType": "Metric",
      "value": 0
    },
    {
      "timestamp": "2020-08-05 15:02:47Z",
      "provider": "Microsoft.AspNetCore.Hosting",
      "name": "Request Rate / 1 sec",
      "counterType": "Rate",
      "value": 0
    },
    {
      "timestamp": "2020-08-05 15:02:47Z",
      "provider": "Microsoft.AspNetCore.Hosting",
      "name": "Total Requests",
      "counterType": "Metric",
      "value": 134
    },
    {
      "timestamp": "2020-08-05 15:02:47Z",
      "provider": "Sample.EventCounter.Minimal",
      "name": "Request Processing Time (ms)",
      "counterType": "Metric",
      "value": 0
    },
    {
      "timestamp": "2020-08-05 15:02:47Z",
      "provider": "System.Runtime",
      "name": "CPU Usage (%)",
      "counterType": "Metric",
      "value": 0
    },
    {
      "timestamp": "2020-08-05 15:02:48Z",
      "provider": "Microsoft.AspNetCore.Hosting",
      "name": "Request Rate / 1 sec",
      "counterType": "Rate",
      "value": 0
    },
    {
      "timestamp": "2020-08-05 15:02:48Z",
      "provider": "Microsoft.AspNetCore.Hosting",
      "name": "Total Requests",
      "counterType": "Metric",
      "value": 134
    },
    {
      "timestamp": "2020-08-05 15:02:48Z",
      "provider": "Sample.EventCounter.Minimal",
      "name": "Request Processing Time (ms)",
      "counterType": "Metric",
      "value": 0
    },
    {
      "timestamp": "2020-08-05 15:02:48Z",
      "provider": "System.Runtime",
      "name": "CPU Usage (%)",
      "counterType": "Metric",
      "value": 0
    },
    {
      "timestamp": "2020-08-05 15:02:50Z",
      "provider": "Microsoft.AspNetCore.Hosting",
      "name": "Request Rate / 1 sec",
      "counterType": "Rate",
      "value": 0
    },
    {
      "timestamp": "2020-08-05 15:02:50Z",
      "provider": "Microsoft.AspNetCore.Hosting",
      "name": "Total Requests",
      "counterType": "Metric",
      "value": 134
    },
    {
      "timestamp": "2020-08-05 15:02:50Z",
      "provider": "Sample.EventCounter.Minimal",
      "name": "Request Processing Time (ms)",
      "counterType": "Metric",
      "value": 0
    },
    {
      "timestamp": "2020-08-05 15:02:50Z",
      "provider": "System.Runtime",
      "name": "CPU Usage (%)",
      "counterType": "Metric",
      "value": 0
    }
  ]
}
```

```
        "provider": "Microsoft.AspNetCore.Hosting",
        "name": "Request Rate / 1 sec",
        "counterType": "Rate",
        "value": 0
    },
    {
        "timestamp": "2020-08-05 15:02:50Z",
        "provider": "Microsoft.AspNetCore.Hosting",
        "name": "Total Requests",
        "counterType": "Metric",
        "value": 134
    },
    {
        "timestamp": "2020-08-05 15:02:50Z",
        "provider": "Sample.EventCounter.Minimal",
        "name": "Request Processing Time (ms)",
        "counterType": "Metric",
        "value": 0
    }
]
}
```

Next steps

[EventCounters](#)

Metric APIs comparison

9/20/2022 • 3 minutes to read • [Edit Online](#)

When adding new metric instrumentation to a .NET app or library, there are various different APIs to choose from. This article will help you understand what is available and some of the tradeoffs involved.

There are two major categories of APIs, vendor-neutral and vendor-specific. Vendor-specific APIs have the advantage that the vendor can iterate their designs quickly, add specialized features, and achieve tight integration between their instrumentation APIs and their backend systems. As an example, if you instrumented your app with metric APIs provided by [Application Insights](#), then you would expect to find well-integrated functionality and all of Application Insights' latest features when working with their analysis tools. However the library or app would also now be coupled to this vendor and changing to a different one in the future would require rewriting the instrumentation. For libraries, this coupling can be particularly problematic because the library developer might use one vendor's API and the app developer that references the library wants to work with a different vendor. To resolve this coupling issue, vendor-neutral options provide a standardized API façade and extensibility points to route data to various vendor backend systems depending on configuration. However, vendor-neutral APIs may provide fewer capabilities, and you're still constrained to pick a vendor that has integrated with the façade's extensibility mechanism.

.NET APIs

Over .NET's 20+ year history, we've iterated a few times on the design for metric APIs, all of which are supported and vendor-neutral:

PerformanceCounter

[System.Diagnostics.PerformanceCounter](#) APIs are the oldest metric APIs. They're only supported on Windows and provide a managed wrapper for Windows OS [Performance Counter](#) technology. They are available in all supported versions of .NET.

These APIs are provided primarily for compatibility; the .NET team considers this a stable area that's unlikely to receive further improvement aside from bug fixes. These APIs are not suggested for new development projects unless the project is Windows-only and you have a desire to use Windows Performance Counter tools.

For more information, see [Performance counters in .NET Framework](#).

EventCounters

The [EventCounters](#) API came next after [PerformanceCounters](#). This API aimed to provide a uniform cross-platform experience. The APIs are available by targeting .NET Core 3.1+, and a small subset is available on .NET Framework 4.7.1 and above. These APIs are fully supported and are actively used by key .NET libraries, but they have less functionality than the newer [System.Diagnostics.Metrics](#) APIs. EventCounters are able to report rates of change and averages, but do not support histograms and percentiles. There is also no support for multi-dimensional metrics. Custom tooling is possible via the [EventListener](#) API, though it is not strongly typed, only gives access to the aggregated values, and has limitations when using more than one listener simultaneously. EventCounters are supported directly by [Visual Studio](#), [Application Insights](#), [dotnet-counters](#), and [dotnet-monitor](#). For third-party tool support, check the vendor or project documentation to see if it's available.

At the time of writing, this is the cross-platform .NET runtime API that has the broadest and most stable ecosystem support. However, it will likely be overtaken soon by growing support for [System.Diagnostics.Metrics](#). The .NET team doesn't expect to make substantial new investments on this API going forward, but as with [PerformanceCounters](#), the API remains actively supported for all current and future users.

System.Diagnostics.Metrics

[System.Diagnostics.Metrics](#) APIs are the newest cross-platform APIs, and were designed in close collaboration with the [OpenTelemetry](#) project. The OpenTelemetry effort is an industry-wide collaboration across telemetry tooling vendors, programming languages, and application developers to create a broadly compatible standard for telemetry APIs. To eliminate any friction associated with adding third-party dependencies, .NET embeds the metrics API directly into the base class libraries. It's available by targeting .NET 6, or in older .NET Core and .NET Framework apps by adding a reference to the .NET [System.Diagnostics.DiagnosticsSource](#) 6.0 NuGet package. In addition to aiming at broad compatibility, this API adds support for many things that were lacking from EventCounters, such as:

- Histograms and percentiles
- Multi-dimensional metrics
- Strongly typed high-performance listener API
- Multiple simultaneous listeners
- Listener access to unaggregated measurements

Although this API was designed to work well with OpenTelemetry and its growing ecosystem of pluggable vendor integration libraries, applications also have the option to use the .NET built-in listener APIs directly. With this option, you can create custom metric tooling without taking any external library dependencies. At the time of writing, the System.Diagnostics.Metrics APIs are brand new and support is limited to [dotnet-counters](#) and [OpenTelemetry.NET](#). However, we expect support for these APIs will grow quickly given the active nature of the OpenTelemetry project.

Third-party APIs

Most application performance monitoring (APM) vendors such as [AppDynamics](#), [Application Insights](#), [DataDog](#), [DynaTrace](#), and [NewRelic](#) include metrics APIs as part of their instrumentation libraries. [Prometheus](#) and [AppMetrics](#) are also popular .NET OSS projects. To learn more about these projects, check the various project websites.

.NET distributed tracing

9/20/2022 • 2 minutes to read • [Edit Online](#)

Distributed tracing is a diagnostic technique that helps engineers localize failures and performance issues within applications, especially those that may be distributed across multiple machines or processes. This technique tracks requests through an application correlating together work done by different application components and separating it from other work the application may be doing for concurrent requests. For example, a request to a typical web service might be first received by a load balancer, then forwarded to a web server process, which then makes several queries to a database. Using distributed tracing allows engineers to distinguish if any of those steps failed, how long each step took, and potentially logging messages produced by each step as it ran.

Getting started for .NET app developers

Key .NET libraries are instrumented to produce distributed tracing information automatically. However, this information needs to be collected and stored so that it will be available for review later. Typically, app developers select a telemetry service that stores this trace information for them and then use a corresponding library to transmit the distributed tracing telemetry to their chosen service:

- [OpenTelemetry](#) is a vendor-neutral library that supports several services. For more information, see [Collect distributed traces with OpenTelemetry](#).
- [Application Insights](#) is a full-featured service provided by Microsoft. For more information, see [Collect distributed traces with Application Insights](#).
- There are many high-quality third-party application performance monitoring (APM) vendors that offer integrated .NET solutions.

For more information, see [Understand distributed tracing concepts](#) and the following guides:

- [Collect distributed traces with custom logic](#)
- [Adding custom distributed trace instrumentation](#)

For third-party telemetry collection services, follow the setup instructions provided by the vendor.

Getting started for .NET library developers

.NET libraries don't need to be concerned with how telemetry is ultimately collected, only with how it is produced. If you want consumers of your library to be able to see the work that it does detailed in a distributed trace, add distributed tracing instrumentation to support it.

For more information, see [Understand distributed tracing concepts](#) and the [Adding custom distributed trace instrumentation](#) guide.

.NET distributed tracing concepts

9/20/2022 • 5 minutes to read • [Edit Online](#)

Distributed tracing is a diagnostic technique that helps engineers localize failures and performance issues within applications, especially those that may be distributed across multiple machines or processes. See the [Distributed Tracing Overview](#) for general information about where distributed tracing is useful and example code to get started.

Traces and Activities

Each time a new request is received by an application, it can be associated with a trace. In application components written in .NET, units of work in a trace are represented by instances of [System.Diagnostics.Activity](#) and the trace as a whole forms a tree of these Activities, potentially spanning across many distinct processes. The first Activity created for a new request forms the root of the trace tree and it tracks the overall duration and success/failure handling the request. Child activities can be optionally created to subdivide the work into different steps that can be tracked individually. For example given an Activity that tracked a specific inbound HTTP request in a web server, child activities could be created to track each of the database queries that were necessary to complete the request. This allows the duration and success for each query to be recorded independently. Activities can record other information for each unit of work such as [OperationName](#), name-value pairs called [Tags](#), and [Events](#). The name identifies the type of work being performed, tags can record descriptive parameters of the work, and events are a simple logging mechanism to record timestamped diagnostic messages.

NOTE

Another common industry name for units of work in a distributed trace are 'Spans'. .NET adopted the term 'Activity' many years ago, before the name 'Span' was well established for this concept.

Activity IDs

Parent-Child relationships between Activities in the distributed trace tree are established using unique IDs. .NET's implementation of distributed tracing supports two ID schemes: the W3C standard [TraceContext](#), which is the default in .NET 5+, and an older .NET convention called 'Hierarchical' that's available for backwards compatibility. [Activity.DefaultIdFormat](#) controls which ID scheme is used. In the W3C TraceContext standard, every trace is assigned a globally unique 16-byte trace-id ([Activity.TraceId](#)), and every Activity within the trace is assigned a unique 8-byte span-id ([Activity.SpanId](#)). Each Activity records the trace-id, its own span-id, and the span-id of its parent ([Activity.ParentSpanId](#)). Because distributed traces can track work across process boundaries, parent and child Activities may not be in the same process. The combination of a trace-id and parent span-id can uniquely identify the parent Activity globally, regardless of what process it resides in.

[Activity.DefaultIdFormat](#) controls which ID format is used for starting new traces, but by default adding a new Activity to an existing trace uses whatever format the parent Activity is using. Setting [Activity.ForceDefaultIdFormat](#) to true overrides this behavior and creates all new Activities with the DefaultIdFormat, even when the parent uses a different ID format.

Start and stop Activities

Each thread in a process may have a corresponding Activity object that is tracking the work occurring on that thread, accessible via [Activity.Current](#). The current activity automatically flows along all synchronous calls on a

thread as well as following async calls that are processed on different threads. If Activity A is the current activity on a thread and code starts a new Activity B, then B becomes the new current activity on that thread. By default, activity B will also treat Activity A as its parent. When Activity B is later stopped, activity A will be restored as the current Activity on the thread. When an Activity is started, it captures the current time as the [Activity.StartTimeUtc](#). When it stops, [Activity.Duration](#) is calculated as the difference between the current time and the start time.

Coordinate across process boundaries

To track work across process boundaries, Activity parent IDs need to be transmitted across the network so that the receiving process can create Activities that refer to them. When using W3C TraceContext ID format, .NET also uses the HTTP headers recommended by [the standard](#) to transmit this information. When using the [Hierarchical](#) ID format, .NET uses a custom request-id HTTP header to transmit the ID. Unlike many other language runtimes, .NET in-box libraries such as the ASP.NET web server and System.Net.Http natively understand how to decode and encode Activity IDs on HTTP messages. The runtime also understands how to flow the ID through synchronous and asynchronous calls. This means that .NET applications that receive and emit HTTP messages participate in flowing distributed trace IDs automatically, with no special coding by the app developer or third-party library dependencies. Third-party libraries may add support for transmitting IDs over non-HTTP message protocols or supporting custom encoding conventions for HTTP.

Collect traces

Instrumented code can create [Activity](#) objects as part of a distributed trace, but the information in these objects needs to be transmitted and serialized in a centralized persistent store so that the entire trace can be usefully reviewed later. There are several telemetry collection libraries that can do this task such as [Application Insights](#), [OpenTelemetry](#), or a library provided by a third-party telemetry or APM vendor. Alternately developers can author their own custom Activity telemetry collection by using [System.Diagnostics.ActivityListener](#) or [System.Diagnostics.DiagnosticListener](#). ActivityListener supports observing any Activity regardless of whether the developer has any prior knowledge about it. This makes ActivityListener a simple and flexible general purpose solution. By contrast using DiagnosticListener is a more complex scenario that requires the instrumented code to opt in by invoking [DiagnosticSource.StartActivity](#) and the collection library needs to know the exact naming information that the instrumented code used when starting it. Using DiagnosticSource and DiagnosticListener allows the creator and listener to exchange arbitrary .NET objects and establish customized information passing conventions.

Sampling

For improved performance in high throughput applications, distributed tracing on .NET supports sampling only a subset of traces rather than recording all of them. For activities created with the recommended [ActivitySource.StartActivity](#) API, telemetry collection libraries can control sampling with the [ActivityListener.Sample](#) callback. The logging library can elect not to create the Activity at all, to create it with minimal information necessary to propagate distributing tracing IDs, or to populate it with complete diagnostic information. These choices trade-off increasing performance overhead for increasing diagnostic utility. Activities that are started using the older pattern of invoking [Activity.Activity](#) and [DiagnosticSource.StartActivity](#) may also support DiagnosticListener sampling by first calling [DiagnosticSource.IsEnabled](#). Even when capturing full diagnostic information the .NET implementation is designed to be fast - coupled with an efficient collector an Activity can be created, populated, and transmitted in about a microsecond on modern hardware. Sampling can reduce the instrumentation cost to less than 100 nanoseconds for each Activity that isn't recorded.

Next steps

For example code to get started using distributed tracing in .NET applications, see the [Distributed Tracing Instrumentation](#).

Adding distributed tracing instrumentation

9/20/2022 • 9 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 2.1 and later versions ✓ .NET Framework 4.5 and later versions

.NET applications can be instrumented using the [System.Diagnostics.Activity](#) API to produce distributed tracing telemetry. Some instrumentation is built into standard .NET libraries, but you may want to add more to make your code more easily diagnosable. In this tutorial, you will add new custom distributed tracing instrumentation. See [the collection tutorial](#) to learn more about recording the telemetry produced by this instrumentation.

Prerequisites

- [.NET Core 2.1 SDK](#) or a later version

Create initial app

First you will create a sample app that collects telemetry using OpenTelemetry, but doesn't yet have any instrumentation.

```
dotnet new console
```

Applications that target .NET 5 and later already have the necessary distributed tracing APIs included. For apps targeting older .NET versions, add the [System.Diagnostics.DiagnosticSource NuGet package](#) version 5 or greater.

```
dotnet add package System.Diagnostics.DiagnosticSource
```

Add the [OpenTelemetry](#) and [OpenTelemetry.Exporter.Console](#) NuGet packages, which will be used to collect the telemetry.

```
dotnet add package OpenTelemetry
dotnet add package OpenTelemetry.Exporter.Console
```

Replace the contents of the generated Program.cs with this example source:

```

using OpenTelemetry;
using OpenTelemetry.Resources;
using OpenTelemetry.Trace;
using System;
using System.Threading.Tasks;

namespace Sample.DistributedTracing
{
    class Program
    {
        static async Task Main(string[] args)
        {
            using var tracerProvider = Sdk.CreateTracerProviderBuilder()
                .SetResourceBuilder(ResourceBuilder.CreateDefault().AddService("MySample"))
                .AddSource("Sample.DistributedTracing")
                .AddConsoleExporter()
                .Build();

            await DoSomeWork("banana", 8);
            Console.WriteLine("Example work done");
        }

        // All the functions below simulate doing some arbitrary work
        static async Task DoSomeWork(string foo, int bar)
        {
            await StepOne();
            await StepTwo();
        }

        static async Task StepOne()
        {
            await Task.Delay(500);
        }

        static async Task StepTwo()
        {
            await Task.Delay(1000);
        }
    }
}

```

The app has no instrumentation yet so there is no trace information to display:

```

> dotnet run
Example work done

```

Best practices

Only app developers need to reference an optional third-party library for collecting the distributed trace telemetry, such as OpenTelemetry in this example. .NET library authors can exclusively rely on APIs in `System.Diagnostics.DiagnosticSource`, which is part of .NET runtime. This ensures that libraries will run in a wide range of .NET apps, regardless of the app developer's preferences about which library or vendor to use for collecting telemetry.

Add basic instrumentation

Applications and libraries add distributed tracing instrumentation using the `System.Diagnostics.ActivitySource` and `System.Diagnostics.Activity` classes.

ActivitySource

First create an instance of `ActivitySource`. `ActivitySource` provides APIs to create and start `Activity` objects. Add

the static ActivitySource variable above Main() and `using System.Diagnostics;` to the using statements.

```
using OpenTelemetry;
using OpenTelemetry.Resources;
using OpenTelemetry.Trace;
using System;
using System.Diagnostics;
using System.Threading.Tasks;

namespace Sample.DistributedTracing
{
    class Program
    {
        private static ActivitySource source = new ActivitySource("Sample.DistributedTracing", "1.0.0");

        static async Task Main(string[] args)
        {
            ...
        }
    }
}
```

Best practices

- Create the ActivitySource once, store it in a static variable and use that instance as long as needed. Each library or library subcomponent can (and often should) create its own source. Consider creating a new source rather than reusing an existing one if you anticipate app developers would appreciate being able to enable and disable the Activity telemetry in the sources independently.
- The source name passed to the constructor has to be unique to avoid the conflicts with any other sources. If there are multiple sources within the same assembly, use a hierarchical name that contains the assembly name and optionally a component name, for example, `Microsoft.AspNetCore.Hosting`. If an assembly is adding instrumentation for code in a second, independent assembly, the name should be based on the assembly that defines the ActivitySource, not the assembly whose code is being instrumented.
- The version parameter is optional. We recommend that you provide the version in case you release multiple versions of the library and make changes to the instrumented telemetry.

NOTE

OpenTelemetry uses alternate terms 'Tracer' and 'Span'. In .NET 'ActivitySource' is the implementation of Tracer and Activity is the implementation of 'Span'. .NET's Activity type long pre-dates the OpenTelemetry specification and the original .NET naming has been preserved for consistency within the .NET ecosystem and .NET application compatibility.

Activity

Use the ActivitySource object to Start and Stop Activity objects around meaningful units of work. Update DoSomeWork() with the code shown here:

```
static async Task DoSomeWork(string foo, int bar)
{
    using (Activity activity = source.StartActivity("SomeWork"))
    {
        await StepOne();
        await StepTwo();
    }
}
```

Running the app now shows the new Activity being logged:

```
> dotnet run
Activity.Id:          00-f443e487a4998c41a6fd6fe88bae644e-5b7253de08ed474f-01
Activity.DisplayName: SomeWork
Activity.Kind:        Internal
Activity.StartTime:   2021-03-18T10:36:51.4720202Z
Activity.Duration:    00:00:01.5025842
Resource associated with Activity:
  service.name: MySample
  service.instance.id: 067f4bb5-a5a8-4898-a288-dec569d6dbef
```

Notes

- [ActivitySource.StartActivity](#) creates and starts the activity at the same time. The listed code pattern is using the `using` block, which automatically disposes the created `Activity` object after executing the block. Disposing the `Activity` object will stop it so the code doesn't need to explicitly call [Activity.Stop\(\)](#). That simplifies the coding pattern.
- [ActivitySource.StartActivity](#) internally determines if there are any listeners recording the `Activity`. If there are no registered listeners or there are listeners that are not interested, `StartActivity()` will return `null` and avoid creating the `Activity` object. This is a performance optimization so that the code pattern can still be used in functions that are called frequently.

Optional: Populate tags

Activities support key-value data called Tags, commonly used to store any parameters of the work that may be useful for diagnostics. Update `DoSomeWork()` to include them:

```
static async Task DoSomeWork(string foo, int bar)
{
    using (Activity activity = source.StartActivity("SomeWork"))
    {
        activity?.SetTag("foo", foo);
        activity?.SetTag("bar", bar);
        await StepOne();
        await StepTwo();
    }
}
```

```
> dotnet run
Activity.Id:          00-2b56072db8cb5a4496a4fb69f46aa06-7bc4acda3b9cce4d-01
Activity.DisplayName: SomeWork
Activity.Kind:        Internal
Activity.StartTime:   2021-03-18T10:37:31.4949570Z
Activity.Duration:    00:00:01.5417719
Activity.TagObjects:
  foo: banana
  bar: 8
Resource associated with Activity:
  service.name: MySample
  service.instance.id: 25bbc1c3-2de5-48d9-9333-062377fea49c

Example work done
```

Best practices

- As mentioned above, `activity` returned by [ActivitySource.StartActivity](#) may be null. The null-coalescing operator `?.` in C# is a convenient short-hand to only invoke [Activity.SetTag](#) if `activity` is not null. The behavior is identical to writing:

```

if(activity != null)
{
    activity.SetTag("foo", foo);
}

```

- OpenTelemetry provides a set of recommended [conventions](#) for setting Tags on Activities that represent common types of application work.
- If you are instrumenting functions with high-performance requirements, [Activity.IsAllDataRequested](#) is a hint that indicates whether any of the code listening to Activities intends to read auxiliary information such as Tags. If no listener will read it, then there is no need for the instrumented code to spend CPU cycles populating it. For simplicity, this sample doesn't apply that optimization.

Optional: Add events

Events are timestamped messages that can attach an arbitrary stream of additional diagnostic data to Activities. Add some events to the Activity:

```

static async Task DoSomeWork(string foo, int bar)
{
    using (Activity activity = source.StartActivity("SomeWork"))
    {
        activity?.SetTag("foo", foo);
        activity?.SetTag("bar", bar);
        await StepOne();
        activity?.AddEvent(new ActivityEvent("Part way there"));
        await StepTwo();
        activity?.AddEvent(new ActivityEvent("Done now"));
    }
}

```

```

> dotnet run
Activity.Id:          00-82cf6ea92661b84d9fd881731741d04e-33fff2835a03c041-01
Activity.DisplayName: SomeWork
Activity.Kind:        Internal
Activity.StartTime:   2021-03-18T10:39:10.6902609Z
Activity.Duration:    00:00:01.5147582
Activity.TagObjects:
    foo: banana
    bar: 8
Activity.Events:
    Part way there [3/18/2021 10:39:11 AM +00:00]
    Done now [3/18/2021 10:39:12 AM +00:00]
Resource associated with Activity:
    service.name: MySample
    service.instance.id: ea7f0fc...-e4af5a86ce4f

Example work done

```

Best practices

- Events are stored in an in-memory list until they can be transmitted which makes this mechanism only suitable for recording a modest number of events. For a large or unbounded volume of events, it's better to use a logging API focused on this task, such as [ILogger](#). ILogger also ensures that the logging information will be available regardless whether the app developer opts to use distributed tracing. ILogger supports automatically capturing the active Activity IDs so messages logged via that API can still be correlated with the distributed trace.

Optional: Add status

OpenTelemetry allows each Activity to report a [Status](#) that represents the pass/fail result of the work. .NET does not currently have a strongly typed API for this purpose but there is an established convention using Tags:

- `otel.status_code` is the Tag name used to store `StatusCode`. Values for the StatusCode tag must be one of the strings "UNSET", "OK", or "ERROR", which correspond respectively to the enums `Unset`, `Ok`, and `Error` from `StatusCode`.
- `otel.status_description` is the Tag name used to store the optional `Description`

Update `DoSomeWork()` to set status:

```
static async Task DoSomeWork(string foo, int bar)
{
    using (Activity activity = source.StartActivity("SomeWork"))
    {
        activity?.SetTag("foo", foo);
        activity?.SetTag("bar", bar);
        await StepOne();
        activity?.AddEvent(new ActivityEvent("Part way there"));
        await StepTwo();
        activity?.AddEvent(new ActivityEvent("Done now"));

        // Pretend something went wrong
        activity?.SetTag("otel.status_code", "ERROR");
        activity?.SetTag("otel.status_description", "Use this text give more information about the
error");
    }
}
```

Optional: Add additional Activities

Activities can be nested to describe portions of a larger unit of work. This can be valuable around portions of code that might not execute quickly or to better localize failures that come from specific external dependencies. Although this sample uses an Activity in every method, that is solely because extra code has been minimized. In a larger and more realistic project, using an Activity in every method would produce extremely verbose traces, so it's not recommended.

Update `StepOne` and `StepTwo` to add more tracing around these separate steps:

```
static async Task StepOne()
{
    using (Activity activity = source.StartActivity("StepOne"))
    {
        await Task.Delay(500);
    }
}

static async Task StepTwo()
{
    using (Activity activity = source.StartActivity("StepTwo"))
    {
        await Task.Delay(1000);
    }
}
```

```

> dotnet run
Activity.Id:          00-9d5aa439e0df7e49b4abff8d2d5329a9-39cac574e8fda44b-01
Activity.ParentId:    00-9d5aa439e0df7e49b4abff8d2d5329a9-f16529d0b7c49e44-01
Activity.DisplayName: StepOne
Activity.Kind:        Internal
Activity.StartTime:   2021-03-18T10:40:51.4278822Z
Activity.Duration:    00:00:00.5051364
Resource associated with Activity:
  service.name: MySample
  service.instance.id: e0a8c12c-249d-4bdd-8180-8931b9b6e8d0

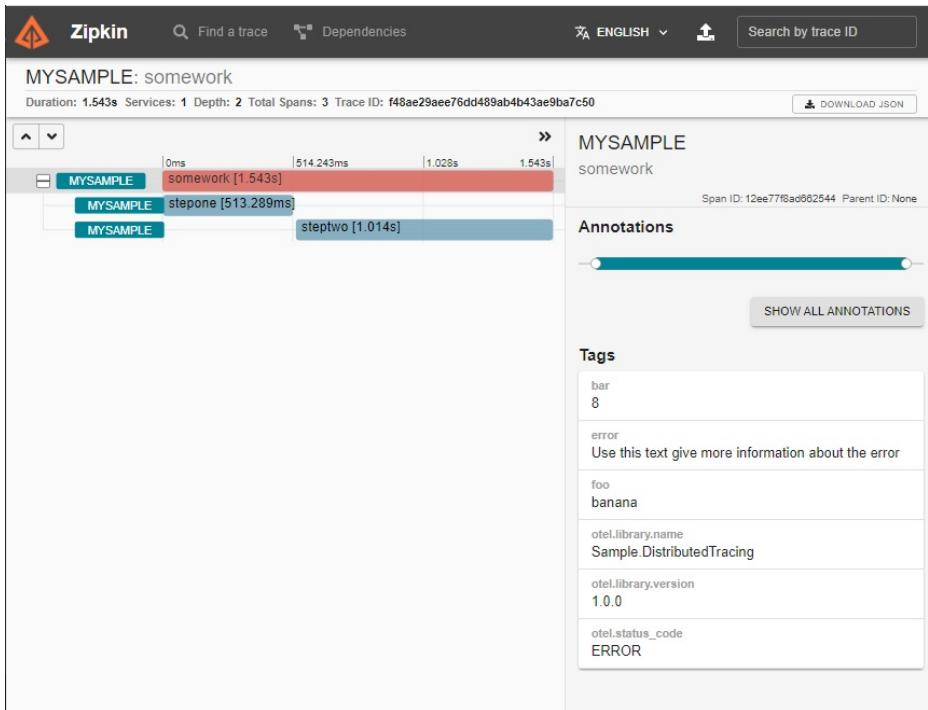
Activity.Id:          00-9d5aa439e0df7e49b4abff8d2d5329a9-4cccb6efdc59546-01
Activity.ParentId:    00-9d5aa439e0df7e49b4abff8d2d5329a9-f16529d0b7c49e44-01
Activity.DisplayName: StepTwo
Activity.Kind:        Internal
Activity.StartTime:   2021-03-18T10:40:51.9441095Z
Activity.Duration:    00:00:01.0052729
Resource associated with Activity:
  service.name: MySample
  service.instance.id: e0a8c12c-249d-4bdd-8180-8931b9b6e8d0

Activity.Id:          00-9d5aa439e0df7e49b4abff8d2d5329a9-f16529d0b7c49e44-01
Activity.DisplayName: SomeWork
Activity.Kind:        Internal
Activity.StartTime:   2021-03-18T10:40:51.4256627Z
Activity.Duration:    00:00:01.5286408
Activity.TagObjects:
  foo: banana
  bar: 8
  otel.status_code: ERROR
  otel.status_description: Use this text give more information about the error
Activity.Events:
  Part way there [3/18/2021 10:40:51 AM +00:00]
  Done now [3/18/2021 10:40:52 AM +00:00]
Resource associated with Activity:
  service.name: MySample
  service.instance.id: e0a8c12c-249d-4bdd-8180-8931b9b6e8d0

Example work done

```

Notice that both StepOne and StepTwo include a ParentId that refers to SomeWork. The console is not a great visualization of nested trees of work, but many GUI viewers such as [Zipkin](#) can show this as a Gantt chart:



Optional: ActivityKind

Activities have an [Activity.Kind](#) property, which describes the relationship between the Activity, its parent, and its children. By default, all new Activities are set to [Internal](#), which is appropriate for Activities that are an internal operation within an application with no remote parent or children. Other kinds can be set using the kind parameter on [ActivitySource.StartActivity](#). For other options, see [System.Diagnostics.ActivityKind](#).

Optional: Links

When work occurs in batch processing systems, a single Activity might represent work on behalf of many different requests simultaneously, each of which has its own trace-id. Although Activity is restricted to have a single parent, it can link to additional trace-ids using [System.Diagnostics.ActivityLink](#). Each ActivityLink is populated with an [ActivityContext](#) that stores ID information about the Activity being linked to. ActivityContext can be retrieved from in-process Activity objects using [Activity.Context](#) or it can be parsed from serialized ID information using [ActivityContext.Parse\(String, String\)](#).

```
void DoBatchWork(ActivityContext[] requestContexts)
{
    // Assume each context in requestContexts encodes the trace-id that was sent with a request
    using(Activity activity = s_source.StartActivity(name: "BigBatchOfWork",
        kind: ActivityKind.Internal,
        parentContext: null,
        links: requestContexts.Select(ctx => new
            ActivityLink(ctx)))
    {
        // do the batch of work here
    }
}
```

Unlike events and Tags that can be added on-demand, links must be added during `StartActivity()` and are immutable afterwards.

Collect a distributed trace

9/20/2022 • 6 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 2.1 and later versions ✓ .NET Framework 4.5 and later versions

Instrumented code can create [Activity](#) objects as part of a distributed trace, but the information in these objects needs to be collected into centralized storage so that the entire trace can be reviewed later. In this tutorial, you will collect the distributed trace telemetry in different ways so that it's available to diagnose application issues when needed. See [the instrumentation tutorial](#) if you need to add new instrumentation.

Collect traces using OpenTelemetry

Prerequisites

- [.NET Core 2.1 SDK](#) or a later version

Create an example application

Before any distributed trace telemetry can be collected, we need to produce it. Often this instrumentation might be in libraries, but for simplicity, you'll create a small app that has some example instrumentation using [StartActivity](#). At this point, no collection has happened, and StartActivity() has no side-effect and returns null. See [the instrumentation tutorial](#) for more details.

```
dotnet new console
```

Applications that target .NET 5 and later already have the necessary distributed tracing APIs included. For apps targeting older .NET versions, add the [System.Diagnostics.DiagnosticSource NuGet package](#) version 5 or greater.

```
dotnet add package System.Diagnostics.DiagnosticSource
```

Replace the contents of the generated Program.cs with this example source:

```

using System;
using System.Diagnostics;
using System.Threading.Tasks;

namespace Sample.DistributedTracing
{
    class Program
    {
        static ActivitySource s_source = new ActivitySource("Sample.DistributedTracing");

        static async Task Main(string[] args)
        {
            await DoSomeWork();
            Console.WriteLine("Example work done");
        }

        static async Task DoSomeWork()
        {
            using (Activity a = s_source.StartActivity("SomeWork"))
            {
                await StepOne();
                await StepTwo();
            }
        }

        static async Task StepOne()
        {
            using (Activity a = s_source.StartActivity("StepOne"))
            {
                await Task.Delay(500);
            }
        }

        static async Task StepTwo()
        {
            using (Activity a = s_source.StartActivity("StepTwo"))
            {
                await Task.Delay(1000);
            }
        }
    }
}

```

Running the app does not collect any trace data yet:

```

> dotnet run
Example work done

```

Collect using OpenTelemetry

[OpenTelemetry](#) is a vendor-neutral open-source project supported by the [Cloud Native Computing Foundation](#) that aims to standardize generating and collecting telemetry for cloud-native software. In this example, you will collect and display distributed trace information on the console though OpenTelemetry can be reconfigured to send it elsewhere. For more information, see the [OpenTelemetry getting started guide](#).

Add the [OpenTelemetry.Exporter.Console](#) NuGet package.

```
dotnet add package OpenTelemetry.Exporter.Console
```

Update Program.cs with additional OpenTelemetry `using` directives:

```
using OpenTelemetry;
using OpenTelemetry.Resources;
using OpenTelemetry.Trace;
using System;
using System.Diagnostics;
using System.Threading.Tasks;
```

Update `Main()` to create the OpenTelemetry TracerProvider:

```
public static async Task Main()
{
    using var tracerProvider = Sdk.CreateTracerProviderBuilder()
        .SetResourceBuilder(ResourceBuilder.CreateDefault().AddService("MySample"))
        .AddSource("Sample.DistributedTracing")
        .AddConsoleExporter()
        .Build();

    await DoSomeWork();
    Console.WriteLine("Example work done");
}
```

Now the app collects distributed trace information and displays it to the console:

```
> dotnet run
Activity.Id:          00-7759221f2c5599489d455b84fa0f90f4-6081a9b8041cd840-01
Activity.ParentId:    00-7759221f2c5599489d455b84fa0f90f4-9a52f72c08a9d447-01
Activity.DisplayName: StepOne
Activity.Kind:        Internal
Activity.StartTime:   2021-03-18T10:46:46.8649754Z
Activity.Duration:    00:00:00.5069226
Resource associated with Activity:
  service.name: MySample
  service.instance.id: 909a4624-3b2e-40e4-a86b-4a2c8003219e

Activity.Id:          00-7759221f2c5599489d455b84fa0f90f4-d2b283db91cf774c-01
Activity.ParentId:    00-7759221f2c5599489d455b84fa0f90f4-9a52f72c08a9d447-01
Activity.DisplayName: StepTwo
Activity.Kind:        Internal
Activity.StartTime:   2021-03-18T10:46:47.3838737Z
Activity.Duration:    00:00:01.0142278
Resource associated with Activity:
  service.name: MySample
  service.instance.id: 909a4624-3b2e-40e4-a86b-4a2c8003219e

Activity.Id:          00-7759221f2c5599489d455b84fa0f90f4-9a52f72c08a9d447-01
Activity.DisplayName: SomeWork
Activity.Kind:        Internal
Activity.StartTime:   2021-03-18T10:46:46.8634510Z
Activity.Duration:    00:00:01.5402045
Resource associated with Activity:
  service.name: MySample
  service.instance.id: 909a4624-3b2e-40e4-a86b-4a2c8003219e

Example work done
```

Sources

In the example code, you invoked `AddSource("Sample.DistributedTracing")` so that OpenTelemetry would capture the Activities produced by the `ActivitySource` that was already present in the code:

```
static ActivitySource s_source = new ActivitySource("Sample.DistributedTracing");
```

Telemetry from any ActivitySource can be captured by calling `AddSource()` with the source's name.

Exporters

The console exporter is helpful for quick examples or local development but in a production deployment you will probably want to send traces to a centralized store. OpenTelemetry supports various destinations using different [exporters](#). For more information about configuring OpenTelemetry, see the [OpenTelemetry getting started guide](#).

Collect traces using Application Insights

Distributed tracing telemetry is automatically captured after configuring the Application Insights SDK for [ASP.NET](#) or [ASP.NET Core](#) apps, or by enabling [code-less instrumentation](#).

For more information, see the [Application Insights distributed tracing documentation](#).

NOTE

Currently, Application Insights only supports collecting specific well-known Activity instrumentation and ignores new user-added Activities. Application Insights offers [TrackDependency](#) as a vendor-specific API for adding custom distributed tracing information.

Collect traces using custom logic

Developers are free to create their own customized collection logic for Activity trace data. This example collects the telemetry using the [System.Diagnostics.ActivityListener](#) API provided by .NET and prints it to the console.

Prerequisites

- [.NET Core 2.1 SDK](#) or a later version

Create an example application

First you will create an example application that has some distributed trace instrumentation but no trace data is being collected.

```
dotnet new console
```

Applications that target .NET 5 and later already have the necessary distributed tracing APIs included. For apps targeting older .NET versions, add the [System.Diagnostics.DiagnosticSource NuGet package](#) version 5 or greater.

```
dotnet add package System.Diagnostics.DiagnosticSource
```

Replace the contents of the generated Program.cs with this example source:

```

using System;
using System.Diagnostics;
using System.Threading.Tasks;

namespace Sample.DistributedTracing
{
    class Program
    {
        static ActivitySource s_source = new ActivitySource("Sample.DistributedTracing");

        static async Task Main(string[] args)
        {
            await DoSomeWork();
            Console.WriteLine("Example work done");
        }

        static async Task DoSomeWork()
        {
            using (Activity a = s_source.StartActivity("SomeWork"))
            {
                await StepOne();
                await StepTwo();
            }
        }

        static async Task StepOne()
        {
            using (Activity a = s_source.StartActivity("StepOne"))
            {
                await Task.Delay(500);
            }
        }

        static async Task StepTwo()
        {
            using (Activity a = s_source.StartActivity("StepTwo"))
            {
                await Task.Delay(1000);
            }
        }
    }
}

```

Running the app does not collect any trace data yet:

```

> dotnet run
Example work done

```

Add code to collect the traces

Update Main() with this code:

```

static async Task Main(string[] args)
{
    Activity.DefaultIdFormat = ActivityIdFormat.W3C;
    Activity.ForceDefaultIdFormat = true;

    Console.WriteLine("      {0,-15} {1,-60} {2,-15}", "OperationName", "Id", "Duration");
    ActivitySource.AddActivityListener(new ActivityListener()
    {
        ShouldListenTo = (source) => true,
        Sample = (ref ActivityCreationOptions<ActivityContext> options) =>
ActivitySamplingResult.AllDataAndRecorded,
        ActivityStarted = activity => Console.WriteLine("Started: {0,-15} {1,-60}",
activity.OperationName, activity.Id),
        ActivityStopped = activity => Console.WriteLine("Stopped: {0,-15} {1,-60} {2,-15}",
activity.OperationName, activity.Id, activity.Duration)
    });

    await DoSomeWork();
    Console.WriteLine("Example work done");
}

```

The output now includes logging:

```

> dotnet run
      OperationName   Id                               Duration
Started: SomeWork  00-bdb5fafffc2fc1548b6ba49a31c4a0ae0-c447fb302059784f-01
Started: StepOne   00-bdb5fafffc2fc1548b6ba49a31c4a0ae0-a7c77a4e9a02dc4a-01
Stopped: StepOne  00-bdb5fafffc2fc1548b6ba49a31c4a0ae0-a7c77a4e9a02dc4a-01  00:00:00.5093849
Started: StepTwo   00-bdb5fafffc2fc1548b6ba49a31c4a0ae0-9210ad536cae9e4e-01
Stopped: StepTwo  00-bdb5fafffc2fc1548b6ba49a31c4a0ae0-9210ad536cae9e4e-01  00:00:01.0111847
Stopped: SomeWork  00-bdb5fafffc2fc1548b6ba49a31c4a0ae0-c447fb302059784f-01  00:00:01.5236391
Example work done

```

Setting [DefaultIdFormat](#) and [ForceDefaultIdFormat](#) is optional but helps ensure the sample produces similar output on different .NET runtime versions. .NET 5 uses the W3C TraceContext ID format by default but earlier .NET versions default to using [Hierarchical](#) ID format. For more information, see [Activity IDs](#).

[System.Diagnostics.ActivityListener](#) is used to receive callbacks during the lifetime of an Activity.

- [ShouldListenTo](#) - Each Activity is associated with an [ActivitySource](#), which acts as its namespace and producer. This callback is invoked once for each [ActivitySource](#) in the process. Return true if you are interested in performing sampling or being notified about start/stop events for Activities produced by this source.
- [Sample](#) - By default [StartActivity](#) does not create an Activity object unless some [ActivityListener](#) indicates it should be sampled. Returning [AllDataAndRecorded](#) indicates that the Activity should be created, [IsAllDataRequested](#) should be set to true, and [ActivityTraceFlags](#) will have the [Recorded](#) flag set. [IsAllDataRequested](#) can be observed by the instrumented code as a hint that a listener wants to ensure that auxiliary Activity information such as Tags and Events are populated. The Recorded flag is encoded in the W3C TraceContext ID and is a hint to other processes involved in the distributed trace that this trace should be sampled.
- [ActivityStarted](#) and [ActivityStopped](#) are called when an Activity is started and stopped respectively. These callbacks provide an opportunity to record relevant information about the Activity or potentially to modify it. When an Activity has just started, much of the data may still be incomplete and it will be populated before the Activity stops.

Once an [ActivityListener](#) has been created and the callbacks are populated, calling [ActivitySource.AddActivityListener\(ActivityListener\)](#) initiates invoking the callbacks. Call [ActivityListener.Dispose\(\)](#) to stop the flow of callbacks. Be aware that in multi-threaded code, callback notifications in progress could be received while [Dispose\(\)](#) is running or even shortly after it has returned.

Symbols

9/20/2022 • 3 minutes to read • [Edit Online](#)

Symbols are useful for debugging and other diagnostic tools. The contents of symbol files vary between languages, compilers, and platforms. At a very high level symbols are a mapping between the source code and the binary produced by the compiler. These mappings are used by tools like [Visual Studio](#) and [Visual Studio Code](#) to resolve source line number information or local variable names.

The [Windows documentation on symbols](#) contain more detailed information on symbols for Windows, although many of the concepts apply to other platforms as well.

Learn about .NET's Portable PDB format

.NET Core introduces a new symbol file (PDB) format - the portable PDB. Unlike traditional PDBs which are Windows-only, portable PDBs can be created and read on all platforms.

What is a PDB?

A PDB file is an auxiliary file produced by a compiler to provide other tools, especially debuggers, information about what is in the main executable file and how it was produced. For example, a debugger reads a PDB to map foo.cs line 12 to the right executable location so that it can set a breakpoint. The Windows PDB format has been around a long time, and it evolved from other native debugging symbol formats which were even older. It started out its life as a format for native (C/C++) programs. For the first release of the .NET Framework, the Windows PDB format was extended to support .NET.

The Portable PDB format was introduced in .NET Core, and it's used by default when targeting .NET Core and .NET 5+. When targeting .NET Framework, you can enable Portable PDB symbols by specifying

`<DebugType>portable</DebugType>` in your project file. The Portable PDB format is based on ECMA-335 metadata format. For more information, see [Portable PDB v1.0: Format Specification](#). Diagnostic tools can use the [System.Reflection.Metadata](#) library to read Portable PDB files (for an example, see [System.Reflection.Metadata.Document](#)).

Use the correct PDB format for your scenario

Neither portable PDBs nor Windows PDBs are supported everywhere, so you need to consider where your project will want to be used and debugged to decide which format to use. If you have a project that you want to be able to use and debug in both formats, you can use different build configurations and build the project twice to support both types of consumer.

Support for Portable PDBs

Portable PDBs can be read on any operating systems and is the recommended symbol format for managed code, but there are a number of legacy tools and applications where they aren't supported:

- Applications targeting .NET Framework 4.7.1 or earlier: printing stack traces with mappings back to line numbers (such as in an ASP.NET error page). The name of methods is unaffected, only the source file names and line numbers are unsupported.
- Using .NET decompilers such as ildasm or .NET reflector and expecting to see source line mappings or local parameter names.
- The latest versions of [DIA](#) and tools using it for reading symbols, such as WinDBG support Portable PDBs, but older versions do not.

- There may be older versions of profilers that do not support portable PDBs.

To use portable PDBs on tools that do not support them, you can try using [Pdb2Pdb](#) which converts between Portable PDBs and Windows PDBs.

Support for Windows PDBs

Windows PDBs can only be written or read on Windows. Using Windows PDBs for managed code is obsolete and is only needed for legacy tools. It is recommended that you use portable PDBs instead of Windows PDBs as some newer compiler features that are implemented for only portable PDBs.

See also

- [dotnet-symbol](#) can be used to download symbol files for framework binaries
- [Windows documentation on symbols](#)

Diagnostics client library

9/20/2022 • 4 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 3.0 SDK and later versions for target apps, .NET Standard 2.0 to use the library.

`Microsoft.Diagnostics.NETCore.Client` (also known as the Diagnostics client library) is a managed library that lets you interact with .NET Core runtime (CoreCLR) for various diagnostics related tasks, such as tracing via [EventPipe](#), requesting a dump, or attaching an `ICorProfiler`. This library is the backing library behind many diagnostics tools such as [dotnet-counters](#), [dotnet-trace](#), [dotnet-gcdump](#), [dotnet-dump](#), and [dotnet-monitor](#). Using this library, you can write your own diagnostics tools customized for your particular scenario.

You can acquire [Microsoft.Diagnostics.NETCore.Client](#) by adding a `PackageReference` to your project. The package is hosted on [NuGet.org](#).

The samples in the following sections show how to use `Microsoft.Diagnostics.NETCore.Client` library. Some of these examples also show parsing the event payloads by using [TraceEvent](#) library.

Attach to a process and print out all GC events

This snippet shows how to start an EventPipe session using the [.NET runtime provider](#) with the GC keyword at informational level. It also shows how to use the `EventPipeEventSource` class provided by the [TraceEvent library](#) to parse the incoming events and print their names to the console in real time.

```

using Microsoft.Diagnostics.NETCore.Client;
using Microsoft.Diagnostics.Tracing;
using Microsoft.Diagnostics.Tracing.EventPipe;
using Microsoft.Diagnostics.Tracing.Parsers;
using System;
using System.Collections.Generic;
using System.Diagnostics.Tracing;

public class RuntimeGCEventsPrinter
{
    public static void PrintRuntimeGCEvents(int processId)
    {
        var providers = new List<EventPipeProvider>()
        {
            new EventPipeProvider("Microsoft-Windows-DotNETRuntime",
                EventLevel.Informational, (long)ClrTraceEventParser.Keywords.GC)
        };

        var client = new DiagnosticsClient(processId);
        using (EventPipeSession session = client.StartEventPipeSession(providers, false))
        {
            var source = new EventPipeEventSource(session.EventStream);

            source.Clr.All += (TraceEvent obj) => Console.WriteLine(obj.ToString());

            try
            {
                source.Process();
            }
            catch (Exception e)
            {
                Console.WriteLine("Error encountered while processing events");
                Console.WriteLine(e.ToString());
            }
        }
    }
}

```

Write a core dump

This sample shows how to trigger the collection of a [core dump](#) using `DiagnosticsClient`.

```

using Microsoft.Diagnostics.NETCore.Client;

public partial class Dumper
{
    public static void TriggerCoreDump(int processId)
    {
        var client = new DiagnosticsClient(processId);
        client.WriteDump(DumpType.Normal, "/tmp/minidump.dmp");
    }
}

```

Trigger a core dump when CPU usage goes above a threshold

This sample shows how to monitor the `cpu-usage` counter published by the .NET runtime and request a dump when the CPU usage grows beyond a certain threshold.

```

using Microsoft.Diagnostics.NETCore.Client;
using Microsoft.Diagnostics.Tracing;
using Microsoft.Diagnostics.Tracing.EventPipe;
using Microsoft.Diagnostics.Tracing.Parsers;
using System;
using System.Collections.Generic;
using System.Diagnostics.Tracing;

public partial class Dumper
{
    public static void TriggerDumpOnCpuUsage(int processId, int threshold)
    {
        var providers = new List<EventPipeProvider>()
        {
            new EventPipeProvider(
                "System.Runtime",
                EventLevel.Informational,
                (long)ClrTraceEventParser.Keywords.None,
                new Dictionary<string, string>
                {
                    ["EventCounterIntervalSec"] = "1"
                }
            )
        };
        var client = new DiagnosticsClient(processId);
        using (var session = client.StartEventPipeSession(providers))
        {
            var source = new EventPipeEventSource(session.EventStream);
            source.Dynamic.All += (TraceEvent obj) =>
            {
                if (obj.EventName.Equals("EventCounters"))
                {
                    var payloadVal = (IDictionary<string, object>)(obj.PayloadValue(0));
                    var payloadFields = (IDictionary<string, object>)(payloadVal["Payload"]);
                    if (payloadFields["Name"].ToString().Equals("cpu-usage"))
                    {
                        double cpuUsage = Double.Parse(payloadFields["Mean"].ToString());
                        if (cpuUsage > (double)threshold)
                        {
                            client.WriteDump(DumpType.Normal, "/tmp/minidump.dmp");
                        }
                    }
                }
            };
            try
            {
                source.Process();
            }
            catch (Exception) {}
        }
    }
}

```

Trigger a CPU trace for given number of seconds

This sample shows how to trigger an EventPipe session for certain period of time with the default CLR trace keyword as well as the sample profiler. Afterward, it reads the output stream and writes the bytes out to a file. Essentially this is what `dotnet-trace` uses internally to write a trace file.

```

using Microsoft.Diagnostics.Tracing;
using Microsoft.Diagnostics.Tracing.Parsers;
using Microsoft.Diagnostics.NETCore.Client;
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Diagnostics.Tracing;
using System.IO;
using System.Threading.Tasks;

public partial class Tracer
{
    public void TraceProcessForDuration(int processId, int duration, string traceName)
    {
        var cpuProviders = new List<EventPipeProvider>()
        {
            new EventPipeProvider("Microsoft-Windows-DotNETRuntime", EventLevel.Informational,
(Clong)ClrTraceEventParser.Keywords.Default),
            new EventPipeProvider("Microsoft-DotNETCore-SampleProfiler", EventLevel.Informational,
(Clong)ClrTraceEventParser.Keywords.None)
        };
        var client = new DiagnosticsClient(processId);
        using (var traceSession = client.StartEventPipeSession(cpuProviders))
        {
            Task copyTask = Task.Run(async () =>
            {
                using (FileStream fs = new FileStream(traceName, FileMode.Create, FileAccess.Write))
                {
                    await traceSession.EventStream.CopyToAsync(fs);
                }
            });
            Task.WhenAny(copyTask, Task.Delay(TimeSpan.FromMilliseconds(duration * 1000)));
            traceSession.Stop();
        }
    }
}

```

Print names of processes that published a diagnostics channel

This sample shows how to use `DiagnosticsClient.GetPublishedProcesses` API to print the names of the .NET processes that published a diagnostics IPC channel.

```

using Microsoft.Diagnostics.NETCore.Client;
using System;
using System.Diagnostics;
using System.Linq;

public class ProcessTracker
{
    public static void PrintProcessStatus()
    {
        var processes = DiagnosticsClient.GetPublishedProcesses()
            .Select(Process.GetProcessById)
            .Where(process => process != null);

        foreach (var process in processes)
        {
            Console.WriteLine($"{process.ProcessName}");
        }
    }
}

```

Parse events in real time

This sample shows an example where we create two tasks, one that parses the events coming in live with `EventPipeEventSource` and one that reads the console input for a user input signaling the program to end. If the target app exits before the user presses enter, the app exits gracefully. Otherwise, `inputTask` will send the Stop command to the pipe and exit gracefully.

```
using Microsoft.Diagnostics.NETCore.Client;
using Microsoft.Diagnostics.Tracing;
using Microsoft.Diagnostics.Tracing.EventPipe;
using Microsoft.Diagnostics.Tracing.Parsers;
using System;
using System.Collections.Generic;
using System.Diagnostics.Tracing;
using System.Threading.Tasks;

public partial class Tracer
{
    public static void PrintEventsLive(int processId)
    {
        var providers = new List<EventPipeProvider>()
        {
            new EventPipeProvider("Microsoft-Windows-DotNETRuntime",
                EventLevel.Informational, (long)ClrTraceEventParser.Keywords.Default)
        };
        var client = new DiagnosticsClient(processId);
        using (var session = client.StartEventPipeSession(providers, false))
        {

            Task streamTask = Task.Run(() =>
            {
                var source = new EventPipeEventSource(session.EventStream);
                source.Clr.All += (TraceEvent obj) => Console.WriteLine(obj.EventName);
                try
                {
                    source.Process();
                }
                // NOTE: This exception does not currently exist. It is something that needs to be added to
                TraceEvent.
                catch (Exception e)
                {
                    Console.WriteLine("Error encountered while processing events");
                    Console.WriteLine(e.ToString());
                }
            });
            Task inputTask = Task.Run(() =>
            {
                Console.WriteLine("Press Enter to exit");
                while (Console.ReadKey().Key != ConsoleKey.Enter)
                {
                    Task.Delay(TimeSpan.FromMilliseconds(100));
                }
                session.Stop();
            });

            Task.WaitAny(streamTask, inputTask);
        }
    }
}
```

Attach an ICorProfiler profiler

This sample shows how to attach an ICorProfiler to a process via profiler attach.

```
using System;
using Microsoft.Diagnostics.NETCore.Client;

public class Profiler
{
    public static void AttachProfiler(int processId, Guid profilerGuid, string profilerPath)
    {
        var client = new DiagnosticsClient(processId);
        client.AttachProfiler(TimeSpan.FromSeconds(10), profilerGuid, profilerPath);
    }
}
```

Microsoft.Diagnostics.NETCore.Client API

9/20/2022 • 6 minutes to read • [Edit Online](#)

This section describes the APIs of the diagnostics client library.

DiagnosticsClient class

```
public DiagnosticsClient
{
    public DiagnosticsClient(int processId);

    public EventPipeSession StartEventPipeSession(
        IEnumerable<EventPipeProvider> providers,
        bool requestRundown = true,
        int circularBufferMB = 256);

    public void WriteDump(
        DumpType dumpType,
        string dumpPath,
        bool logDumpGeneration = false);

    public async Task WriteDumpAsync(
        DumpType dumpType,
        string dumpPath,
        bool logDumpGeneration,
        CancellationToken token);

    public void AttachProfiler(
        TimeSpan attachTimeout,
        Guid profilerGuid,
        string profilerPath,
        byte[] additionalData = null);

    public void SetStartupProfiler(
        Guid profilerGuid,
        string profilerPath);

    public void ResumeRuntime();

    public void SetEnvironmentVariable(
        string name,
        string value);

    public Dictionary<string, string> GetProcessEnvironment();

    public static IEnumerable<int> GetPublishedProcesses();
}
```

Constructor

```
public DiagnosticsClient(int processId);
```

Creates a new instance of `DiagnosticsClient` for a compatible .NET process running with process ID of `processId`.

`processID` : Process ID of the target application.

StartEventPipeSession methods

```
public EventPipeSession StartEventPipeSession(
    IEnumerable<EventPipeProvider> providers,
    bool requestRundown = true,
    int circularBufferMB = 256);
```

Starts an EventPipe tracing session using the given providers and settings.

- `providers` : An `IEnumerable` of `EventPipeProvider`s to start tracing.
- `requestRundown` : A `bool` specifying whether rundown provider events from the target app's runtime should be requested.
- `circularBufferMB` : An `int` specifying the total size of circular buffer used by the target app's runtime on collecting events.

```
public EventPipeSession StartEventPipeSession(EventPipeProvider providers, bool requestRundown=true, int
circularBufferMB=256)
```

- `providers` : An `EventPipeProvider` to start tracing.
- `requestRundown` : A `bool` specifying whether rundown provider events from the target app's runtime should be requested.
- `circularBufferMB` : An `int` specifying the total size of circular buffer used by the target app's runtime on collecting events.

NOTE

Rundown events contain payloads that may be needed for post analysis, such as resolving method names of thread samples. Unless you know you do not want this, we recommend setting this to true. In large applications, this may take a while.

- `circularBufferMB` : The size of the circular buffer to be used as a buffer for writing events within the runtime.

WriteDump method

```
public void WriteDump(
    DumpType dumpType,
    string dumpPath,
    bool logDumpGeneration=false);
```

Request a dump for post-mortem debugging of the target application. The type of the dump can be specified using the `DumpType` enum.

- `dumpType` : Type of the dump to be requested.
- `dumpPath` : The path to the dump to be written out to.
- `logDumpGeneration` : If set to `true`, the target application will write out diagnostic logs during dump generation.

```
public void WriteDump(DumpType dumpType, string dumpPath, WriteDumpFlags flags)
```

Request a dump for post-mortem debugging of the target application. The type of the dump can be specified using the `DumpType` enum.

- `dumpType` : Type of the dump to be requested.
- `dumpPath` : The path to the dump to be written out to.

- `flags` : logging and crash report flags. On runtimes less than 6.0, only LoggingEnabled is supported.

```
public async Task WriteDumpAsync(DumpType dumpType, string dumpPath, bool logDumpGeneration,
CancellationToken token)
```

Request a dump for post-mortem debugging of the target application. The type of the dump can be specified using the `DumpType` enum.

- `dumpType` : Type of the dump to be requested.
- `dumpPath` : The path to the dump to be written out to.
- `logDumpGeneration` : If set to `true`, the target application will write out diagnostic logs during dump generation.
- `token` : The token to monitor for cancellation requests.

```
public async Task WriteDumpAsync(DumpType dumpType, string dumpPath, WriteDumpFlags flags, CancellationToken
token)
```

Request a dump for post-mortem debugging of the target application. The type of the dump can be specified using the `DumpType` enum.

- `dumpType` : Type of the dump to be requested.
- `dumpPath` : The path to the dump to be written out to.
- `flags` : logging and crash report flags. On runtimes less than 6.0, only LoggingEnabled is supported.
- `token` : The token to monitor for cancellation requests.

AttachProfiler method

```
public void AttachProfiler(
    TimeSpan attachTimeout,
    Guid profilerGuid,
    string profilerPath,
    byte[] additionalData=null);
```

Request to attach an ICorProfiler to the target application.

- `attachTimeout` : A `TimeSpan` after which attach will be aborted.
- `profilerGuid` : `Guid` of the ICorProfiler to be attached.
- `profilerPath` : Path to the ICorProfiler dll to be attached.
- `additionalData` : Optional additional data that can be passed to the runtime during profiler attach.

SetStartupProfiler method

```
public void SetStartupProfiler(
    Guid profilerGuid,
    string profilerPath);
```

Set a profiler as the startup profiler. It is only valid to issue this command while the runtime is paused at startup.

- `profilerGuid` : `Guid` for the profiler to be attached.
- `profilerPath` : Path to the profiler to be attached.

ResumeRuntime method

```
public void ResumeRuntime();
```

Tell the runtime to resume execution after being paused at startup.

SetEnvironmentVariable method

```
public void SetEnvironmentVariable(  
    string name,  
    string value);
```

Set an environment variable in the target process.

- `name` : The name of the environment variable to set.
- `value` : The value of the environment variable to set.

GetProcessEnvironment

```
public Dictionary<string, string> GetProcessEnvironment()
```

Gets all environment variables and their values from the target process.

GetPublishedProcesses method

```
public static IEnumerable<int> GetPublishedProcesses();
```

Get an `IEnumerable` of process IDs of all the active .NET processes that can be attached to.

EventPipeProvider class

```
public class EventPipeProvider  
{  
    public EventPipeProvider(  
        string name,  
        EventLevel eventLevel,  
        long keywords = 0,  
        IDictionary<string, string> arguments = null)  
  
    public string Name { get; }  
  
    public EventLevel EventLevel { get; }  
  
    public long Keywords { get; }  
  
    public IDictionary<string, string> Arguments { get; }  
  
    public override string ToString();  
  
    public override bool Equals(object obj);  
  
    public override int GetHashCode();  
  
    public static bool operator ==(Provider left, Provider right);  
  
    public static bool operator !=(Provider left, Provider right);  
}
```

Constructor

```
public EventPipeProvider(
    string name,
    EventLevel eventLevel,
    long keywords = 0,
    IDictionary<string, string> arguments = null)
```

Creates a new instance of `EventPipeProvider` with the given provider name, `EventLevel`, keywords, and arguments.

Name property

```
public string Name { get; }
```

Gets the name of the Provider.

EventLevel property

```
public EventLevel EventLevel { get; }
```

Gets the `EventLevel` of the given instance of `EventPipeProvider`.

Keywords property

```
public long Keywords { get; }
```

Gets a value that represents bitmask for keywords of the `EventSource`.

Arguments property

```
public IDictionary<string, string> Arguments { get; }
```

Gets an `IDictionary` of key-value pair strings representing optional arguments to be passed to `EventSource` representing the given `EventPipeProvider`.

Remarks

This class is immutable, because EventPipe does not allow a provider's configuration to be modified during an EventPipe session as of .NET Core 3.1.

EventPipeSession class

```
public class EventPipeSession : IDisposable
{
    public Stream EventStream { get; }
    public void Stop();
}
```

This class represents an ongoing EventPipe session. It is immutable and acts as a handle to an EventPipe session of the given runtime.

EventStream property

```
public Stream EventStream { get; }
```

Gets a `Stream` that can be used to read the event stream.

Stop method

```
public void Stop();
```

Stops the given `EventPipe` session.

DumpType enum

```
public enum DumpType
{
    Normal = 1,
    WithHeap = 2,
    Triage = 3,
    Full = 4
}
```

Represents the type of dump that can be requested.

- `Normal` : Include just the information necessary to capture stack traces for all existing traces for all existing threads in a process. Limited GC heap memory and information.
- `WithHeap` : Includes the GC heaps and information necessary to capture stack traces for all existing threads in a process.
- `Triage` : Include just the information necessary to capture stack traces for all existing traces for all existing threads in a process. Limited GC heap memory and information.
- `Full` : Include all accessible memory in the process. The raw memory data is included at the end, so that the initial structures can be mapped directly without the raw memory information. This option can result in a very large dump file.

Exceptions

Exceptions that are thrown from the library are of type `DiagnosticsClientException` or a derived type.

```
public class DiagnosticsClientException : Exception
```

UnsupportedCommandException

```
public class UnsupportedCommandException : DiagnosticsClientException
```

This may be thrown when the command is not supported by either the library or the target process's runtime.

UnsupportedProtocolException

```
public class UnsupportedProtocolException : DiagnosticsClientException
```

This may be thrown when the target process's runtime is not compatible with the diagnostics IPC protocol used by the library.

ServerNotAvailableException

```
public class ServerNotAvailableException : DiagnosticsClientException
```

This may be thrown when the runtime is not available for diagnostics IPC commands, such as early during runtime startup before the runtime is ready for diagnostics commands, or when the runtime is shutting down.

ServerErrorException

```
public class ServerErrorException : DiagnosticsClientException
```

This may be thrown when the runtime responds with an error to a given command.

.NET runtime events

9/20/2022 • 2 minutes to read • [Edit Online](#)

The .NET runtime (CoreCLR) emits various events that can be used to diagnose issues with your .NET application that can be consumed via various mechanisms such as [ETW](#), [LTTng](#), and [EventPipe](#).

This document serves as a reference on the events that are fired by .NET Core runtime.

For runtime events in .NET Framework, see [CLR ETW Events](#).

In this section

[Contention Events](#)

These events collect information about monitor lock contentions.

[Garbage Collection Events](#)

These events collect information pertaining to garbage collection. They help in diagnostics and debugging, including determining how many times garbage collection was performed, how much memory was freed during garbage collection, etc.

[Exception Events](#)

These runtime events capture information about exceptions that are thrown.

[Interop Events](#)

These runtime events capture information about Common Intermediate Language (CIL) stub generation.

[Loader and Binder Events](#)

These events collect information relating to loading and unloading assemblies and modules.

[Method Events](#)

These events collect information that is specific to methods. The payload of these events is required for symbol resolution. In addition, these events provide helpful information such as the number of times a method was called.

[Thread Events](#)

These events collect information about worker and I/O threads.

[Type Events](#)

These events collect information about the type system.

.NET runtime contention events

9/20/2022 • 2 minutes to read • [Edit Online](#)

These runtime events capture information about monitor lock contentions such as with `Monitor.Enter` or the C# lock keyword. For more information about how to use these events for diagnostic purposes, see [logging and tracing .NET applications](#)

ContentionStart_V2 event

This event is emitted at the start of a monitor lock contention.

KEYWORD FOR RAISING THE EVENT	LEVEL
<code>ContentionKeyword</code> (0x4000)	Informational (4)

The following table shows event information.

EVENT	EVENT ID	RAISED WHEN
<code>ContentionStart_V2</code>	81	A monitor lock contention starts.

FIELD NAME	DATA TYPE	DESCRIPTION
<code>Flags</code>	<code>win:UInt8</code>	<code>0</code> for managed; <code>1</code> for native.
<code>ClrInstanceID</code>	<code>win:UInt16</code>	Unique ID for the instance of CoreCLR.
<code>LockObjectID</code>	<code>win:Pointer</code>	Address of the lock object.
<code>LockOwnerThreadID</code>	<code>win:Pointer</code>	Address of the thread that owns the lock.

ContentionStop_V1 event

This event is emitted at the end of a monitor lock contention.

KEYWORD FOR RAISING THE EVENT	LEVEL
<code>ContentionKeyword</code> (0x4000)	Informational (4)

The following table shows event information.

EVENT	EVENT ID	RAISED WHEN
<code>ContentionStop_V1</code>	91	A monitor lock contention ends.

FIELD NAME	DATA TYPE	DESCRIPTION
Flags	win:UInt8	0 for managed; 1 for native.
ClrInstanceID	win:UInt16	Unique ID for the instance of CoreCLR.
DurationNs	win:Double	Duration of the contention in nanoseconds.

.NET runtime exception events

9/20/2022 • 2 minutes to read • [Edit Online](#)

These runtime events capture information about exceptions that are thrown. For more information about how to use these events for diagnostic purposes, see [logging and tracing .NET applications](#)

ExceptionThrown_V1 event

KEYWORD FOR RAISING THE EVENT	LEVEL
<code>ExceptionKeyword</code> (0x8000)	Error (1)

The following table shows event information.

EVENT	EVENT ID	RAISED WHEN
<code>ExceptionThrown_V1</code>	80	A managed exception is thrown.
FIELD NAME	DATA TYPE	DESCRIPTION
<code>ExceptionType</code>	<code>win:UnicodeString</code>	Type of the exception; for example, <code>System.NullReferenceException</code> .
<code>ExceptionMessage</code>	<code>win:UnicodeString</code>	Actual exception message.
<code>EIPCodeThrow</code>	<code>win:Pointer</code>	Instruction pointer where exception occurred.
<code>ExceptionHR</code>	<code>win:UInt32</code>	Exception HRESULT .
<code>ExceptionFlags</code>	<code>win:UInt16</code>	 <code>0x01</code> : HasInnerException. <code>0x02</code> : IsNestedException. <code>0x04</code> : IsRethrownException. <code>0x08</code> : IsCorruptedStateException (indicates that the process state is corrupt; see Handling Corrupted State Exceptions). <code>0x10</code> : IsCLSCompliant (an exception that derives from <code>Exception</code> is CLS-compliant; otherwise, it is not CLS-compliant).
<code>ClrInstanceID</code>	<code>win:UInt16</code>	Unique ID for the instance of CLR or CoreCLR.

ExceptionCatchStart event

This event is emitted when a managed exception catch handler begins.

KEYWORD FOR RAISING THE EVENT	LEVEL
ExceptionKeyword (0x8000)	Informational (4)

The following table shows event information.

EVENT	EVENT ID	RAISED WHEN
ExceptionCatchStart	250	A managed exception is handled by the runtime.

FIELD NAME	DATA TYPE	DESCRIPTION
EIPCodeThrow	win:Pointer	Instruction pointer where exception occurred.
MethodID	win:Pointer	Pointer to the method descriptor on the method where exception occurred.
MethodName	win:UnicodeString	Name of the method where exception occurred.
ClrInstanceID	win:UInt16	Unique ID for the instance of CLR or CoreCLR.

ExceptionCatchStop event

This event is emitted when a managed exception catch handler ends.

KEYWORD FOR RAISING THE EVENT	LEVEL
ExceptionKeyword (0x8000)	Informational (4)

The following table shows event information.

EVENT	EVENT ID	RAISED WHEN
ExceptionCatchStop	251	A managed exception catch handler is done.

ExceptionFinallyStart event

This event is emitted when a managed exception finally handler begins.

KEYWORD FOR RAISING THE EVENT	LEVEL
ExceptionKeyword (0x8000)	Informational (4)

The following table shows event information.

EVENT	EVENT ID	RAISED WHEN
<code>ExceptionFinallyStart</code>	252	A managed exception is handled by the runtime.
FIELD NAME	DATA TYPE	DESCRIPTION
<code>EIPCodeThrow</code>	<code>win:Pointer</code>	Instruction pointer where exception occurred.
<code>MethodID</code>	<code>win:Pointer</code>	Pointer to the method descriptor on the method where exception occurred.
<code>MethodName</code>	<code>win:UnicodeString</code>	Name of the method where exception occurred.
<code>ClrInstanceID</code>	<code>win:UInt16</code>	Unique ID for the instance of CLR or CoreCLR.

ExceptionFinallyStop event

This event is emitted when a managed exception catch handler ends.

KEYWORD FOR RAISING THE EVENT	LEVEL
<code>ExceptionKeyword</code> (0x8000)	Informational (4)

The following table shows event information.

EVENT	EVENT ID	RAISED WHEN
<code>ExceptionFinallyStop</code>	253	A managed exception finally handler is done.

ExceptionFilterStart event

This event is emitted when a managed exception filtering begins.

KEYWORD FOR RAISING THE EVENT	LEVEL
<code>ExceptionKeyword</code> (0x8000)	Informational (4)

The following table shows event information.

EVENT	EVENT ID	RAISED WHEN
<code>ExceptionFilterStart</code>	254	A managed exception filtering begins.
FIELD NAME	DATA TYPE	DESCRIPTION
<code>EIPCodeThrow</code>	<code>win:Pointer</code>	Instruction pointer where exception occurred.

FIELD NAME	DATA TYPE	DESCRIPTION
MethodID	win:Pointer	Pointer to the method descriptor on the method where exception occurred.
MethodName	win:UnicodeString	Name of the method where exception occurred.
ClrInstanceID	win:UInt16	Unique ID for the instance of CoreCLR.

ExceptionFilterStop event

This event is emitted when a managed exception filtering ends.

KEYWORD FOR RAISING THE EVENT	LEVEL
ExceptionKeyword (0x8000)	Informational (4)

The following table shows event information.

EVENT	EVENT ID	RAISED WHEN
ExceptionFilteringStart	255	A managed exception filtering ends.

ExceptionThrownStop event

This event is emitted when the runtime is done handling a managed exception that was thrown.

KEYWORD FOR RAISING THE EVENT	LEVEL
ExceptionKeyword (0x8000)	Informational (4)

The following table shows event information.

EVENT	EVENT ID	RAISED WHEN
ExceptionThrownStop	256	A managed exception filtering ends.

.NET runtime garbage collection events

9/20/2022 • 11 minutes to read • [Edit Online](#)

These events collect information pertaining to garbage collection. They help in diagnostics and debugging, including determining how many times garbage collection was performed, how much memory was freed during garbage collection, etc. For more information about how to use these events for diagnostic purposes, see [logging and tracing .NET applications](#)

GCStart_V2 Event

The following table shows the keyword and level:

KEYWORD FOR RAISING THE EVENT	LEVEL
<code>GCKeyword</code> (0x1)	Informational (4)

The following table shows the event information:

EVENT	EVENT ID	RAISED WHEN
<code>GCStart_V1</code>	1	A garbage collection has started.

The following table shows the event data:

FIELD NAME	DATA TYPE	DESCRIPTION
<code>Count</code>	<code>win:UInt32</code>	The <i>n</i> th garbage collection.
<code>Depth</code>	<code>win:UInt32</code>	The generation that is being collected.
<code>Reason</code>	<code>win:UInt32</code>	Why the garbage collection was triggered: 0x0 - Small object heap allocation. 0x1 - Induced. 0x2 - Low memory. 0x3 - Empty. 0x4 - Large object heap allocation. 0x5 - Out of space (for small object heap). 0x6 - Out of space (for large object heap). 0x7 - Induced but not forced as blocking.

FIELD NAME	DATA TYPE	DESCRIPTION
Type	win:UInt32	<p>0x0 - Blocking garbage collection occurred outside background garbage collection.</p> <p>0x1 - Background garbage collection.</p> <p>0x2 - Blocking garbage collection occurred during background garbage collection.</p>
ClrInstanceID	win:UInt16	Unique ID for the instance of CoreCLR.

GCEnd_V1 Event

The following table shows the keyword and level:

KEYWORD FOR RAISING THE EVENT	LEVEL
GCKeyword (0x1)	Informational (4)

The following table shows the event information:

EVENT	EVENT ID	RAISED WHEN
GCEnd_V1	2	The garbage collection has ended.

The following table shows the event data:

FIELD NAME	DATA TYPE	DESCRIPTION
Count	win:UInt32	The <i>n</i> th garbage collection.
Depth	win:UInt32	The generation that was collected.
ClrInstanceID	win:UInt16	Unique ID for the instance of CoreCLR.

GCHeapStats_V2 Event

The following table shows the keyword and level:

KEYWORD FOR RAISING THE EVENT	LEVEL
GCKeyword (0x1)	Informational (4)

The following table shows the event information:

EVENT	EVENT ID	DESCRIPTION
GCHeapStats_V2	4	Shows the heap statistics at the end of each garbage collection.

The following table shows the event data:

FIELD NAME	DATA TYPE	DESCRIPTION
GenerationSize0	win:UInt64	The size, in bytes, of generation 0 memory.
TotalPromotedSize0	win:UInt64	The number of bytes that are promoted from generation 0 to generation 1.
GenerationSize1	win:UInt64	The size, in bytes, of generation 1 memory.
TotalPromotedSize1	win:UInt64	The number of bytes that are promoted from generation 1 to generation 2.
GenerationSize2	win:UInt64	The size, in bytes, of generation 2 memory.
TotalPromotedSize2	win:UInt64	The number of bytes that survived in generation 2 after the last collection.
GenerationSize3	win:UInt64	The size, in bytes, of the large object heap.
TotalPromotedSize3	win:UInt64	The number of bytes that survived in the large object heap after the last collection.
FinalizationPromotedSize	win:UInt64	The total size, in bytes, of the objects that are ready for finalization.
FinalizationPromotedCount	win:UInt64	The number of objects that are ready for finalization.
PinnedObjectCount	win:UInt32	The number of pinned (unmovable) objects.
SinkBlockCount	win:UInt32	The number of synchronization blocks in use.
GCHandleCount	win:UInt32	The number of garbage collection handles in use.
ClrInstanceID	win:UInt16	Unique ID for the instance of CoreCLR.
GenerationSize4	win:UInt64	The size, in bytes, of the pinned object heap.
TotalPromotedSize4	win:UInt64	The number of bytes that survived in the pinned object heap after the last collection.

GCCreateSegment_V1 event

The following table shows the keyword and level:

KEYWORD FOR RAISING THE EVENT	LEVEL
GCKeyword (0x1)	Informational (4)

The following table shows the event information:

EVENT	EVENT ID	RAISED WHEN
GCCreateSegment_V1	5	A new garbage collection segment has been created. In addition, when tracing is enabled on a process that is already running, this event is raised for each existing segment.

The following table shows the event data:

FIELD NAME	DATA TYPE	DESCRIPTION
Address	win:UInt64	The address of the segment.
Size	win:UInt64	The size of the segment.
Type	win:UInt32	0x0 - Small object heap. 0x1 - Large object heap. 0x2 - Read-only heap.
ClrInstanceID	win:UInt16	Unique ID for the instance of CoreCLR.

Note that the size of segments allocated by the garbage collector is implementation-specific and is subject to change at any time, including in periodic updates. Your app should never make assumptions about or depend on a particular segment size, nor should it attempt to configure the amount of memory available for segment allocations.

GCFreeSegment_V1 event

The following table shows the keyword and level:

KEYWORD FOR RAISING THE EVENT	LEVEL
GCKeyword (0x1)	Informational (4)

The following table shows the event information:

EVENT	EVENT ID	RAISED WHEN
GCFreeSegment_V1	6	A garbage collection segment has been released.

The following table shows the event data:

FIELD NAME	DATA TYPE	DESCRIPTION
Address	win:UInt64	The address of the segment.
ClrInstanceID	win:UInt16	Unique ID for the instance of CoreCLR.

GCRestartEEBegin_V1 event

The following table shows the keyword and level:

KEYWORD FOR RAISING THE EVENT	LEVEL
GCKeyword (0x1)	Informational (4)

The following table shows the event information:

EVENT	EVENT ID	RAISED WHEN
GCRestartEEBegin_V1	7	Resumption from common language runtime suspension has begun.

This event does not have any event data.

GCRestartEEEEnd_V1 event

The following table shows the keyword and level:

KEYWORD FOR RAISING THE EVENT	LEVEL
GCKeyword (0x1)	Informational (4)

The following table shows the event information:

EVENT	EVENT ID	RAISED WHEN
GCRestartEEEEnd_V1	3	Resumption from common language runtime suspension has ended.

This event does not have any event data.

GCSuspendEEEEnd_V1 event

The following table shows the keyword and level:

KEYWORD FOR RAISING THE EVENT	LEVEL
GCKeyword (0x1)	Informational (4)

The following table shows the event information:

EVENT	EVENT ID	RAISED WHEN

EVENT	EVENT ID	RAISED WHEN
GCSuspendEEEnd_V1	8	End of suspension of the execution engine for garbage collection.

This event does not have any event data.

GCSuspendEEBegin_V1 event

The following table shows the keyword and level:

KEYWORD FOR RAISING THE EVENT	LEVEL
GCKeyword (0x1)	Informational (4)

The following table shows the event information:

EVENT	EVENT ID	RAISED WHEN
GCSuspendEEBegin_V1	8	Start of suspension of the execution engine for garbage collection.
FIELD NAME	DATA TYPE	DESCRIPTION
Count	win:UInt32	The <i>n</i> th garbage collection.
Reason	win:UInt32	Reason for EE suspension. 0x0 : Suspend for Other 0x1 : Suspend for GC. 0x2 : Suspend for AppDomain shutdown. 0x3 : Suspend for code pitching. 0x4 : Suspend for shutdown. 0x5 : Suspend for debugger. 0x6 : Suspend for GC Prep. 0x7 : Suspend for debugger sweep

GCAccumulationTick_V3 event

The following table shows the keyword and level:

KEYWORD FOR RAISING THE EVENT	LEVEL
GCKeyword (0x1)	Verbose (4)

The following table shows the event information:

EVENT	EVENT ID	RAISED WHEN
GCAllocationTick_V3	10	Each time approximately 100 KB is allocated.

The following table shows the event data:

FIELD NAME	DATA TYPE	DESCRIPTION
AllocationAmount	win:UInt32	The allocation size, in bytes. This value is accurate for allocations that are less than the length of a ULONG (4,294,967,295 bytes). If the allocation is greater, this field contains a truncated value. Use AllocationAmount64 for very large allocations.
AllocationKind	win:UInt32	<p>0x0 - Small object allocation (allocation is in small object heap).</p> <p>0x1 - Large object allocation (allocation is in large object heap).</p>
AllocationAmount64	win:UInt64	The allocation size, in bytes. This value is accurate for very large allocations.
TypeId	win:Pointer	The address of the MethodTable. When there are several types of objects that were allocated during this event, this is the address of the MethodTable that corresponds to the last object allocated (the object that caused the 100 KB threshold to be exceeded).
TypeName	win:UnicodeString	The name of the type that was allocated. When there are several types of objects that were allocated during this event, this is the type of the last object allocated (the object that caused the 100 KB threshold to be exceeded).
HeapIndex	win:UInt32	The heap where the object was allocated. This value is 0 (zero) when running with workstation garbage collection.
Address	win:Pointer	The address of the last allocated object.
ClrInstanceID	win:UInt16	Unique ID for the instance of CoreCLR.

GCCreateConcurrentThread_V1 event

The following table shows the keyword and level:

KEYWORD FOR RAISING THE EVENT	LEVEL
<code>GCKeyword</code> (0x1)	Informational (4)
<code>ThreadingKeyword</code> (0x10000)	Informational (4)

The following table shows the event information:

EVENT	EVENT ID	RAISED WHEN
<code>GCCreateConcurrentThread_V1</code>	11	Concurrent garbage collection thread was created.

This event does not have any event data.

GCTerminateConcurrentThread_V1 event

The following table shows the keyword and level:

KEYWORD FOR RAISING THE EVENT	LEVEL
<code>GCKeyword</code> (0x1)	Informational (4)
<code>ThreadingKeyword</code> (0x10000)	Informational (4)

The following table shows the event information:

EVENT	EVENT ID	RAISED WHEN
<code>GCTerminateConcurrentThread_V1</code>	12	Concurrent garbage collection thread was terminated.

This event does not have any event data.

GCFinalizersBegin_V1 event

The following table shows the keyword and level:

KEYWORD FOR RAISING THE EVENT	LEVEL
<code>GCKeyword</code> (0x1)	Informational (4)

The following table shows the event information:

EVENT	EVENT ID	RAISED WHEN
<code>GCFinalizersBegin_V1</code>	14	The start of running finalizers.

This event does not have any event data.

GCFinalizersEnd_V1 event

The following table shows the keyword and level:

KEYWORD FOR RAISING THE EVENT	LEVEL
GCKeyword (0x1)	Informational (4)

The following table shows the event information:

EVENT	EVENT ID	RAISED WHEN
GCFinalizersEnd_V1	13	The end of running finalizers.

The following table shows the event data:

FIELD NAME	DATA TYPE	DESCRIPTION
Count	win:UInt32	The number of finalizers that were run.
ClrInstanceId	win:UInt16	Unique ID for the instance of CLR or CoreCLR.

SetGCHandle event

The following table shows the keyword and level:

KEYWORD FOR RAISING THE EVENT	LEVEL
GCHandleKeyword (0x2)	Informational (4)

The following table shows the event information:

EVENT	EVENT ID	RAISED WHEN
SetGCHandle	30	A GC Handle has been set.

The following table shows the event data:

FIELD NAME	DATA TYPE	DESCRIPTION
HandleID	win:Pointer	The address of the allocated handle.
ObjectID	win:Pointer	The address of the object whose handle was created.

FIELD NAME	DATA TYPE	DESCRIPTION
Kind	win:UInt32	The type of GC handle that was set. 0x0 : WeakShort 0x1 : WeakLong 0x2 : Strong 0x3 : Pinned 0x4 : Variable 0x5 : RefCounted 0x6 : Dependent 0x7 : AsyncPinned 0x8 : SizedRef
Generation	win:UInt32	The generation of the object whose handle was created.
AppDomainID	win:UInt64	The AppDomain ID.
ClrInstanceID	win:UInt16	Unique ID for the instance of CoreCLR.

DestroyGCHandle event

The following table shows the keyword and level:

KEYWORD FOR RAISING THE EVENT	LEVEL
GCHandleKeyword (0x2)	Informational (4)

The following table shows the event information:

EVENT	EVENT ID	RAISED WHEN
DestroyGCHandle	31	A GC Handle is destroyed.

The following table shows the event data:

FIELD NAME	DATA TYPE	DESCRIPTION
HandleID	win:Pointer	The address of the destroyed handle.
ClrInstanceID	win:UInt16	Unique ID for the instance of CoreCLR.

PinObjectAtGCTime event

The following table shows the keyword and level:

KEYWORD FOR RAISING THE EVENT	LEVEL
GCKeyword (0x1)	Verbose (5)

The following table shows the event information:

EVENT	EVENT ID	RAISED WHEN
PinObjectAtGCTime	33	An object was pinned during a GC.

The following table shows the event data:

FIELD NAME	DATA TYPE	DESCRIPTION
HandleID	win:Pointer	The address of the handle.
ObjectID	win:Pointer	The address of the pinned object.
ObjectSize	win:UInt64	The size of the pinned object.
TypeName	win:UnicodeString	The name of the type of the pinned object.
ClrInstanceID	win:UInt16	Unique ID for the instance of CoreCLR.

GCTriggered event

The following table shows the keyword and level:

KEYWORD FOR RAISING THE EVENT	LEVEL
GCKeyword (0x1)	Verbose (5)

The following table shows the event information:

EVENT	EVENT ID	RAISED WHEN
GCTriggered	33	A GC has been triggered.

The following table shows the event data:

FIELD NAME	DATA TYPE	DESCRIPTION

FIELD NAME	DATA TYPE	DESCRIPTION
Reason	win:UInt32	<p>The reason a GC was triggered.</p> <p>0x0 : AllocSmall</p> <p>0x1 : Induced</p> <p>0x2 : LowMemory</p> <p>0x3 : Empty</p> <p>0x4 : AllocLarge</p> <p>0x5 : OutOfSpaceSmallObjectHeap</p> <p>0x6 : OutOfSpaceLargeObjectHeap</p> <p>0x7 : InducedNoForce</p> <p>0x8 : Stress</p> <p>0x9 : InducedLowMemory</p>
ClrInstanceId	win:UInt16	Unique ID for the instance of CoreCLR.

IncreaseMemoryPressure event

The following table shows the keyword and level:

KEYWORD FOR RAISING THE EVENT	LEVEL
GCKeyword (0x1)	Information (4)

The following table shows the event information:

EVENT	EVENT ID	RAISED WHEN
IncreaseMemoryPressure	200	Memory pressure was increased.

The following table shows the event data:

FIELD NAME	DATA TYPE	DESCRIPTION
ClrInstanceId	win:UInt16	Unique ID for the instance of CoreCLR.

DecreaseMemoryPressure event

The following table shows the keyword and level:

KEYWORD FOR RAISING THE EVENT	LEVEL
GCKeyword (0x1)	Information (4)

The following table shows the event information:

EVENT	EVENT ID	RAISED WHEN
DecreaseMemoryPressure	201	Memory pressure was decreased.

The following table shows the event data:

FIELD NAME	DATA TYPE	DESCRIPTION
BytesFreed	win:UInt32	Bytes freed.
ClrInstanceId	win:UInt16	Unique ID for the instance of CoreCLR.

GCMarkWithType event

The following table shows the keyword and level:

KEYWORD FOR RAISING THE EVENT	LEVEL
GCKeyword (0x1)	Information (4)

The following table shows the event information:

EVENT	EVENT ID	RAISED WHEN
GCMarkWithType	202	A GC root has been marked during GC mark phase.

The following table shows the event data:

FIELD NAME	DATA TYPE	DESCRIPTION
HeapNum	win:UInt32	The heap number.
ClrInstanceId	win:UInt16	Unique ID for the instance of CoreCLR.
Type	win:UInt32	<p>The GC root type.</p> <ul style="list-style-type: none"> <code>0x0</code> : Stack <code>0x1</code> : Finalizer <code>0x2</code> : Handle <code>0x3</code> : Older <code>0x4</code> : SizedRef <code>0x5</code> : Overflow
Bytes	win:UInt64	The number of bytes marked.

GCJoin_V2 event

The following table shows the keyword and level:

KEYWORD FOR RAISING THE EVENT	LEVEL
<code>GCKeyword</code> (0x1)	Verbose (5)

The following table shows the event information:

EVENT	EVENT ID	RAISED WHEN
<code>GCJoin_V2</code>	203	A GC thread joined.

The following table shows the event data:

FIELD NAME	DATA TYPE	DESCRIPTION
<code>Heap</code>	<code>win:UInt32</code>	The heap number
<code>JoinTime</code>	<code>win:UInt32</code>	Indicates whether this event is fired at the start of a join or end of a join (0x0 for join start, 0x1 for join end)
<code>JoinType</code>	<code>win:UInt32</code>	<p>The join type.</p> <p>0x0 : Last Join</p> <p>0x1 : Join</p> <p>0x2 : Restart</p> <p>0x3 : First Reverse Join</p> <p>0x4 : Reverse Join</p>
<code>ClrInstanceID</code>	<code>win:UInt16</code>	Unique ID for the instance of CoreCLR.

.NET runtime interop events

9/20/2022 • 2 minutes to read • [Edit Online](#)

These runtime events capture information about Common Intermediate Language (CIL) stub generation. For more information about how to use these events for diagnostic purposes, see [logging and tracing .NET applications](#)

ILStubGenerated event

KEYWORD FOR RAISING THE EVENT	LEVEL	
<code>InteropKeyword</code> (0x2000)	Informational(4)	
EVENT	EVENT ID	RAISED WHEN
<code>ILStubGenerated</code>	88	An IL Stub is generated.
FIELD NAME	DATA TYPE	DESCRIPTION
<code>ModuleID</code>	<code>win:UInt16</code>	The module identifier.
<code>StubMethodID</code>	<code>win:UInt64</code>	The stub method identifier.
<code>StubFlags</code>	<code>win:UInt32</code>	<p>The flags for the stub:</p> <ul style="list-style-type: none"><code>0x1</code> - Reverse interop.<code>0x2</code> - COM interop.<code>0x4</code> - Stub generated by NGen.exe.<code>0x8</code> - Delegate.<code>0x10</code> - Variable argument.<code>0x20</code> - Unmanaged callee.<code>0x40</code> - Struct Marshal
<code>ManagedInteropMethodToken</code>	<code>win:UInt32</code>	The token for the managed interop method.
<code>ManagedInteropMethodNameSpace</code>	<code>win:UnicodeString</code>	The namespace and enclosing type of the managed interop method.
<code>ManagedInteropMethodName</code>	<code>win:UnicodeString</code>	The name of the managed interop method.
<code>ManagedInteropMethodSignature</code>	<code>win:UnicodeString</code>	The signature of the managed interop method.

FIELD NAME	DATA TYPE	DESCRIPTION
NativeMethodSignature	win:UnicodeString	The native method signature.
StubMethodSignature	win:UnicodeString	The stub method signature.
StubMethodILCode	win:UnicodeString	The Common Intermediate Language (CIL) code for the stub method.
ClrInstanceID	win:UInt16	Unique ID for the instance of CLR or CoreCLR.

.NET runtime loader and binder events

9/20/2022 • 9 minutes to read • [Edit Online](#)

These events collect information relating to loading and unloading assemblies and modules. For more information about how to use these events for diagnostic purposes, see [logging and tracing .NET applications](#)

KEYWORD FOR RAISING THE EVENT	EVENT	LEVEL
<code>LoaderKeyword</code> (0x8)	<code>DomainModuleLoad_V1</code>	Informational (4)
EVENT	EVENT ID	DESCRIPTION
<code>DomainModuleLoad_V1</code>	151	Raised when a module is loaded for an application domain.

ModuleLoad_V2 event

KEYWORD FOR RAISING THE EVENT	EVENT	LEVEL
<code>LoaderKeyword</code> (0x8)	<code>DomainModuleLoad_V1</code>	Informational (4)
EVENT	EVENT ID	DESCRIPTION
<code>ModuleLoad_V2</code>	152	Raised when a module is loaded during the lifetime of a process.
FIELD NAME	DATA TYPE	DESCRIPTION
<code>ModuleID</code>	<code>win:UInt64</code>	Unique ID for the module.
<code>AssemblyID</code>	<code>win:UInt64</code>	ID of the assembly in which this module resides.
<code>ModuleFlags</code>	<code>win:UInt32</code>	<p>0x1: Domain neutral module.</p> <p>0x2: Module has a native image.</p> <p>0x4: Dynamic module.</p> <p>0x8: Manifest module.</p>
<code>Reserved1</code>	<code>win:UInt32</code>	Reserved field.
<code>ModuleILPath</code>	<code>win:UnicodeString</code>	Path of the Common Intermediate Language (CIL) image for the module, or dynamic module name if it is a dynamic assembly (null-terminated).

FIELD NAME	DATA TYPE	DESCRIPTION
ModuleNativePath	win:UnicodeString	Path of the module native image, if present (null-terminated).
ClrInstanceId	win:UInt16	Unique ID for the instance of CLR or CoreCLR.
ManagedPdbSignature	win:GUID	GUID signature of the managed program database (PDB) that matches this module.
ManagedPdbAge	win:UInt32	Age number written to the managed PDB that matches this module.
ManagedPdbBuildPath	win:UnicodeString	Path to the location where the managed PDB that matches this module was built. In some cases, this may just be a file name.
NativePdbSignature	win:GUID	GUID signature of the Native Image Generator (NGen) PDB that matches this module, if applicable.
NativePdbAge	win:UInt32	Age number written to the NGen PDB that matches this module, if applicable.
NativePdbBuildPath	win:UnicodeString	Path to the location where the NGen PDB that matches this module was built, if applicable. In some cases, this may just be a file name.

ModuleUnload_V2 event

KEYWORD FOR RAISING THE EVENT	EVENT	LEVEL
LoaderKeyword (0x8)	DomainModuleLoad_V1	Informational (4)
EVENT	EVENT ID	DESCRIPTION
ModuleUnload_V2	153	Raised when a module is unloaded during the lifetime of a process.
FIELD NAME	DATA TYPE	DESCRIPTION
ModuleID	win:UInt64	Unique ID for the module.
AssemblyID	win:UInt64	ID of the assembly in which this module resides.

FIELD NAME	DATA TYPE	DESCRIPTION
ModuleFlags	win:UInt32	0x1: Domain neutral module. 0x2: Module has a native image. 0x4: Dynamic module. 0x8: Manifest module.
Reserved1	win:UInt32	Reserved field.
ModuleILPath	win:UnicodeString	Path of the Common Intermediate Language (CIL) image for the module, or dynamic module name if it is a dynamic assembly (null-terminated).
ModuleNativePath	win:UnicodeString	Path of the module native image, if present (null-terminated).
ClrInstanceId	win:UInt16	Unique ID for the instance of CLR or CoreCLR.
ManagedPdbSignature	win:GUID	GUID signature of the managed program database (PDB) that matches this module.
ManagedPdbAge	win:UInt32	Age number written to the managed PDB that matches this module.
ManagedPdbBuildPath	win:UnicodeString	Path to the location where the managed PDB that matches this module was built. In some cases, this may just be a file name.
NativePdbSignature	win:GUID	GUID signature of the Native Image Generator (NGen) PDB that matches this module, if applicable.
NativePdbAge	win:UInt32	Age number written to the NGen PDB that matches this module, if applicable.
NativePdbBuildPath	win:UnicodeString	Path to the location where the NGen PDB that matches this module was built, if applicable. In some cases, this may just be a file name.

ModuleDCStart_V2 event

KEYWORD FOR RAISING THE EVENT	EVENT	LEVEL
LoaderKeyword (0x8)	DomainModuleLoad_V1	Informational (4)

EVENT	EVENT ID	DESCRIPTION
ModuleDCStart_V2	153	Enumerates modules during a start rundown.
FIELD NAME	DATA TYPE	DESCRIPTION
ModuleID	win:UInt64	Unique ID for the module.
AssemblyID	win:UInt64	ID of the assembly in which this module resides.
ModuleFlags	win:UInt32	<p>0x1: Domain neutral module.</p> <p>0x2: Module has a native image.</p> <p>0x4: Dynamic module.</p> <p>0x8: Manifest module.</p>
Reserved1	win:UInt32	Reserved field.
ModuleILPath	win:UnicodeString	Path of the Common Intermediate Language (CIL) image for the module, or dynamic module name if it is a dynamic assembly (null-terminated).
ModuleNativePath	win:UnicodeString	Path of the module native image, if present (null-terminated).
ClrInstanceId	win:UInt16	Unique ID for the instance of CLR or CoreCLR.
ManagedPdbSignature	win:GUID	GUID signature of the managed program database (PDB) that matches this module.
ManagedPdbAge	win:UInt32	Age number written to the managed PDB that matches this module.
ManagedPdbBuildPath	win:UnicodeString	Path to the location where the managed PDB that matches this module was built. In some cases, this may just be a file name.
NativePdbSignature	win:GUID	GUID signature of the Native Image Generator (NGen) PDB that matches this module, if applicable.
NativePdbAge	win:UInt32	Age number written to the NGen PDB that matches this module, if applicable.
NativePdbBuildPath	win:UnicodeString	Path to the location where the NGen PDB that matches this module was built, if applicable. In some cases, this may just be a file name.

ModuleDCEnd_V2 event

KEYWORD FOR RAISING THE EVENT	EVENT	LEVEL
LoaderKeyword (0x8)	DomainModuleLoad_V1	Informational (4)
EVENT	EVENT ID	DESCRIPTION
ModuleDCEnd_V2	154	Enumerates modules during an end rundown.
FIELD NAME	DATA TYPE	DESCRIPTION
ModuleID	win:UInt64	Unique ID for the module.
AssemblyID	win:UInt64	ID of the assembly in which this module resides.
ModuleFlags	win:UInt32	<p>0x1: Domain neutral module.</p> <p>0x2: Module has a native image.</p> <p>0x4: Dynamic module.</p> <p>0x8: Manifest module.</p>
Reserved1	win:UInt32	Reserved field.
ModuleILPath	win:UnicodeString	Path of the Common Intermediate Language (CIL) image for the module, or dynamic module name if it is a dynamic assembly (null-terminated).
ModuleNativePath	win:UnicodeString	Path of the module native image, if present (null-terminated).
ClrInstanceId	win:UInt16	Unique ID for the instance of CLR or CoreCLR.
ManagedPdbSignature	win:GUID	GUID signature of the managed program database (PDB) that matches this module.
ManagedPdbAge	win:UInt32	Age number written to the managed PDB that matches this module.
ManagedPdbBuildPath	win:UnicodeString	Path to the location where the managed PDB that matches this module was built. In some cases, this may just be a file name.
NativePdbSignature	win:GUID	GUID signature of the Native Image Generator (NGen) PDB that matches this module, if applicable.

FIELD NAME	DATA TYPE	DESCRIPTION
NativePdbAge	win:UInt32	Age number written to the NGen PDB that matches this module, if applicable.
NativePdbBuildPath	win:UnicodeString	Path to the location where the NGen PDB that matches this module was built, if applicable. In some cases, this may just be a file name.

AssemblyLoad_V1 event

KEYWORD FOR RAISING THE EVENT	EVENT	LEVEL
LoaderKeyword (0x8)	DomainModuleLoad_V1	Informational (4)
EVENT	EVENT ID	DESCRIPTION
AssemblyLoad_V1	154	Raised when an assembly is loaded.

FIELD NAME	DATA TYPE	DESCRIPTION
AssemblyID	win:UInt64	Unique ID for the assembly.
AppDomainID	win:UInt64	ID of the domain of this assembly.
BindingID	win:UInt64	ID that uniquely identifies the assembly binding.
AssemblyFlags	win:UInt32	0x1: Domain neutral assembly. 0x2: Dynamic assembly. 0x4: Assembly has a native image. 0x8: Collectible assembly.
AssemblyName	win:UnicodeString	Fully qualified assembly name.
ClrInstanceID	win:UInt16	Unique ID for the instance of CoreCLR.

AssemblyUnload_V1 event

KEYWORD FOR RAISING THE EVENT	EVENT	LEVEL
LoaderKeyword (0x8)	DomainModuleLoad_V1	Informational (4)
EVENT	EVENT ID	DESCRIPTION
FireAssemblyUnload_V1	155	Raised when an assembly is loaded.

FIELD NAME	DATA TYPE	DESCRIPTION
AssemblyID	win:UInt64	Unique ID for the assembly.
AppDomainID	win:UInt64	ID of the domain of this assembly.
BindingID	win:UInt64	ID that uniquely identifies the assembly binding.
AssemblyFlags	win:UInt32	0x1: Domain neutral assembly. 0x2: Dynamic assembly. 0x4: Assembly has a native image. 0x8: Collectible assembly.
AssemblyName	win:UnicodeString	Fully qualified assembly name.
ClrInstanceID	win:UInt16	Unique ID for the instance of CoreCLR.

AssemblyDCStart_V1 event

KEYWORD FOR RAISING THE EVENT	EVENT	LEVEL
LoaderKeyword (0x8)	DomainModuleLoad_V1	Informational (4)
EVENT	EVENT ID	DESCRIPTION
AssemblyDCStart_V1	155	Enumerates assemblies during a start rundown.
FIELD NAME	DATA TYPE	DESCRIPTION
AssemblyID	win:UInt64	Unique ID for the assembly.
AppDomainID	win:UInt64	ID of the domain of this assembly.
BindingID	win:UInt64	ID that uniquely identifies the assembly binding.
AssemblyFlags	win:UInt32	0x1: Domain neutral assembly. 0x2: Dynamic assembly. 0x4: Assembly has a native image. 0x8: Collectible assembly.
AssemblyName	win:UnicodeString	Fully qualified assembly name.
ClrInstanceID	win:UInt16	Unique ID for the instance of CoreCLR.

AssemblyLoadStart event

KEYWORD FOR RAISING THE EVENT	EVENT	LEVEL
<code>Binder</code> (0x4)	<code>AssemblyLoadStart</code>	Informational (4)
EVENT	EVENT ID	DESCRIPTION
<code>AssemblyLoadStart</code>	290	An assembly load has been requested.
FIELD NAME	DATA TYPE	DESCRIPTION
<code>AssemblyName</code>	<code>win:UnicodeString</code>	Name of assembly name.
<code>AssemblyPath</code>	<code>win:UnicodeString</code>	Path of assembly name.
<code>RequestingAssembly</code>	<code>win:UnicodeString</code>	Name of the requesting ("parent") assembly.
<code>AssemblyLoadContext</code>	<code>win:UnicodeString</code>	Load context of the assembly.
<code>RequestingAssemblyLoadContext</code>	<code>win:UnicodeString</code>	Load context of the requesting ("parent") assembly.
<code>ClrInstanceID</code>	<code>win:UInt16</code>	Unique ID for the instance of CoreCLR.

AssemblyLoadStop event

KEYWORD FOR RAISING THE EVENT	EVENT	LEVEL
<code>Binder</code> (0x4)	<code>AssemblyLoadStart</code>	Informational (4)
EVENT	EVENT ID	DESCRIPTION
<code>AssemblyLoadStart</code>	291	An assembly load has been requested.
FIELD NAME	DATA TYPE	DESCRIPTION
<code>AssemblyName</code>	<code>win:UnicodeString</code>	Name of assembly name.
<code>AssemblyPath</code>	<code>win:UnicodeString</code>	Path of assembly name.
<code>RequestingAssembly</code>	<code>win:UnicodeString</code>	Name of the requesting ("parent") assembly.
<code>AssemblyLoadContext</code>	<code>win:UnicodeString</code>	Load context of the assembly.
<code>RequestingAssemblyLoadContext</code>	<code>win:UnicodeString</code>	Load context of the requesting ("parent") assembly.

FIELD NAME	DATA TYPE	DESCRIPTION
Success	win:Boolean	Whether the assembly load succeeded.
ResultAssemblyName	win:UnicodeString	The name of assembly that was loaded.
ResultAssemblyPath	win:UnicodeString	The path of the assembly that was loaded from.
Cached	win:UnicodeString	Whether the load was cached.
ClrInstanceID	win:UInt16	Unique ID for the instance of CoreCLR.

ResolutionAttempted event

KEYWORD FOR RAISING THE EVENT		
Binder (0x4)		Informational (4)
EVENT	EVENT ID	DESCRIPTION
ResolutionAttempted	292	An assembly load has been requested.
FIELD NAME	DATA TYPE	DESCRIPTION
AssemblyName	win:UnicodeString	Name of assembly name.
Stage	win:UInt16	<p>The resolution stage.</p> <p>0: Find in load.</p> <p>1: Assembly Load Context</p> <p>2: Application assemblies.</p> <p>3: Default assembly load context fallback.</p> <p>4: Resolve satellite assembly.</p> <p>5: Assembly load context resolving.</p> <p>6: AppDomain assembly resolving.</p>
AssemblyLoadContext	win:UnicodeString	Load context of the assembly.

FIELD NAME	DATA TYPE	DESCRIPTION
Result	win:UInt16	The result of resolution attempt. 0: Success 1: Assembly NotFound 2: Incompatible Version 3: Mismatched Assembly Name 4: Failure 5: Exception
ResultAssemblyName	win:UnicodeString	The name of assembly that was resolved.
ResultAssemblyPath	win:UnicodeString	The path of the assembly that was resolved from.
ErrorMessage	win:UnicodeString	Error message if there is an exception.
ClrInstanceID	win:UInt16	Unique ID for the instance of CoreCLR.

AssemblyLoadContextResolvingHandlerInvoked event

KEYWORD FOR RAISING THE EVENT	LEVEL	
Binder (0x4)	Informational (4)	
EVENT	EVENT ID	DESCRIPTION
AssemblyLoadContextResolvingHandlerInvoked	293	An AssemblyLoadContext.Resolving handler has been invoked.

FIELD NAME	DATA TYPE	DESCRIPTION
AssemblyName	win:UnicodeString	Name of assembly name.
HandlerName	win:UnicodeString	Name of the handler invoked.
AssemblyLoadContext	win:UnicodeString	Load context of the assembly.
ResultAssemblyName	win:UnicodeString	The name of assembly that was resolved.
ResultAssemblyPath	win:UnicodeString	The path of the assembly that was resolved from.
ClrInstanceID	win:UInt16	Unique ID for the instance of CoreCLR.

AppDomainAssemblyResolveHandlerInvoked event

KEYWORD FOR RAISING THE EVENT	LEVEL	
Binder (0x4)	Informational (4)	
EVENT	EVENT ID	DESCRIPTION
AppDomainAssemblyResolveHandlerInvoke	294	An AppDomain.AssemblyResolve handler has been invoked.
FIELD NAME	DATA TYPE	DESCRIPTION
AssemblyName	win:UnicodeString	Name of assembly name.
HandlerName	win:UnicodeString	Name of the handler invoked.
ResultAssemblyName	win:UnicodeString	The name of assembly that was resolved.
ResultAssemblyPath	win:UnicodeString	The path of the assembly that was resolved from.
ClrInstanceId	win:UInt16	Unique ID for the instance of CoreCLR.

AssemblyLoadFromResolveHandlerInvoked event

KEYWORD FOR RAISING THE EVENT	LEVEL	
Binder (0x4)	Informational (4)	
EVENT	EVENT ID	DESCRIPTION
AssemblyLoadFromResolveHandlerInvoke	295	An Assembly.LoadFrom handler has been invoked.
FIELD NAME	DATA TYPE	DESCRIPTION
AssemblyName	win:UnicodeString	Name of assembly name.
IsTrackedLoad	win:Boolean	Whether the assembly load is tracked.
RequestingAssemblyPath	win:UnicodeString	The path of the requesting assembly.
ComputedRequestedAssemblyPath	win:UnicodeString	The path of the assembly that was requested.
ClrInstanceId	win:UInt16	Unique ID for the instance of CoreCLR.

KnownPathProbed event

KEYWORD FOR RAISING THE EVENT	LEVEL	
<code>Binder</code> (0x4)	Informational (4)	
EVENT	EVENT ID	DESCRIPTION
<code>KnownPathProbed</code>	296	A known path was probed for an assembly.
FIELD NAME	DATA TYPE	DESCRIPTION
<code>FilePath</code>	<code>win:UnicodeString</code>	Path probed.
<code>Source</code>	<code>win:UInt16</code>	<p>Source of the path probed.</p> <p>0x0:Application Assemblies.</p> <p>0x1:App native image path.</p> <p>0x2:App path.</p> <p>0x3:Platform resource roots.</p> <p>0x4:Satellite Subdirectory.</p>
<code>Result</code>	<code>win:UInt32</code>	HRESULT for the probe.
<code>ClrInstanceID</code>	<code>win:UInt16</code>	Unique ID for the instance of CoreCLR.

.NET runtime method events

9/20/2022 • 11 minutes to read • [Edit Online](#)

These events collect information that is specific to methods. The payload of these events is required for symbol resolution. In addition, these events provide helpful information such as methods that are loaded and unloaded. For more information about how to use these events for diagnostic purposes, see [logging and tracing .NET applications](#)

All method events have a level of "Informational (4)". All method verbose events have a level of "Verbose (5)".

All method events are raised by the `JITKeyword` (0x10) keyword or the `NGenKeyword` (0x20) keyword under the runtime provider, or `JitRundownKeyword` (0x10) or `NGENRundownKeyword` (0x20) under the rundown provider.

The V2 versions of these events include the ReJITID, the V1 versions do not.

MethodLoad_V1 event

The following table shows the event information:

EVENT	EVENT ID	DESCRIPTION
<code>MethodLoad_V1</code>	141	Raised when a method is just-in-time loaded (JIT-loaded) or an NGEN image is loaded. Dynamic and generic methods do not use this version for method loads. JIT helpers never use this version.

KEYWORD FOR RAISING THE EVENT	LEVEL
<code>JITKeyword</code> (0x10) runtime provider	Informational (4)
<code>NGenKeyword</code> (0x20) runtime provider	Informational (4)

FIELD NAME	DATA TYPE	DESCRIPTION
<code>MethodID</code>	<code>win:UInt64</code>	Unique identifier of a method. For JIT helper methods, this is set to the start address of the method.
<code>ModuleID</code>	<code>win:UInt64</code>	Identifier of the module to which this method belongs (0 for JIT helpers).
<code>MethodStartAddress</code>	<code>win:UInt64</code>	Start address of the method.
<code>MethodSize</code>	<code>win:UInt32</code>	Size of the method.
<code>MethodToken</code>	<code>win:UInt32</code>	0 for dynamic methods and JIT helpers.

FIELD NAME	DATA TYPE	DESCRIPTION
MethodFlags	win:UInt32	0x1: Dynamic method. 0x2: Generic method. 0x4: JIT-compiled code method (otherwise NGEN native image code). 0x8: Helper method.
ClrInstanceID	win:UInt16	Unique ID for the instance of CoreCLR.

MethodLoad_V2 event

EVENT	EVENT ID	DESCRIPTION
MethodLoad_V2	141	Raised when a method is just-in-time loaded (JIT-loaded) or an NGEN image is loaded. Dynamic and generic methods do not use this version for method loads. JIT helpers never use this version.

KEYWORD FOR RAISING THE EVENT	LEVEL
JITKeyword (0x10) runtime provider	Informational (4)
NGenKeyword (0x20) runtime provider	Informational (4)

FIELD NAME	DATA TYPE	DESCRIPTION
MethodID	win:UInt64	Unique identifier of a method. For JIT helper methods, this is set to the start address of the method.
ModuleID	win:UInt64	Identifier of the module to which this method belongs (0 for JIT helpers).
MethodStartAddress	win:UInt64	Start address of the method.
MethodSize	win:UInt32	Size of the method.
MethodToken	win:UInt32	0 for dynamic methods and JIT helpers.
MethodFlags	win:UInt32	0x1: Dynamic method. 0x2: Generic method. 0x4: JIT-compiled code method (otherwise NGEN native image code). 0x8: Helper method.

FIELD NAME	DATA TYPE	DESCRIPTION
ReJITID	win:UInt64	ReJIT ID of the method.
ClrInstanceID	win:UInt16	Unique ID for the instance of CoreCLR.

MethodUnLoad_V1 event

EVENT	EVENT ID	DESCRIPTION
MethodUnLoad_V1	142	Raised when a module is unloaded, or an application domain is destroyed. Dynamic methods never use this version for method unloads.

KEYWORD FOR RAISING THE EVENT	LEVEL
JITKeyword (0x10)	Informational (4)
NGenKeyword (0x20)	Informational (4)

FIELD NAME	DATA TYPE	DESCRIPTION
MethodID	win:UInt64	Unique identifier of a method. For JIT helper methods, this is set to the start address of the method.
ModuleID	win:UInt64	Identifier of the module to which this method belongs (0 for JIT helpers).
MethodStartAddress	win:UInt64	Start address of the method.
MethodSize	win:UInt32	Size of the method.
MethodToken	win:UInt32	0 for dynamic methods and JIT helpers.
MethodFlags	win:UInt32	0x1: Dynamic method. 0x2: Generic method. 0x4: JIT-compiled code method (otherwise NGEN native image code). 0x8: Helper method.
ClrInstanceID	win:UInt16	Unique ID for the instance of CoreCLR.

MethodUnLoad_V2 event

EVENT	EVENT ID	DESCRIPTION
MethodUnLoad_V2	142	Raised when a module is unloaded, or an application domain is destroyed. Dynamic methods never use this version for method unloads.
KEYWORD FOR RAISING THE EVENT	LEVEL	
JITKeyword (0x10)	Informational (4)	
NGenKeyword (0x20)	Informational (4)	
FIELD NAME	DATA TYPE	DESCRIPTION
MethodID	win:UInt64	Unique identifier of a method. For JIT helper methods, this is set to the start address of the method.
ModuleID	win:UInt64	Identifier of the module to which this method belongs (0 for JIT helpers).
MethodStartAddress	win:UInt64	Start address of the method.
MethodSize	win:UInt32	Size of the method.
MethodToken	win:UInt32	0 for dynamic methods and JIT helpers.
MethodFlags	win:UInt32	0x1: Dynamic method. 0x2: Generic method. 0x4: JIT-compiled code method (otherwise NGEN native image code). 0x8: Helper method.
ReJITID	win:UInt64	ReJIT ID of the method.
ClrInstanceID	win:UInt16	Unique ID for the instance of CoreCLR.

R2RGetEntryPoint event

EVENT	EVENT ID	DESCRIPTION
R2RGetEntryPoint	159	Raised when an R2R entry point lookup ends.
KEYWORD FOR RAISING THE EVENT	LEVEL	
CompilationDiagnosticKeyword (0x2000000000)	Informational (4)	

FIELD NAME	DATA TYPE	DESCRIPTION
MethodID	win:UInt64	Unique identifier of a R2R method.
MethodNamespace	win:UnicodeString	The namespace of method being looked up.
MethodName	win:UnicodeString	The name of the method being looked up.
MethodSignature	win:UnicodeString	Signature of the method (comma-separated list of type names).
EntryPoint	win:UInt64	The pointer to the entry point of the R2R method
ClrInstanceID	win:UInt16	Unique ID for the instance of CoreCLR.

R2RGetEntryPointStart event

EVENT	EVENT ID	DESCRIPTION
R2RGetEntryPointStart	160	Raised when an R2R entry point lookup starts.
KEYWORD FOR RAISING THE EVENT	LEVEL	
CompilationDiagnosticKeyword (0x2000000000)	Informational (4)	
FIELD NAME	DATA TYPE	DESCRIPTION
MethodID	win:UInt64	Unique identifier of a R2R method.
ClrInstanceID	win:UInt16	Unique ID for the instance of CoreCLR.

MethodLoadVerbose_V1 event

EVENT	EVENT ID	DESCRIPTION
MethodLoadVerbose_V1	143	Raised when a method is JIT-loaded or an NGEN image is loaded. Dynamic and generic methods always use this version for method loads. JIT helpers always use this version.
KEYWORD FOR RAISING THE EVENT	LEVEL	
JITKeyword (0x10)	Informational (4)	
NGenKeyword (0x20)	Informational (4)	

FIELD NAME	DATA TYPE	DESCRIPTION
MethodID	win:UInt64	Unique identifier of the method. For JIT helper methods, set to the start address of the method.
ModuleID	win:UInt64	Identifier of the module to which this method belongs (0 for JIT helpers).
MethodStartAddress	win:UInt64	Start address.
MethodSize	win:UInt32	Method length.
MethodToken	win:UInt32	0 for dynamic methods and JIT helpers.
MethodFlags	win:UInt32	0x1: Dynamic method. 0x2: Generic method. 0x4: JIT-compiled method (otherwise, generated by NGen.exe) 0x8: Helper method.
MethodNameSpace	win:UnicodeString	Full namespace name associated with the method.
MethodName	win:UnicodeString	Full class name associated with the method.
MethodSignature	win:UnicodeString	Signature of the method (comma-separated list of type names).
ClrInstanceID	win:UInt16	Unique ID for the instance of CoreCLR.

MethodLoadVerbose_V2 event

EVENT	EVENT ID	DESCRIPTION
MethodLoadVerbose_V1	143	Raised when a method is JIT-loaded or an NGEN image is loaded. Dynamic and generic methods always use this version for method loads. JIT helpers always use this version.

KEYWORD FOR RAISING THE EVENT	LEVEL
JITKeyword (0x10)	Informational (4)
NGenKeyword (0x20)	Informational (4)

FIELD NAME	DATA TYPE	DESCRIPTION
MethodID	win:UInt64	Unique identifier of the method. For JIT helper methods, set to the start address of the method.
ModuleID	win:UInt64	Identifier of the module to which this method belongs (0 for JIT helpers).
MethodStartAddress	win:UInt64	Start address.
MethodSize	win:UInt32	Method length.
MethodToken	win:UInt32	0 for dynamic methods and JIT helpers.
MethodFlags	win:UInt32	0x1: Dynamic method. 0x2: Generic method. 0x4: JIT-compiled method (otherwise, generated by NGen.exe) 0x8: Helper method.
MethodNameSpace	win:UnicodeString	Full namespace name associated with the method.
MethodName	win:UnicodeString	Full class name associated with the method.
MethodSignature	win:UnicodeString	Signature of the method (comma-separated list of type names).
ReJITID	win:UInt64	ReJIT ID of the method.
ClrInstanceID	win:UInt16	Unique ID for the instance of CoreCLR.

MethodUnLoadVerbose_V1 event

EVENT	EVENT ID	DESCRIPTION
MethodUnLoadVerbose_V1	144	Raised when a dynamic method is destroyed, a module is unloaded, or an application domain is destroyed. Dynamic methods always use this version for method unloads.

KEYWORD FOR RAISING THE EVENT	LEVEL
JITKeyword (0x10)	Informational (4)
NGenKeyword (0x20)	Informational (4)

FIELD NAME	DATA TYPE	DESCRIPTION
MethodID	win:UInt64	Unique identifier of the method. For JIT helper methods, set to the start address of the method.
ModuleID	win:UInt64	Identifier of the module to which this method belongs (0 for JIT helpers).
MethodStartAddress	win:UInt64	Start address.
MethodSize	win:UInt32	Method length.
MethodToken	win:UInt32	0 for dynamic methods and JIT helpers.
MethodFlags	win:UInt32	0x1: Dynamic method. 0x2: Generic method. 0x4: JIT-compiled method (otherwise, generated by NGen.exe) 0x8: Helper method.
MethodNameSpace	win:UnicodeString	Full namespace name associated with the method.
MethodName	win:UnicodeString	Full class name associated with the method.
MethodSignature	win:UnicodeString	Signature of the method (comma-separated list of type names).
ClrInstanceID	win:UInt16	Unique ID for the instance of CoreCLR.

MethodUnLoadVerbose_V2 event

EVENT	EVENT ID	DESCRIPTION
MethodUnLoadVerbose_V2	144	Raised when a dynamic method is destroyed, a module is unloaded, or an application domain is destroyed. Dynamic methods always use this version for method unloads.

KEYWORD FOR RAISING THE EVENT	LEVEL
JITKeyword (0x10)	Informational (4)
NGenKeyword (0x20)	Informational (4)

FIELD NAME	DATA TYPE	DESCRIPTION
MethodID	win:UInt64	Unique identifier of the method. For JIT helper methods, set to the start address of the method.
ModuleID	win:UInt64	Identifier of the module to which this method belongs (0 for JIT helpers).
MethodStartAddress	win:UInt64	Start address.
MethodSize	win:UInt32	Method length.
MethodToken	win:UInt32	0 for dynamic methods and JIT helpers.
MethodFlags	win:UInt32	0x1: Dynamic method. 0x2: Generic method. 0x4: JIT-compiled method (otherwise, generated by NGen.exe) 0x8: Helper method.
MethodNameSpace	win:UnicodeString	Full namespace name associated with the method.
MethodName	win:UnicodeString	Full class name associated with the method.
MethodSignature	win:UnicodeString	Signature of the method (comma-separated list of type names).
ReJITID	win:UInt64	ReJIT ID of the method.
ClrInstanceID	win:UInt16	Unique ID for the instance of CoreCLR.

MethodJittingStarted_V1 event

The following table shows the keyword and level:

KEYWORD FOR RAISING THE EVENT	LEVEL
JITKeyword (0x10)	Verbose (5)
NGenKeyword (0x20)	Verbose (5)

EVENT	EVENT ID	DESCRIPTION
MethodJittingStarted_V1	145	Raised when a method is being JIT-compiled.

FIELD NAME	DATA TYPE	DESCRIPTION
MethodID	win:UInt64	Unique identifier of the method.
ModuleID	win:UInt64	Identifier of the module to which this method belongs.
MethodToken	win:UInt32	0 for dynamic methods and JIT helpers.
MethodILSize	win:UInt32	The size of the Common Intermediate Language (CIL) for the method that is being JIT-compiled.
MethodNameSpace	win:UnicodeString	Full class name associated with the method.
MethodName	win:UnicodeString	Name of the method.
MethodSignature	win:UnicodeString	Signature of the method (comma-separated list of type names).
ClrInstanceID	win:UInt16	Unique ID for the instance of CoreCLR.

MethodJitInliningSucceeded event

KEYWORD FOR RAISING THE EVENT	LEVEL
JITTracingKeyword (0x1000)	Verbose (5)

EVENT	EVENT ID	DESCRIPTION
MethodJitInliningSucceeded	185	Raised when a method is successfully inlined by the JIT compiler.

FIELD NAME	DATA TYPE	DESCRIPTION
MethodBeingCompiledNamespace	win:UnicodeString	Namespace of the method being compiled.
MethodBeingCompiledName	win:UnicodeString	Name of the method being compiled.
MethodBeingCompiledNameSignature	win:UnicodeString	Signature of the method (comma-separated list of type names) being compiled.
InlinerNamespace	win:UnicodeString	The namespace of inliner ("parent") method.
InlinerName	win:UnicodeString	Name of the inliner ("parent") method.

FIELD NAME	DATA TYPE	DESCRIPTION
InlinerNameSignature	win:UnicodeString	Signature of the inliner ("parent") method (comma-separated list of type names).
InlineeNamespace	win:UnicodeString	The namespace of inlinee ("child") method.
InlineeName	win:UnicodeString	Name of the inlinee ("child") method.
InlineeNameSignature	win:UnicodeString	Signature of the inlinee ("child") method (comma-separated list of type names).
ClrInstanceID	win:UInt16	Unique ID for the instance of CoreCLR.

MethodJitInliningFailed event

KEYWORD FOR RAISING THE EVENT	LEVEL
JITTracingKeyword (0x1000)	Verbose (5)

EVENT	EVENT ID	DESCRIPTION
MethodJitInliningFailed	192	Raised when a method was failed to be inlined by the JIT compiler.

FIELD NAME	DATA TYPE	DESCRIPTION
MethodBeingCompiledNamespace	win:UnicodeString	Namespace of the method being compiled.
MethodBeingCompiledName	win:UnicodeString	Name of the method being compiled.
MethodBeingCompiledNameSignature	win:UnicodeString	Signature of the method (comma-separated list of type names) being compiled.
InlineerNamespace	win:UnicodeString	The namespace of inliner ("parent") method.
InlineerName	win:UnicodeString	Name of the inliner ("parent") method.
InlineerNameSignature	win:UnicodeString	Signature of the inliner ("parent") method (comma-separated list of type names).
InlineeNamespace	win:UnicodeString	The namespace of inlinee ("child") method.
InlineeName	win:UnicodeString	Name of the inlinee ("child") method.

FIELD NAME	DATA TYPE	DESCRIPTION
InlineeNameSignature	win:UnicodeString	Signature of the inlinee ("child") method (comma-separated list of type names).
FailAlways	win:Boolean	Whether the method is marked as not inlinable.
FailReason	win:UnicodeString	Reason inlining failed.
ClrInstanceID	win:UInt16	Unique ID for the instance of CoreCLR.

MethodJitTailCallSucceeded event

KEYWORD FOR RAISING THE EVENT	LEVEL
JITTracingKeyword (0x1000)	Verbose (5)

EVENT	EVENT ID	DESCRIPTION
MethodJitTailCallSucceeded	192	Raised by the JIT compiler when a method can be successfully tail called.
FIELD NAME	DATA TYPE	DESCRIPTION
MethodBeingCompiledNamespace	win:UnicodeString	Namespace of the method being compiled.
MethodBeingCompiledName	win:UnicodeString	Name of the method being compiled.
MethodBeingCompiledNameSignature	win:UnicodeString	Signature of the method (comma-separated list of type names) being compiled.
CallerNamespace	win:UnicodeString	Namespace of the caller method.
CallerName	win:UnicodeString	Name of the caller method.
CallerNameSignature	win:UnicodeString	Signature of the caller method (Comma-separated list of type names).
CalleeNamespace	win:UnicodeString	Namespace of the callee method.
CalleeName	win:UnicodeString	Name of the callee method.
CalleeNameSignature	win:UnicodeString	Signature of the callee method (Comma-separated list of type names).
TailPrefix	win:Boolean	Whether it is a tail prefix instruction.

FIELD NAME	DATA TYPE	DESCRIPTION
TailCallType	win:UInt32	<p>The type of tail call.</p> <p>0: Optimized tail call (epilog + jmp)</p> <p>1: Recursive tail call (method tail calls itself)</p> <p>2: Helper assisted tail call</p>
ClrInstanceID	win:UInt16	Unique ID for the instance of CoreCLR.

MethodJitTailCallFailed event

KEYWORD FOR RAISING THE EVENT	LEVEL
JITTracingKeyword (0x1000)	Verbose (5)

EVENT	EVENT ID	DESCRIPTION
MethodJitTailCallFailed	191	Raised by the JIT compiler when a method was failed to be tail called.

FIELD NAME	DATA TYPE	DESCRIPTION
MethodBeingCompiledNamespace	win:UnicodeString	Namespace of the method being compiled.
MethodBeingCompiledName	win:UnicodeString	Name of the method being compiled.
MethodBeingCompiledNameSignature	win:UnicodeString	Signature of the method (comma-separated list of type names) being compiled.
CallerNamespace	win:UnicodeString	Namespace of the caller method.
CallerName	win:UnicodeString	Name of the caller method.
CallerNameSignature	win:UnicodeString	Signature of the caller method (Comma-separated list of type names).
CalleeNamespace	win:UnicodeString	Namespace of the callee method.
CalleeName	win:UnicodeString	Name of the callee method.
CalleeNameSignature	win:UnicodeString	Signature of the callee method (Comma-separated list of type names).
TailPrefix	win:Boolean	Whether it is a tail prefix instruction.
FailReason	win:UnicodeString	Reason tail call failed.

FIELD NAME	DATA TYPE	DESCRIPTION
ClrInstanceID	win:UInt16	Unique ID for the instance of CoreCLR.

MethodILToNativeMap event

KEYWORD FOR RAISING THE EVENT	LEVEL
JittedMethodILToNativeMapKeyword (0x20000)	Verbose (5)

EVENT	EVENT ID	DESCRIPTION
MethodILToNativeMap	190	Maps the IL-to-native map event for JIT-compiled methods.

FIELD NAME	DATA TYPE	DESCRIPTION
MethodID	win:UInt64	Unique identifier of a method.
ReJITID	win:UInt64	The ReJIT ID of the method.
MethodExtent	win:UInt8	The extent for the jitted method.
CountOfMapEntries	win:UInt8	Number of map entries
ILOffsets	win:UInt32	The IL offset.
NativeOffsets	win:UInt32	The native code offset.
ClrInstanceID	win:UInt16	Unique ID for the instance of CoreCLR.

.NET runtime thread pool events

9/20/2022 • 8 minutes to read • [Edit Online](#)

These events collect information about worker and I/O threads in the threadpool. For more information about how to use these events for diagnostic purposes, see [logging and tracing .NET applications](#)

IOThreadCreate_V1 event

The following table shows the keyword and level.

KEYWORD FOR RAISING THE EVENT	LEVEL
<code>ThreadingKeyword</code> (0x10000)	Informational (4)

The following table shows the event information.

EVENT	EVENT ID	RAISED WHEN
<code>IOThreadCreate_V1</code>	44	An I/O thread is created in the thread pool.

The following table shows the event data.

FIELD NAME	DATA TYPE	DESCRIPTION
<code>Count</code>	<code>win:UInt64</code>	Number of I/O threads, including the newly created thread.
<code>NumRetired</code>	<code>win:UInt64</code>	Number of retired worker threads.
<code>ClrInstanceId</code>	<code>win:UInt16</code>	Unique ID for the instance of CLR or CoreCLR.

IOThreadTerminate_V1 event

The following table shows the keyword and level

KEYWORD FOR RAISING THE EVENT	LEVEL
<code>ThreadingKeyword</code> (0x10000)	Informational (4)

The following table shows the event information.

EVENT	EVENT ID	RAISED WHEN
<code>IOThreadTerminate</code>	45	An I/O thread is terminated in the thread pool.

The following table shows the event data.

FIELD NAME	DATA TYPE	DESCRIPTION
Count	win:UInt64	Number of I/O threads remaining in the thread pool.
NumRetired	win:UInt64	Number of retired I/O threads.
ClrInstanceID	win:UInt16	Unique ID for the instance of CLR or CoreCLR.

IOThreadRetire_V1 event

The following table shows the keyword and level.

KEYWORD FOR RAISING THE EVENT	LEVEL
ThreadingKeyword (0x10000)	Informational (4)

The following table shows the event information.

EVENT	EVENT ID	RAISED WHEN
IOThreadRetire_V1	46	An I/O thread becomes a retirement candidate.

The following table shows the event data.

FIELD NAME	DATA TYPE	DESCRIPTION
Count	win:UInt64	Number of I/O threads remaining in the thread pool.
NumRetired	win:UInt64	Number of retired I/O threads.
ClrInstanceID	win:UInt16	Unique ID for the instance of CLR or CoreCLR.

IOThreadUnretire_V1 event

The following table shows the keyword and level.

KEYWORD FOR RAISING THE EVENT	LEVEL
ThreadingKeyword (0x10000)	Informational (4)

The following table shows the event information.

EVENT	EVENT ID	RAISED WHEN
IOThreadUnretire_V1	47	An I/O thread is unretired because of I/O that arrives within a waiting period after the thread becomes a retirement candidate.

The following table shows the event data.

FIELD NAME	DATA TYPE	DESCRIPTION
Count	win:UInt64	Number of I/O threads in the thread pool, including this one.
NumRetired	win:UInt64	Number of retired I/O threads.
ClrInstanceID	Win:UInt16	Unique ID for the instance of CLR or CoreCLR.

ThreadPoolWorkerThreadStart event

KEYWORD FOR RAISING THE EVENT		
<code>ThreadingKeyword</code> (0x10000)		
EVENT	EVENT ID	DESCRIPTION
ThreadPoolWorkerThreadStart	50	A worker thread is created.
FIELD NAME	DATA TYPE	DESCRIPTION
ActiveWorkerThreadCount	win:UInt32	Number of worker threads available to process work, including those that are already processing work.
RetiredWorkerThreadCount	win:UInt32	Number of worker threads that are not available to process work, but that are being held in reserve in case more threads are needed later.
ClrInstanceID	win:UInt16	Unique ID for the instance of CLR or CoreCLR.

ThreadPoolWorkerThreadStop event

KEYWORD FOR RAISING THE EVENT		
<code>ThreadingKeyword</code> (0x10000)		
EVENT	EVENT ID	DESCRIPTION
ThreadPoolWorkerThreadStop	51	A worker thread is stopped.
FIELD NAME	DATA TYPE	DESCRIPTION
ActiveWorkerThreadCount	win:UInt32	Number of worker threads available to process work, including those that are already processing work.

FIELD NAME	DATA TYPE	DESCRIPTION
RetiredWorkerThreadCount	win:UInt32	Number of worker threads that are not available to process work, but that are being held in reserve in case more threads are needed later.
ClrInstanceId	win:UInt16	Unique ID for the instance of CLR or CoreCLR.

ThreadPoolWorkerThreadWait event

KEYWORD FOR RAISING THE EVENT	LEVEL
ThreadingKeyword (0x10000)	Informational (4)

EVENT	EVENT ID	DESCRIPTION
ThreadPoolWorkerThreadWait	57	A worker thread starts waiting for work.

FIELD NAME	DATA TYPE	DESCRIPTION
ActiveWorkerThreadCount	win:UInt32	Number of worker threads available to process work, including those that are already processing work.
RetiredWorkerThreadCount	win:UInt32	Number of worker threads that are not available to process work, but that are being held in reserve in case more threads are needed later.
ClrInstanceId	win:UInt16	Unique ID for the instance of CLR or CoreCLR.

ThreadPoolWorkerThreadRetirementStart event

KEYWORD FOR RAISING THE EVENT	LEVEL
ThreadingKeyword (0x10000)	Informational (4)

EVENT	EVENT ID	DESCRIPTION
ThreadPoolWorkerThreadRetirementStart	52	A worker thread retires.

FIELD NAME	DATA TYPE	DESCRIPTION
ActiveWorkerThreadCount	win:UInt32	Number of worker threads available to process work, including those that are already processing work.

FIELD NAME	DATA TYPE	DESCRIPTION
RetiredWorkerThreadCount	win:UInt32	Number of worker threads that are not available to process work, but that are being held in reserve in case more threads are needed later.
ClrInstanceId	win:UInt16	Unique ID for the instance of CLR or CoreCLR.

ThreadPoolWorkerThreadRetirementStop event

KEYWORD FOR RAISING THE EVENT	LEVEL
ThreadingKeyword (0x10000)	Informational (4)

EVENT	EVENT ID	DESCRIPTION
ThreadPoolWorkerThreadRetirementStop	53	A retired worker thread becomes active again.

FIELD NAME	DATA TYPE	DESCRIPTION
ActiveWorkerThreadCount	win:UInt32	Number of worker threads available to process work, including those that are already processing work.
RetiredWorkerThreadCount	win:UInt32	Number of worker threads that are not available to process work, but that are being held in reserve in case more threads are needed later.
ClrInstanceId	win:UInt16	Unique ID for the instance of CLR or CoreCLR.

ThreadPoolWorkerThreadAdjustmentSample event

The following table shows the keyword and level.

KEYWORD FOR RAISING THE EVENT	LEVEL
ThreadingKeyword (0x10000)	Informational (4)

The following table shows the event information.

EVENT	EVENT ID	DESCRIPTION
ThreadPoolWorkerThreadAdjustmentSample	64	Refers to the collection of information for one sample; that is, a measurement of throughput with a certain concurrency level, in an instant of time.

The following table shows the event data.

FIELD NAME	DATA TYPE	DESCRIPTION
Throughput	win:Double	Number of completions per unit of time.
ClrInstanceID	win:UInt16	Unique ID for the instance of CLR or CoreCLR.

ThreadPoolWorkerThreadAdjustmentAdjustment event

The following table shows the keyword and level.

KEYWORD FOR RAISING THE EVENT	LEVEL
ThreadingKeyword (0x10000)	Informational (4)

The following table shows the event information.

EVENT	EVENT ID	DESCRIPTION
ThreadPoolWorkerThreadAdjustmentAdjustment	55	Records a change in control, when the thread injection (hill-climbing) algorithm determines that a change in concurrency level is in place.

The following table shows the event data.

FIELD NAME	DATA TYPE	DESCRIPTION
AverageThroughput	win:Double	Average throughput of a sample of measurements.
NewWorkerThreadCount	win:UInt32	New number of active worker threads.
Reason	win:UInt32	Reason for the adjustment. 0x0 - Warmup. 0x1 - Initializing. 0x2 - Random move. 0x3 - Climbing move. 0x4 - Change point. 0x5 - Stabilizing. 0x6 - Starvation. 0x7 - Thread timed out.
ClrInstanceID	win:UInt16	Unique ID for the instance of CLR or CoreCLR.

ThreadPoolWorkerThreadAdjustmentStats event

The following table shows the keyword and level.

KEYWORD FOR RAISING THE EVENT	LEVEL
ThreadingKeyword (0x10000)	Verbose (5)

The following table shows the event information.

EVENT	EVENT ID	DESCRIPTION
ThreadPoolWorkerThreadAdjustmentStats	56	Gathers data on the thread pool.

The following table shows the event data

FIELD NAME	DATA TYPE	DESCRIPTION
Duration	win:Double	Amount of time, in seconds, during which these statistics were collected.
Throughput	win:Double	Average number of completions per second during this interval.
ThreadWave	win:Double	Reserved for internal use.
ThroughputWave	win:Double	Reserved for internal use.
ThroughputErrorEstimate	win:Double	Reserved for internal use.
AverageThroughputErrorEstimate	win:Double	Reserved for internal use.
ThroughputRatio	win:Double	The relative improvement in throughput caused by variations in active worker thread count during this interval.
Confidence	win:Double	A measure of the validity of the ThroughputRatio field.
NewcontrolSetting	win:Double	The number of active worker threads that serve as the baseline for future variations in active thread count.
NewThreadWaveMagnitude	win:UInt16	The magnitude of future variations in active thread count.
ClrInstanceID	win:UInt16	Unique ID for the instance of CLR or CoreCLR.

ThreadPoolEnqueue event

The following table shows the keyword and level.

KEYWORD FOR RAISING THE EVENT	LEVEL
ThreadingKeyword (0x10000)	Verbose (5)

The following table shows the event information.

EVENT	EVENT ID	DESCRIPTION
ThreadPoolEnqueue	61	A work item was enqueued in the thread pool queue.

The following table shows the event data

FIELD NAME	DATA TYPE	DESCRIPTION
WorkID	win:Pointer	Pointer to the work request.
ClrInstanceId	win:UInt16	Unique ID for the instance of CoreCLR.

ThreadPoolDequeue event

The following table shows the keyword and level.

KEYWORD FOR RAISING THE EVENT	LEVEL
ThreadingKeyword (0x10000)	Verbose (5)

The following table shows the event information.

EVENT	EVENT ID	DESCRIPTION
ThreadPoolDequeue	62	A work item was dequeued from the thread pool queue.

The following table shows the event data

FIELD NAME	DATA TYPE	DESCRIPTION
WorkID	win:Pointer	Pointer to the work request.
ClrInstanceId	win:UInt16	Unique ID for the instance of CoreCLR.

ThreadPoolIOEnqueue event

The following table shows the keyword and level.

KEYWORD FOR RAISING THE EVENT	LEVEL
ThreadingKeyword (0x10000)	Verbose (5)

The following table shows the event information.

EVENT	EVENT ID	DESCRIPTION
ThreadPoolIOEnqueue	63	A thread enqueues an IO completion notification after an async IO completion occurs.

The following table shows the event data

FIELD NAME	DATA TYPE	DESCRIPTION
NativeOverlapped	win:Pointer	Reserved for internal use.
Overlapped	win:Pointer	Reserved for internal use.
MultiDequeues	win:Boolean	Reserved for internal use.
ClrInstanceID	win:UInt16	Unique ID for the instance of CoreCLR.

ThreadPoolIODequeue event

The following table shows the keyword and level.

KEYWORD FOR RAISING THE EVENT	LEVEL
ThreadingKeyword (0x10000)	Verbose (5)

The following table shows the event information.

EVENT	EVENT ID	DESCRIPTION
ThreadPoolIODequeue	64	A thread dequeues the IO completion notification.

The following table shows the event data

FIELD NAME	DATA TYPE	DESCRIPTION
NativeOverlapped	win:Pointer	Reserved for internal use.
Overlapped	win:Pointer	Reserved for internal use.
MultiDequeues	win:Boolean	Reserved for internal use.
ClrInstanceID	win:UInt16	Unique ID for the instance of CoreCLR.

ThreadPoolIOPack event

The following table shows the keyword and level.

KEYWORD FOR RAISING THE EVENT	LEVEL
ThreadingKeyword (0x10000)	Verbose (5)

The following table shows the event information.

EVENT	EVENT ID	DESCRIPTION
ThreadPoolIOPack	65	ThreadPool overlapped IO pack is called.

The following table shows the event data

FIELD NAME	DATA TYPE	DESCRIPTION
NativeOverlapped	win:Pointer	Reserved for internal use.
Overlapped	win:Pointer	Reserved for internal use.
ClrInstanceId	win:UInt16	Unique ID for the instance of CoreCLR.

ThreadCreating event

The following table shows the keywords and level.

KEYWORD FOR RAISING THE EVENT	LEVEL
ThreadingKeyword (0x10000)	Informational (4)

The following table shows the event information.

EVENT	EVENT ID	DESCRIPTION
ThreadCreating	70	Thread has been created.

The following table shows the event data.

FIELD NAME	DATA TYPE	DESCRIPTION
ID	win:Pointer	Thread ID
ClrInstanceId	win:UInt16	Unique ID for the instance of CoreCLR.

ThreadRunning event

The following table shows the keywords and level.

KEYWORD FOR RAISING THE EVENT	LEVEL
ThreadingKeyword (0x10000)	Informational (4)

The following table shows the event information.

EVENT	EVENT ID	DESCRIPTION
ThreadRunning	71	Thread has started running.

The following table shows the event data.

FIELD NAME	DATA TYPE	DESCRIPTION
ID	win:Pointer	Thread ID
ClrInstanceID	win:UInt16	Unique ID for the instance of CoreCLR.

.NET runtime type events

9/20/2022 • 2 minutes to read • [Edit Online](#)

These events collect information relating to loading types. For more information about how to use these events for diagnostic purposes, see [logging and tracing .NET applications](#)

TypeLoadStart Event

KEYWORD FOR RAISING THE EVENT	EVENT	LEVEL
TypeDiagnosticKeyword (0x8000000000)	Informational (4)	
EVENT	EVENT ID	DESCRIPTION
TypeLoadStart	73	A type load has started.
FIELD NAME	DATA TYPE	DESCRIPTION
TypeLoadStartID	win:UInt32	ID for the type load operation.
ClrInstanceID	win:UInt16	Unique ID for the instance of CLR or CoreCLR.

TypeLoadStop Event

KEYWORD FOR RAISING THE EVENT	LEVEL	
TypeDiagnosticKeyword (0x8000000000)	Informational (4)	
EVENT	EVENT ID	DESCRIPTION
TypeLoadStop	74	A type load is finished.
FIELD NAME	DATA TYPE	DESCRIPTION
TypeLoadStartID	win:UInt32	ID for the type load operation (matches the corresponding TypeLoadStart event's TypeLoadStartID).
LoadLevel	win:UInt16	Type load level.
TypeID	win:UInt64	Pointer to the type handle.
TypeName	win:UnicodeString	Name of the type.

FIELD NAME	DATA TYPE	DESCRIPTION
ClrInstanceID	win:UInt16	Unique ID for the instance of CLR or CoreCLR.

Collect diagnostics in containers

9/20/2022 • 2 minutes to read • [Edit Online](#)

The same diagnostics tools that are useful for diagnosing .NET Core issues in other scenarios also work in Docker containers. However, some of the tools require special steps to work in a container. This article covers how tools for gathering performance traces and collecting dumps can be used in Docker containers.

Using .NET CLI tools in a container

This tools apply to: ✓ .NET Core 3.1 SDK and later versions

The .NET Core global CLI diagnostic tools ([dotnet-counters](#), [dotnet-dump](#), [dotnet-gcdump](#), [dotnet-monitor](#), and [dotnet-trace](#)) are designed to work in a wide variety of environments and should all work directly in Docker containers. Because of this, these tools are the preferred method of collecting diagnostic information for .NET Core scenarios targeting .NET Core 3.1 or later in containers.

You can also install these tools without the .NET SDK by downloading the single-file variants from the links in the previous paragraph. These installs require a global install of the .NET runtime version 3.1 or later, which you can acquire following any of the prescribed methods in the [.NET installation documentation](#) or by consuming any of the official runtime containers.

Using .NET Core global CLI tools in a sidecar container

If you would like to use .NET Core global CLI diagnostic tools to diagnose processes in a different container, bear the following additional requirements in mind:

1. The containers must [share a process namespace](#) (so that tools in the sidecar container can access processes in the target container).
2. The .NET Core global CLI diagnostic tools need access to files the .NET Core runtime writes to the /tmp directory, so the /tmp directory must be shared between the target and sidecar container via a volume mount. This could be done, for example, by having the containers share a common [volume](#) or a Kubernetes [emptyDir](#) volume. If you attempt to use the diagnostic tools from a sidecar container without sharing the /tmp directory, you will get an error about the process "not running compatible .NET runtime."

Using `PerfCollect` in a container

This tool applies to: ✓ .NET Core 2.1 and later versions

The `PerfCollect` script is useful for collecting performance traces and is the recommended tool for collecting traces prior to .NET Core 3.0. If using `PerfCollect` in a container, keep the following requirements in mind:

- `PerfCollect` requires the [SYS_ADMIN capability](#) (in order to run the `perf` tool), so be sure the container is [started with that capability](#).
- `PerfCollect` requires some environment variables be set prior to the app it is profiling starting. These can be set either in a [Dockerfile](#) or when [starting the container](#). Because these variables shouldn't be set in normal production environments, it's common to just add them when starting a container that will be profiled. The two variables that `PerfCollect` requires are:
 - `DOTNET_PerfMapEnabled=1`
 - `DOTNET_EnableEventLog=1`

NOTE

.NET 6 standardizes on the prefix `DOTNET_` instead of `COMPlus_` for environment variables that configure .NET run-time behavior. However, the `COMPlus_` prefix will continue to work. If you're using a previous version of the .NET runtime, you should still use the `COMPlus_` prefix for environment variables.

Using `PerfCollect` in a sidecar container

If you would like to run `PerfCollect` in one container to profile a .NET Core process in a different container, the experience is almost the same except for these differences:

- The environment variables mentioned previously (`DOTNET_PerfMapEnabled` and `DOTNET_EnableEventLog`) must be set for the target container (not the one running `PerfCollect`).
- The container running `PerfCollect` must have the `SYS_ADMIN` capability (not the target container).
- The two containers must [share a process namespace](#).

Investigate performance counters (dotnet-counters)

9/20/2022 • 13 minutes to read • [Edit Online](#)

This article applies to: ✓ `dotnet-counters` version 3.0.47001 and later versions

Install

There are two ways to download and install `dotnet-counters`:

- **dotnet global tool:**

To install the latest release version of the `dotnet-counters` NuGet package, use the `dotnet tool install` command:

```
dotnet tool install --global dotnet-counters
```

- **Direct download:**

Download the tool executable that matches your platform:

OS	Platform
Windows	x86 x64 Arm Arm-x64
macOS	x64
Linux	x64 Arm Arm64 musl-x64 musl-Arm64

NOTE

To use `dotnet-counters` on an x86 app, you need a corresponding x86 version of the tool.

Synopsis

```
dotnet-counters [-h|--help] [--version] <command>
```

Description

`dotnet-counters` is a performance monitoring tool for ad-hoc health monitoring and first-level performance investigation. It can observe performance counter values that are published via the [EventCounter API](#) or the [Meter API](#). For example, you can quickly monitor things like the CPU usage or the rate of exceptions being thrown in your .NET Core application to see if there's anything suspicious before diving into more serious performance investigation using `PerfView` or `dotnet-trace`.

Options

- `--version`

Displays the version of the dotnet-counters utility.

- `-h|--help`

Shows command-line help.

Commands

COMMAND

```
dotnet-counters collect  
dotnet-counters list  
dotnet-counters monitor  
dotnet-counters ps
```

dotnet-counters collect

Periodically collect selected counter values and export them into a specified file format for post-processing.

Synopsis

```
dotnet-counters collect [-h|--help] [-p|--process-id] [-n|--name] [--diagnostic-port] [--refresh-interval]  
[--counters <COUNTERS>] [--format] [-o|--output] [-- <command>]
```

Options

- `-p|--process-id <PID>`

The ID of the process to collect counter data from.

- `-n|--name <name>`

The name of the process to collect counter data from.

- `--diagnostic-port`

The name of the diagnostic port to create. See [using diagnostic port](#) for how to use this option to start monitoring counters from app startup.

- `--refresh-interval <SECONDS>`

The number of seconds to delay between updating the displayed counters

- `--counters <COUNTERS>`

A comma-separated list of counters. Counters can be specified `[provider_name[:counter_name]]`. If the `provider_name` is used without a qualifying list of counters, then all counters from the provider are shown. To discover provider and counter names, use the [dotnet-counters list](#) command. For [EventCounters](#), `provider_name` is the name of the EventSource and for [Meters](#), `provider_name` is the name of the Meter.

- `--format <csv|json>`

The format to be exported. Currently available: csv, json.

- `-o|--output <output>`

The name of the output file.

- `-- <command>` (for target applications running .NET 5 or later only)

After the collection configuration parameters, the user can append `--` followed by a command to start a .NET application with at least a 5.0 runtime. `dotnet-counters` will launch a process with the provided command and collect the requested metrics. This is often useful to collect metrics for the application's startup path and can be used to diagnose or monitor issues that happen early before or shortly after the main entrypoint.

NOTE

Using this option monitors the first .NET 5 process that communicates back to the tool, which means if your command launches multiple .NET applications, it will only collect the first app. Therefore, it is recommended you use this option on self-contained applications, or using the `dotnet exec <app.dll>` option.

NOTE

Launching a .NET executable via `dotnet-counters` will make its input/output to be redirected and you won't be able to interact with its stdin/stdout. Exiting the tool via CTRL+C or SIGTERM will safely end both the tool and the child process. If the child process exits before the tool, the tool will exit as well and the trace should be safely viewable. If you need to use stdin/stdout, you can use the `--diagnostic-port` option. See [Using diagnostic port](#) for more information.

NOTE

On Linux and macOS, this command expects the target application and `dotnet-counters` to share the same `TMPDIR` environment variable. Otherwise, the command will time out.

NOTE

To collect metrics using `dotnet-counters`, it needs to be run as the same user as the user running target process or as root. Otherwise, the tool will fail to establish a connection with the target process.

Examples

- Collect all counters at a refresh interval of 3 seconds and generate a csv as output:

```
> dotnet-counters collect --process-id 1902 --refresh-interval 3 --format csv  
counter_list is unspecified. Monitoring all counters by default.  
Starting a counter session. Press Q to quit.
```

- Start `dotnet mvc.dll` as a child process and start collecting runtime counters and ASP.NET Core Hosting counters from startup and save it as a JSON output:

```
> dotnet-counters collect --format json --counters System.Runtime,Microsoft.AspNetCore.Hosting --  
dotnet mvc.dll  
Starting a counter session. Press Q to quit.  
File saved to counter.json
```

dotnet-counters list

Displays a list of counter names and descriptions, grouped by provider.

Synopsis

```
dotnet-counters list [-h|--help]
```

Example

```
> dotnet-counters list
Showing well-known counters only. Specific processes may support additional counters.

System.Runtime
  cpu-usage                                Amount of time the process has utilized the CPU (ms)
  working-set                               Amount of working set used by the process (MB)
  gc-heap-size                             Total heap size reported by the GC (MB)
  gen-0-gc-count                          Number of Gen 0 GCs per interval
  gen-1-gc-count                          Number of Gen 1 GCs per interval
  gen-2-gc-count                          Number of Gen 2 GCs per interval
  time-in-gc                               % time in GC since the last GC
  gen-0-size                               Gen 0 Heap Size
  gen-1-size                               Gen 1 Heap Size
  gen-2-size                               Gen 2 Heap Size
  loh-size                                 LOH Heap Size
  alloc-rate                               Allocation Rate
  assembly-count                          Number of Assemblies Loaded
  exception-count                         Number of Exceptions per interval
  threadpool-thread-count                 Number of ThreadPool Threads
  monitor-lock-contention-count          Monitor Lock Contention Count
  threadpool-queue-length                ThreadPool Work Items Queue Length
  threadpool-completed-items-count       ThreadPool Completed Work Items Count
  active-timer-count                      Active Timers Count

Microsoft.AspNetCore.Hosting
  requests-per-second                     Request rate
  total-requests                          Total number of requests
  current-requests                        Current number of requests
  failed-requests                         Failed number of requests
```

NOTE

The `Microsoft.AspNetCore.Hosting` counters are displayed when there are processes identified that support these counters, for example, when an ASP.NET Core application is running on the host machine.

dotnet-counters monitor

Displays periodically refreshing values of selected counters.

Synopsis

```
dotnet-counters monitor [-h|--help] [-p|--process-id] [-n|--name] [--diagnostic-port] [--refresh-interval]
[--counters] [-- <command>]
```

Options

- `-p|--process-id <PID>`

The ID of the process to be monitored.

- `-n|--name <name>`

The name of the process to be monitored.

- `--diagnostic-port`

The name of the diagnostic port to create. See [using diagnostic port](#) for how to use this option to start monitoring counters from app startup.

- `--refresh-interval <SECONDS>`

The number of seconds to delay between updating the displayed counters

- `--counters <COUNTERS>`

A comma-separated list of counters. Counters can be specified `[provider_name[:counter_name]]`. If the `provider_name` is used without a qualifying list of counters, then all counters from the provider are shown. To discover provider and counter names, use the [dotnet-counters list](#) command. For `EventCounters`, `provider_name` is the name of the EventSource and for `Meters`, `provider_name` is the name of the Meter.

`-- <command>` (for target applications running .NET 5 or later only)

After the collection configuration parameters, the user can append `--` followed by a command to start a .NET application with at least a 5.0 runtime. `dotnet-counters` will launch a process with the provided command and monitor the requested metrics. This is often useful to collect metrics for the application's startup path and can be used to diagnose or monitor issues that happen early before or shortly after the main entrypoint.

NOTE

Using this option monitors the first .NET 5 process that communicates back to the tool, which means if your command launches multiple .NET applications, it will only collect the first app. Therefore, it is recommended you use this option on self-contained applications, or using the `dotnet exec <app.dll>` option.

NOTE

Launching a .NET executable via `dotnet-counters` will make its input/output to be redirected and you won't be able to interact with its `stdin/stdout`. Exiting the tool via `CTRL+C` or `SIGTERM` will safely end both the tool and the child process. If the child process exits before the tool, the tool will exit as well. If you need to use `stdin/stdout`, you can use the `--diagnostic-port` option. See [Using diagnostic port](#) for more information.

NOTE

On Linux and macOS, this command expects the target application and `dotnet-counters` to share the same `TMPDIR` environment variable.

NOTE

To monitor metrics using `dotnet-counters`, it needs to be run as the same user as the user running target process or as root.

NOTE

If you see an error message similar to the following one:

```
[ERROR] System.ComponentModel.Win32Exception (299): A 32 bit processes cannot access modules of a 64 bit process.
```

, you are trying to use `dotnet-counters` that has mismatched bitness against the target process. Make sure to download the correct bitness of the tool in the [install](#) link.

Examples

- Monitor all counters from `System.Runtime` at a refresh interval of 3 seconds:

```
> dotnet-counters monitor --process-id 1902 --refresh-interval 3 --counters System.Runtime
Press p to pause, r to resume, q to quit.
Status: Running

[System.Runtime]
% Time in GC since last GC (%) 0
Allocation Rate (B / 1 sec) 5,376
CPU Usage (%) 0
Exception Count (Count / 1 sec) 0
GC Fragmentation (%) 48.467
GC Heap Size (MB) 0
Gen 0 GC Count (Count / 1 sec) 1
Gen 0 Size (B) 24
Gen 1 GC Count (Count / 1 sec) 1
Gen 1 Size (B) 24
Gen 2 GC Count (Count / 1 sec) 1
Gen 2 Size (B) 272,000
IL Bytes Jitted (B) 19,449
LOH Size (B) 19,640
Monitor Lock Contention Count (Count / 1 sec) 0
Number of Active Timers 0
Number of Assemblies Loaded 7
Number of Methods Jitted 166
POH (Pinned Object Heap) Size (B) 24
ThreadPool Completed Work Item Count (Count / 1 sec) 0
ThreadPool Queue Length 0
ThreadPool Thread Count 2
Working Set (MB) 19
```

- Monitor just CPU usage and GC heap size from `System.Runtime`:

```
> dotnet-counters monitor --process-id 1902 --counters System.Runtime[cpu-usage,gc-heap-size]
Press p to pause, r to resume, q to quit.
Status: Running

[System.Runtime]
CPU Usage (%) 24
GC Heap Size (MB) 811
```

- Monitor `EventCounter` values from user-defined `EventSource`. For more information, see [Tutorial: Measure performance using EventCounters in .NET Core](#).

```
> dotnet-counters monitor --process-id 1902 --counters Samples-EventCounterDemos-Minimal
Press p to pause, r to resume, q to quit.
request 100
```

- View all well-known counters that are available in `dotnet-counters` :

```
> dotnet-counters list

Showing well-known counters for .NET (Core) version 3.1 only. Specific processes may support
additional counters.

System.Runtime
  cpu-usage                               The percent of process' CPU usage relative to all of the
  system CPU resources [0-100]
    working-set                            Amount of working set used by the process (MB)
    gc-heap-size                           Total heap size reported by the GC (MB)
    gen-0-gc-count                         Number of Gen 0 GCs between update intervals
    gen-1-gc-count                         Number of Gen 1 GCs between update intervals
    gen-2-gc-count                         Number of Gen 2 GCs between update intervals
    time-in-gc                             % time in GC since the last GC
    gen-0-size                             Gen 0 Heap Size
    gen-1-size                             Gen 1 Heap Size
    gen-2-size                             Gen 2 Heap Size
    loh-size                                LOH Size
    alloc-rate                             Number of bytes allocated in the managed heap between update
intervals
    assembly-count                          Number of Assemblies Loaded
    exception-count                        Number of Exceptions / sec
    threadpool-thread-count                Number of ThreadPool Threads
    monitor-lock-contention-count         Number of times there were contention when trying to take the
monitor lock between update intervals
    threadpool-queue-length                ThreadPool Work Items Queue Length
    threadpool-completed-items-count      ThreadPool Completed Work Items Count
    active-timer-count                    Number of timers that are currently active

Microsoft.AspNetCore.Hosting
  requests-per-second                     Number of requests between update intervals
  total-requests                          Total number of requests
  current-requests                        Current number of requests
  failed-requests                         Failed number of requests
```

- View all well-known counters that are available in `dotnet-counters` for .NET 5 apps:

```
> dotnet-counters list --runtime-version 5.0

Showing well-known counters for .NET (Core) version 5.0 only. Specific processes may support
additional counters.

System.Runtime
  cpu-usage                               The percent of process' CPU usage relative to all of the
  system CPU resources [0-100]
    working-set                            Amount of working set used by the process (MB)
    gc-heap-size                           Total heap size reported by the GC (MB)
    gen-0-gc-count                         Number of Gen 0 GCs between update intervals
    gen-1-gc-count                         Number of Gen 1 GCs between update intervals
    gen-2-gc-count                         Number of Gen 2 GCs between update intervals
    time-in-gc                             % time in GC since the last GC
    gen-0-size                            Gen 0 Heap Size
    gen-1-size                            Gen 1 Heap Size
    gen-2-size                            Gen 2 Heap Size
    loh-size                               LOH Size
    poh-size                               POH (Pinned Object Heap) Size
    alloc-rate                            Number of bytes allocated in the managed heap between update
    intervals
    gc-fragmentation                      GC Heap Fragmentation
    assembly-count                         Number of Assemblies Loaded
    exception-count                        Number of Exceptions / sec
    threadpool-thread-count                Number of ThreadPool Threads
    monitor-lock-contention-count          Number of times there were contention when trying to take the
    monitor lock between update intervals
    threadpool-queue-length                ThreadPool Work Items Queue Length
    threadpool-completed-items-count      ThreadPool Completed Work Items Count
    active-timer-count                     Number of timers that are currently active
    il-bytes-jitted                       Total IL bytes jitted
    methods-jitted-count                  Number of methods jitted

Microsoft.AspNetCore.Hosting
  requests-per-second                     Number of requests between update intervals
  total-requests                          Total number of requests
  current-requests                        Current number of requests
  failed-requests                         Failed number of requests

Microsoft.AspNetCore.Server.Kestrel
  connections-per-second                  Number of connections between update intervals
  total-connections                       Total Connections
  tls-handshakes-per-second               Number of TLS Handshakes made between update intervals
  total-tls-handshakes                   Total number of TLS handshakes made
  current-tls-handshakes                 Number of currently active TLS handshakes
  failed-tls-handshakes                  Total number of failed TLS handshakes
  current-connections                    Number of current connections
  connection-queue-length                Length of Kestrel Connection Queue
  request-queue-length                  Length total HTTP request queue

System.Net.Http
  requests-started                        Total Requests Started
  requests-started-rate                  Number of Requests Started between update intervals
  requests-aborted                        Total Requests Aborted
  requests-aborted-rate                  Number of Requests Aborted between update intervals
  current-requests                       Current Requests
```

- Launch `my-aspnet-server.exe` and monitor the # of assemblies loaded from its startup (.NET 5 or later only):

IMPORTANT

This works for apps running .NET 5 or later only.

```
> dotnet-counters monitor --counters System.Runtime[assembly-count] -- my-aspnet-server.exe

Press p to pause, r to resume, q to quit.
Status: Running

[System.Runtime]
Number of Assemblies Loaded 24
```

- Launch `my-aspnet-server.exe` with `arg1` and `arg2` as command-line arguments and monitor its working set and GC heap size from its startup (.NET 5 or later only):

IMPORTANT

This works for apps running .NET 5 or later only.

```
> dotnet-counters monitor --counters System.Runtime[working-set,gc-heap-size] -- my-aspnet-server.exe
arg1 arg2
```

```
Press p to pause, r to resume, q to quit.
Status: Running
```

[System.Runtime]	
GC Heap Size (MB)	39
Working Set (MB)	59

dotnet-counters ps

Lists the dotnet processes that can be monitored by `dotnet-counters`. `dotnet-counters` version 6.0.320703 and later, also display the command-line arguments that each process was started with, if available.

Synopsis

```
dotnet-counters ps [-h|--help]
```

Example

Suppose you start a long-running app using the command `dotnet run --configuration Release`. In another window, you run the `dotnet-counters ps` command. The output you'll see is as follows. The command-line arguments, if any, are shown in `dotnet-counters` version 6.0.320703 and later.

```
> dotnet-counters ps

21932 dotnet      C:\Program Files\dotnet\dotnet.exe    run --configuration Release
36656 dotnet      C:\Program Files\dotnet\dotnet.exe
```

Using diagnostic port

IMPORTANT

This works for apps running .NET 5 or later only.

[Diagnostic port](#) is a runtime feature added in .NET 5 that allows you to start monitoring or collecting counters

from app startup. To do this using `dotnet-counters`, you can either use `dotnet-counters <collect|monitor> -- <command>` as described in the examples above, or use the `--diagnostic-port` option.

Using `dotnet-counters <collect|monitor> -- <command>` to launch the application as a child process is the simplest way to quickly monitor it from its startup.

However, when you want to gain a finer control over the lifetime of the app being monitored (for example, monitor the app for the first 10 minutes only and continue executing) or if you need to interact with the app using the CLI, using `--diagnostic-port` option allows you to control both the target app being monitored and `dotnet-counters`.

1. The command below makes `dotnet-counters` create a diagnostics socket named `myport.sock` and wait for a connection.

```
dotnet-counters collect --diagnostic-port myport.sock
```

Output:

```
Waiting for connection on myport.sock
Start an application with the following environment variable:
DOTNET_DiagnosticPorts=/home/user/myport.sock
```

2. In a separate console, launch the target application with the environment variable `DOTNET_DiagnosticPorts` set to the value in the `dotnet-counters` output.

```
export DOTNET_DiagnosticPorts=/home/user/myport.sock
./my-dotnet-app arg1 arg2
```

This should then enable `dotnet-counters` to start collecting counters on `my-dotnet-app`:

```
Waiting for connection on myport.sock
Start an application with the following environment variable: DOTNET_DiagnosticPorts=myport.sock
Starting a counter session. Press Q to quit.
```

IMPORTANT

Launching your app with `dotnet run` can be problematic because the dotnet CLI may spawn many child processes that are not your app and they can connect to `dotnet-counters` before your app, leaving your app to be suspended at run time. It is recommended you directly use a self-contained version of the app or use `dotnet exec` to launch the application.

dotnet-coverage code coverage utility

9/20/2022 • 9 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 3.1 SDK and later versions

Synopsis

```
dotnet-coverage [-h, --help] [--version] <command>
```

Description

The `dotnet-coverage` tool:

- Enables the collection of code coverage data of a running process on Windows (x86, x64 and Arm64), Linux (x64) and macOS (x64).
- Provides cross-platform merging of code coverage reports.

Options

- `-h|--help`

Shows command-line help.

- `--version`

Displays the version of the `dotnet-coverage` utility.

Install

To install the latest release version of the `dotnet-coverage` NuGet package, use the `dotnet tool install` command:

```
dotnet tool install --global dotnet-coverage
```

Commands

COMMAND

[dotnet-coverage merge](#)

[dotnet-coverage collect](#)

[dotnet-coverage connect](#)

[dotnet-coverage snapshot](#)

[dotnet-coverage shutdown](#)

dotnet-coverage merge

The `merge` command is used to merge several code coverage reports into one. This command is available on all platforms. This command supports the following code coverage report formats:

- `coverage`
- `cobertura`
- `xml`

Synopsis

```
dotnet-coverage merge  
  [--remove-input-files] [-r|--recursive]  
  [-o|--output <output>] [-f|--output-format <output-format>]  
  [-l|--log-file <log-file>] [-ll|--log-level <log-level>] [-?|-h|--help]  
  <files>
```

Arguments

- `<files>`

The input code coverage reports.

Options

- `--remove-input-files`

Removes all input coverage reports that were merged.

- `-r, --recursive`

Search for coverage reports in subdirectories.

- `-o|--output <output>`

Sets the code coverage report output file.

- `-f|--output-format <output-format>`

The output file format. Supported values: `coverage`, `xml`, and `cobertura`. Default is `coverage` (binary format that can be opened in Visual Studio).

- `-l|--log-file <log-file>`

Sets the log file path. When you provide a directory (with a path separator at the end), a new log file is generated for each process under analysis.

- `-ll|--log-level <log-level>`

Sets the log level. Supported values: `Error`, `Info`, and `Verbose`.

dotnet-coverage collect

The `collect` command is used to collect code coverage data for any .NET process and its subprocesses. For example, you can collect code coverage data for a console application or a Blazor application. This command is available on Windows (x86, x64 and Arm64), Linux (x64), and macOS (x64). The command supports only .NET modules. Native modules are not supported.

Synopsis

The `collect` command can run in two modes.

Command Mode

The `collect` command will collect code coverage for the given process executed by the `command` argument.

```
dotnet-coverage collect  
[-s|--settings <settings>] [-id|--session-id <session-id>]  
[-o|--output <output>] [-f|--output-format <output-format>]  
[-l|--log-file <log-file>] [-ll|--log-level <log-level>] [-?|-h|--help]  
<command> <args>
```

Server Mode

The `collect` command hosts a server for code coverage collection. Clients can connect to the server via `connect` command.

```
dotnet-coverage collect  
[-s|--settings <settings>] [-id|--session-id <session-id>]  
[-sv|--server-mode] [-b|--background] [-t|--timeout]  
[-o|--output <output>] [-f|--output-format <output-format>]  
[-l|--log-file <log-file>] [-ll|--log-level <log-level>] [-?|-h|--help]
```

Arguments

- `<command>`

The command for which to collect code coverage data.

- `<args>`

The command line arguments for the command.

Options

- `-s|--settings <settings>`

Sets the path to the XML code coverage settings.

- `-id|--session-id <session-id>`

Specifies the code coverage session ID. If not provided, the tool will generate a random GUID.

- `-sv|--server-mode`

Starts the collector in server mode. Clients can connect to the server with the `connect` command.

- `-b|--background`

Starts code coverage collection server in a new background process. Clients can connect to the server with the `connect` command.

- `-t|--timeout`

Timeout (in milliseconds) for interprocess communication between clients and the server.

- `-o|--output <output>`

Sets the code coverage report output file.

- `-f|--output-format <output-format>`

The output file format. Supported values: `coverage`, `xml`, and `cobertura`. Default is `coverage` (binary format that can be opened in Visual Studio).

- `-l|--log-file <log-file>`

Sets the log file path. When you provide a directory (with a path separator at the end), a new log file is generated for each process under analysis.

- `-l1|--log-level <log-level>`

Sets the log level. Supported values: `Error`, `Info`, and `Verbose`.

dotnet-coverage connect

The `connect` command is used to connect with the existing server and collects code coverage data for any .NET process and its subprocesses. For example, you can collect code coverage data for a console application or a Blazor application. This command is available on Windows (x86, x64 and Arm64), Linux (x64), and macOS (x64). The command supports only .NET modules. Native modules are not supported.

Synopsis

```
dotnet-coverage connect
[-b|--background] [-t|--timeout]
[-l1|--log-file <log-file>] [-l1|--log-level <log-level>] [-?|-h|--help]
<session>
<command> <args>
```

Arguments

- `<session>`

The session ID of the server hosted by the `collect` command.

- `<command>`

The command for which to collect code coverage data.

- `<args>`

The command line arguments for the command.

Options

- `-b|--background`

Starts the client in a new background process.

- `-t|--timeout`

Timeout (in milliseconds) for interprocess communication between the client and the server.*

```
-l1|--log-file <log-file>
```

Sets the log file path. When you provide a directory (with a path separator at the end), a new log file is generated for each process under analysis.

- `-l1|--log-level <log-level>`

Sets the log level. Supported values: `Error`, `Info`, and `Verbose`.

dotnet-coverage snapshot

Creates a coverage file for existing code coverage collection.

Synopsis

```
dotnet-coverage snapshot
[-r|--reset] [-o|--output <output>] [-t|--timeout]
[-l1|--log-file <log-file>] [-l1|--log-level <log-level>] [-?|-h|--help]
<session>
```

Arguments

- `<session>`

The session ID of the collection for which a coverage file is to be generated.

Options

- `-r|--reset <reset>`

Clears existing coverage information after a coverage file is created.

- `-o|--output <output>`

Sets the code coverage report output file. If not provided, it's generated automatically with a timestamp.

- `-t|--timeout`

Timeout (in milliseconds) for interprocess communication between the client and the server.

- `-l|--log-file <log-file>`

Sets the log file path. When you provide a directory (with a path separator at the end), a new log file is generated for each process under analysis.

- `-ll|--log-level <log-level>`

Sets the log level. Supported values: `Error`, `Info`, and `Verbose`.

dotnet-coverage shutdown

Closes existing code coverage collection.

Synopsis

```
dotnet-coverage shutdown
[-t|--timeout]
[-l|--log-file <log-file>] [-ll|--log-level <log-level>] [-?|-h|--help]
<session>
```

Arguments

- `<session>`

The session ID of the collection to be closed.

Options

- `-t|--timeout`

Timeout (in milliseconds) for interprocess communication with the server.

- `-l|--log-file <log-file>`

Sets the log file path. When you provide a directory (with a path separator at the end), a new log file is generated for each process under analysis.

- `-ll|--log-level <log-level>`

Sets the log level. Supported values: `Error`, `Info`, and `Verbose`.

Sample scenarios

Collecting code coverage

Collect code coverage data for any .NET application (such as console or Blazor) by using the following command:

```
dotnet-coverage collect dotnet run
```

In case of an application that requires a signal to terminate, you can use **Ctrl+C**, which will still let you collect code coverage data. For the argument, you can provide any command that will eventually start a .NET app. For example, it can be a PowerShell script.

Sessions

When you're running code coverage analysis on a .NET server that just waits for messages and sends responses, you need a way to stop the server to get final code coverage results. You can use **Ctrl+C** locally, but not in Azure Pipelines. For these scenarios, you can use sessions. You can specify a session ID when starting collection, and then use the `shutdown` command to stop collection and the server.

For example, assume you have a server in the `D:\serverexample\server` directory and a test project in the `D:\serverexample\tests` directory. Tests are communicating with the server through the network. You can start code coverage collection for the server as follows:

```
D:\serverexample\server> dotnet-coverage collect --session-id serverdemo "dotnet run"
```

Session ID was specified as `serverdemo`. Then you can run tests as follows:

```
D:\serverexample\tests> dotnet test
```

A code coverage file for session `serverdemo` can be generated with current coverage as follows:

```
dotnet-coverage snapshot --output after_first_test.coverage serverdemo
```

Finally, session `serverdemo` and the server can be closed as follows:

```
dotnet-coverage shutdown serverdemo
```

Following is an example of full output on the server side:

```
D:\serverexample\server> dotnet-coverage collect --session-id serverdemo "dotnet run"
SessionId: serverdemo
Waiting for a connection... Connected!
Received: Hello!
Sent: HELLO!
Waiting for a connection... Code coverage results: output.coverage.
D:\serverexample\server>
```

Server and client mode

Code coverage collection can be done in server-client mode as well. In this scenario, a code coverage collection server starts, and multiple clients can connect with the server. Code coverage is collected for all the clients collectively.

Start the code coverage server using the following command:

```
dotnet-coverage collect --session-id serverdemo --server-mode
```

In this example, the session ID was specified as `serverdemo` for the server. A client can connect to the server using this session ID using the following command:

```
dotnet-coverage connect serverdemo dotnet run
```

Finally, you can close the session `serverdemo` and the server using the following command:

```
dotnet-coverage shutdown serverdemo
```

The server process creates a collective code coverage report for all clients and exits.

Following is an example of full output on the server side:

```
D:\serverexample\server> dotnet-coverage collect --session-id serverdemo --server-mode
SessionId: serverdemo
// Server will be in idle state and wait for connect and shutdown commands
Code coverage results: output.coverage.
D:\serverexample\server>
```

Following is an example of full output on the client side:

```
D:\serverexample\server> dotnet-coverage connect serverdemo ConsoleApplication.exe World
Hello World!
D:\serverexample\server> dotnet-coverage connect serverdemo WpfApplication.exe
D:\serverexample\server> dotnet-coverage shutdown serverdemo
D:\serverexample\server>
```

You can also start both server and client in background mode. Another process starts in the background and returns control back to the user.

Following is an example of full output in background server client mode:

```
D:\serverexample\server> dotnet-coverage collect --session-id serverdemo --server-mode --background
D:\serverexample\server> dotnet-coverage connect --background serverdemo ConsoleApplication.exe World
D:\serverexample\server> dotnet-coverage connect --background serverdemo WpfApplication.exe
D:\serverexample\server> dotnet-coverage shutdown serverdemo
D:\serverexample\server>
```

Settings

You can specify a file with settings when you use the `collect` command. The settings file can be used to exclude some modules or methods from code coverage analysis. The format is the same as the data collector configuration inside a *runsettings* file. For more information, see [Customize code coverage analysis](#). Here's an example:

```
<?xml version="1.0" encoding="utf-8"?>
<Configuration>
  <CodeCoverage>
    <!--
        Additional paths to search for .pdb (symbol) files. Symbols must be found for modules to be
        instrumented.
        If .pdb files are in the same folder as the .dll or .exe files, they are automatically found.
        Otherwise, specify them here.
        Note that searching for symbols increases code coverage run time. So keep this small and local
    </CodeCoverage>
  </Configuration>
</?xml>
```

```

NOTE THAT SEARCHING FOR SYMBOLS IN RELEASE CODE COVERAGE TAKES TIME. SO KEEP THIS SMALL AND LOCAL.
-->
<SymbolSearchPaths>
    <Path>C:\Users\User\Documents\Visual Studio 2012\Projects\ProjectX\bin\Debug</Path>
    <Path>..\mybuildshare\builds\ProjectX</Path>
</SymbolSearchPaths>

<!--
About include/exclude lists:
Empty "Include" clauses imply all; empty "Exclude" clauses imply none.
Each element in the list is a regular expression (ECMAScript syntax). See /visualstudio/ide/using-regular-expressions-in-visual-studio.
An item must first match at least one entry in the include list to be included.
Included items must then not match any entries in the exclude list to remain included.
-->

<!-- Match assembly file paths: -->
<ModulePaths>
    <Include>
        <ModulePath>.*\.dll$</ModulePath>
        <ModulePath>.*\.exe$</ModulePath>
    </Include>
    <Exclude>
        <ModulePath>.*CPPUnitTestFramework.*</ModulePath>
    </Exclude>
</ModulePaths>

<!-- Match fully qualified names of functions: -->
<!-- (Use "\." to delimit namespaces in C# or Visual Basic, ":" in C++) -->
<Functions>
    <Exclude>
        <Function>^Fabrikam\\UnitTest\..*</Function>
        <Function>^std::.*</Function>
        <Function>^ATL::.*</Function>
        <Function>.*::__GetTestMethodInfo.*</Function>
        <Function>^Microsoft::VisualStudio::CppCodeCoverageFramework::.*</Function>
        <Function>^Microsoft::VisualStudio::CppUnitTestFramework::.*</Function>
    </Exclude>
</Functions>

<!-- Match attributes on any code element: -->
<Attributes>
    <Exclude>
        <!-- Don't forget "Attribute" at the end of the name -->
        <Attribute>^System\.\Diagnostics\.\DebuggerHiddenAttribute$</Attribute>
        <Attribute>^System\.\Diagnostics\.\DebuggerNonUserCodeAttribute$</Attribute>
        <Attribute>^System\.\CodeDom\.\Compiler\.\GeneratedCodeAttribute$</Attribute>
        <Attribute>^System\.\Diagnostics\.\CodeAnalysis\.\ExcludeFromCodeCoverageAttribute$</Attribute>
    </Exclude>
</Attributes>

<!-- Match the path of the source files in which each method is defined: -->
<Sources>
    <Exclude>
        <Source>.*\\atlmfc\\.*</Source>
        <Source>.*\\vctools\\.*</Source>
        <Source>.*\\public\\sdk\\.*</Source>
        <Source>.*\\microsoft sdks\\.*</Source>
        <Source>.*\\vc\\include\\.*</Source>
    </Exclude>
</Sources>

<!-- Match the company name property in the assembly: -->
<CompanyNames>
    <Exclude>
        <CompanyName>.*microsoft.*</CompanyName>
    </Exclude>
</CompanyNames>

```

```
<!-- Match the public key token of a signed assembly: -->
<PublicKeyTokens>
    <!-- Exclude Visual Studio extensions: -->
    <Exclude>
        <PublicKeyToken>^B77A5C561934E089$</PublicKeyToken>
        <PublicKeyToken>^B03F5F7F11D50A3A$</PublicKeyToken>
        <PublicKeyToken>^31BF3856AD364E35$</PublicKeyToken>
        <PublicKeyToken>^89845DCD8080CC91$</PublicKeyToken>
        <PublicKeyToken>^71E9BCE111E9429C$</PublicKeyToken>
        <PublicKeyToken>^8F50407C4E9E73B6$</PublicKeyToken>
        <PublicKeyToken>^E361AF139669C375$</PublicKeyToken>
    </Exclude>
</PublicKeyTokens>

</CodeCoverage>
</Configuration>
```

Merge code coverage reports

You can merge `a.coverage` and `b.coverage` and store the data in `merged.coverage` as follows:

```
dotnet-coverage merge -o merged.coverage a.coverage b.coverage
```

For example, if you run a command like `dotnet test --collect "Code Coverage"`, the coverage report is stored into a folder that is named a random GUID. Such folders are hard to find and merge. Using this tool, you can merge all code coverage reports for all your projects as follows:

```
dotnet-coverage merge -o merged.cobertura.xml -f cobertura -r *.coverage
```

The preceding command merges all coverage reports from the current directory and all subdirectories and stores the result into a cobertura file. In Azure Pipelines, you can use [Publish Code Coverage Results task](#) to publish a merged cobertura report.

You can use the `merge` command to convert a code coverage report to another format. For example, the following command converts a binary code coverage report into XML format.

```
dotnet-coverage merge -o output.xml -f xml input.coverage
```

See also

- [Customize code coverage analysis](#)
- [Publish Code Coverage Results task](#)

Dump collection and analysis utility (dotnet-dump)

9/20/2022 • 6 minutes to read • [Edit Online](#)

This article applies to: ✓ `dotnet-dump` version 3.0.47001 and later versions

NOTE

`dotnet-dump` for macOS is only supported with .NET 5 and later versions.

Install

There are two ways to download and install `dotnet-dump`:

- **dotnet global tool:**

To install the latest release version of the `dotnet-dump` NuGet package, use the `dotnet tool install` command:

```
dotnet tool install --global dotnet-dump
```

- **Direct download:**

Download the tool executable that matches your platform:

OS	Platform
Windows	x86 x64 Arm Arm-x64
macOS	x64
Linux	x64 Arm Arm64 musl-x64 musl-Arm64

NOTE

To use `dotnet-dump` on an x86 app, you need a corresponding x86 version of the tool.

Synopsis

```
dotnet-dump [-h|--help] [--version] <command>
```

Description

The `dotnet-dump` global tool is a way to collect and analyze dumps on Windows, Linux, and macOS without any native debugger involved. This tool is important on platforms like Alpine Linux where a fully working `lldb` isn't available. The `dotnet-dump` tool allows you to run SOS commands to analyze crashes and the garbage collector (GC), but it isn't a native debugger so things like displaying native stack frames aren't supported.

Options

- `--version`

Displays the version of the dotnet-dump utility.

- `-h|--help`

Shows command-line help.

Commands

COMMAND

`dotnet-dump collect`

`dotnet-dump analyze`

`dotnet-dump ps`

dotnet-dump collect

Captures a dump from a process.

Synopsis

```
dotnet-dump collect [-h|--help] [-p|--process-id] [-n|--name] [--type] [-o|--output] [--diag]
```

Options

- `-h|--help`

Shows command-line help.

- `-p|--process-id <PID>`

Specifies the process ID number to collect a dump from.

- `-n|--name <name>`

Specifies the name of the process to collect a dump from.

- `--type <Full|Heap|Mini>`

Specifies the dump type, which determines the kinds of information that are collected from the process.

There are three types:

- `Full` - The largest dump containing all memory including the module images.
- `Heap` - A large and relatively comprehensive dump containing module lists, thread lists, all stacks, exception information, handle information, and all memory except for mapped images.
- `Mini` - A small dump containing module lists, thread lists, exception information, and all stacks.

If not specified, `Full` is the default.

- `-o|--output <output_dump_path>`

The full path and file name where the collected dump should be written.

If not specified:

- Defaults to `.\dump_YYYYMMDD_HHMMSS.dmp` on Windows.
 - Defaults to `./core_YYYYMMDD_HHMMSS` on Linux and macOS.
- YYYYMMDD is Year/Month/Day and HHMMSS is Hour/Minute/Second.

- `--diag`

Enables dump collection diagnostic logging.

- `--crashreport`

Enables crash report generation.

NOTE

On Linux and macOS, this command expects the target application and `dotnet-dump` to share the same `TMPDIR` environment variable. Otherwise, the command will time out.

NOTE

To collect a dump using `dotnet-dump`, it needs to be run as the same user as the user running target process or as root. Otherwise, the tool will fail to establish a connection with the target process.

dotnet-dump analyze

Starts an interactive shell to explore a dump. The shell accepts various [SOS commands](#).

Synopsis

```
dotnet-dump analyze <dump_path> [-h|--help] [-c|--command]
```

Arguments

- `<dump_path>`

Specifies the path to the dump file to analyze.

Options

- `-c|--command <debug_command>`

Specifies the `command` to run in the shell on start.

Analyze SOS commands

COMMAND	FUNCTION
<code>soshelp</code> or <code>help</code>	Displays all available commands
<code>soshelp <command></code> or <code>help <command></code>	Displays the specified command.
<code>exit</code> or <code>quit</code>	Exits interactive mode.
<code>clrstack <arguments></code>	Provides a stack trace of managed code only.
<code>clrthreads <arguments></code>	Lists the managed threads running.

COMMAND	FUNCTION
<code>dumpasync <arguments></code>	Displays information about async state machines on the garbage-collected heap.
<code>dumpassembly <arguments></code>	Displays details about the assembly at the specified address.
<code>dumpclass <arguments></code>	Displays information about the <code>EEClass</code> structure at the specified address.
<code>dumpdelegate <arguments></code>	Displays information about the delegate at the specified address.
<code>dumpdomain <arguments></code>	Displays information all the AppDomains and all assemblies within the specified domain.
<code>dumpheap <arguments></code>	Displays info about the garbage-collected heap and collection statistics about objects.
<code>dumpil <arguments></code>	Displays the Microsoft intermediate language (MSIL) that is associated with a managed method.
<code>dumplog <arguments></code>	Writes the contents of an in-memory stress log to the specified file.
<code>dumpmd <arguments></code>	Displays information about the <code>MethodDesc</code> structure at the specified address.
<code>dumpmodule <arguments></code>	Displays information about the module at the specified address.
<code>dumpmt <arguments></code>	Displays information about the <code>MethodTable</code> at the specified address.
<code>dumpobj <arguments></code>	Displays info about the object at the specified address.
<code>dso <arguments></code> or <code>dumpstackobjects <arguments></code>	Displays all managed objects found within the bounds of the current stack.
<code>eeheap <arguments></code>	Displays info about process memory consumed by internal runtime data structures.
<code>finalizequeue <arguments></code>	Displays all objects registered for finalization.
<code>gcroot <arguments></code>	Displays info about references (or roots) to the object at the specified address.
<code>gcwhere <arguments></code>	Displays the location in the GC heap of the argument passed in.
<code>ip2md <arguments></code>	Displays the <code>MethodDesc</code> structure at the specified address in JIT code.

COMMAND	FUNCTION
<code>histclear <arguments></code>	Releases any resources used by the family of <code>hist*</code> commands.
<code>histinit <arguments></code>	Initializes the SOS structures from the stress log saved in the debugger.
<code>histobj <arguments></code>	Displays the garbage collection stress log relocations related to <code><arguments></code> .
<code>histobjfind <arguments></code>	Displays all the log entries that reference the object at the specified address.
<code>histroot <arguments></code>	Displays information related to both promotions and relocations of the specified root.
<code>lm</code> or <code>modules</code>	Displays the native modules in the process.
<code>name2ee <arguments></code>	Displays the <code>MethodTable</code> and <code>EEClass</code> structures for the <code><argument></code> .
<code>pe <arguments></code> or <code>printexception <arguments></code>	Displays any object derived from the <code>Exception</code> class for the <code><argument></code> .
<code>setsymbolserver <arguments></code>	Enables the symbol server support
<code>syncblk <arguments></code>	Displays the SyncBlock holder info.
<code>threads <threadid></code> or <code>setthread <threadid></code>	Sets or displays the current thread ID for the SOS commands.

NOTE

Additional details can be found in [SOS Debugging Extension for .NET](#).

dotnet-dump ps

Lists the dotnet processes that dumps can be collected from. `dotnet-dump` version 6.0.320703 and later versions also display the command-line arguments that each process was started with, if available.

Synopsis

```
dotnet-dump ps [-h|--help]
```

Example

Suppose you start a long-running app using the command `dotnet run --configuration Release`. In another window, you run the `dotnet-dump ps` command. The output you'll see is as follows. The command-line arguments, if any, are shown in `dotnet-dump` version 6.0.320703 and later.

```
> dotnet-dump ps

21932 dotnet      C:\Program Files\dotnet\dotnet.exe    run --configuration Release
36656 dotnet      C:\Program Files\dotnet\dotnet.exe
```

Using `dotnet-dump`

The first step is to collect a dump. This step can be skipped if a core dump has already been generated. The operating system or the .NET Core runtime's built-in [dump generation feature](#) can each create core dumps.

```
$ dotnet-dump collect --process-id 1902
Writing minidump to file ./core_20190226_135837
Written 98983936 bytes (24166 pages) to core file
Complete
```

Now analyze the core dump with the `analyze` command:

```
$ dotnet-dump analyze ./core_20190226_135850
Loading core dump: ./core_20190226_135850
Ready to process analysis commands. Type 'help' to list available commands or 'help [command]' to get
detailed help on a command.
Type 'quit' or 'exit' to exit the session.
>
```

This action brings up an interactive session that accepts commands like:

```
> clrstack
OS Thread Id: 0x573d (0)
    Child SP          IP Call Site
00007FFD28B42C58 00007fb22c1a8ed9 [HelperMethodFrame_PROTECTOBJ: 00007ffd28b42c58]
System.RuntimeMethodHandle.InvokeMethod(System.Object, System.Object[], System.Signature, Boolean, Boolean)
00007FFD28B42DD0 00007FB1B1334F67 System.Reflection.RuntimeMethodInfo.Invoke(System.Object,
System.Reflection.BindingFlags, System.Reflection.Binder, System.Object[], System.Globalization.CultureInfo)
[/root/coreclr/src/mscorlib/src/System/Reflection/RuntimeMethodInfo.cs @ 472]
00007FFD28B42E20 00007FB1B18D33ED SymbolTestApp.Program.Foo4(System.String)
[/home/mikem/builds/SymbolTestApp/SymbolTestApp/SymbolTestApp.cs @ 54]
00007FFD28B42ED0 00007FB1B18D2FC4 SymbolTestApp.Program.Foo2(Int32, System.String)
[/home/mikem/builds/SymbolTestApp/SymbolTestApp/SymbolTestApp.cs @ 29]
00007FFD28B42F00 00007FB1B18D2F5A SymbolTestApp.Program.Foo1(Int32, System.String)
[/home/mikem/builds/SymbolTestApp/SymbolTestApp/SymbolTestApp.cs @ 24]
00007FFD28B42F30 00007FB1B18D168E SymbolTestApp.Program.Main(System.String[])
[/home/mikem/builds/SymbolTestApp/SymbolTestApp/SymbolTestApp.cs @ 19]
00007FFD28B43210 00007fb22aa9cedf [GCFrame: 00007ffd28b43210]
00007FFD28B43610 00007fb22aa9cedf [GCFrame: 00007ffd28b43610]
```

To see an unhandled exception that killed your app:

```

> pe -lines
Exception object: 00007fb18c038590
Exception type: System.Reflection.TargetInvocationException
Message: Exception has been thrown by the target of an invocation.
InnerException: System.Exception, Use !PrintException 00007FB18C038368 to see more.
StackTrace (generated):
SP          IP          Function
00007FFD28B42DD0 0000000000000000
System.Private.CoreLib.dll!System.RuntimeMethodHandle.InvokeMethod(System.Object, System.Object[], System.Signature, Boolean, Boolean)
00007FFD28B42DD0 00007FB1B1334F67
System.Private.CoreLib.dll!System.Reflection.RuntimeMethodInfo.Invoke(System.Object, System.Reflection.BindingFlags, System.Reflection.Binder, System.Object[], System.Globalization.CultureInfo)+0xa7
[/root/coreclr/src/mscorlib/src/System/Reflection/RuntimeMethodInfo.cs @ 472]
00007FFD28B42E20 00007FB1B18D33ED SymbolTestApp.dll!SymbolTestApp.Program.Foo4(System.String)+0x15d
[/home/mikem/builds/SymbolTestApp/SymbolTestApp/SymbolTestApp.cs @ 54]
00007FFD28B42ED0 00007FB1B18D2FC4 SymbolTestApp.dll!SymbolTestApp.Program.Foo2(Int32, System.String)+0x34
[/home/mikem/builds/SymbolTestApp/SymbolTestApp/SymbolTestApp.cs @ 29]
00007FFD28B42F00 00007FB1B18D2F5A SymbolTestApp.dll!SymbolTestApp.Program.Foo1(Int32, System.String)+0x3a
[/home/mikem/builds/SymbolTestApp/SymbolTestApp/SymbolTestApp.cs @ 24]
00007FFD28B42F30 00007FB1B18D168E SymbolTestApp.dll!SymbolTestApp.Program.Main(System.String[])+0x6e
[/home/mikem/builds/SymbolTestApp/SymbolTestApp/SymbolTestApp.cs @ 19]

StackTraceString: <none>
HResult: 80131604

```

Special instructions for Docker

If you're running under Docker, dump collection requires `SYS_PTRACE` capabilities (`--cap-add=SYS_PTRACE` or `--privileged`).

On Microsoft .NET SDK Linux Docker images, some `dotnet-dump` commands can throw the following exception:

Unhandled exception: System.DllNotFoundException: Unable to load shared library 'libdl.so' or one of its dependencies' exception.

To work around this problem, install the "libc6-dev" package.

See also

- [Collecting and analyzing memory dumps blog](#)
- [Heap analysis tool \(dotnet-gcdump\)](#)

Heap analysis tool (dotnet-gcdump)

9/20/2022 • 4 minutes to read • [Edit Online](#)

This article applies to: ✓ `dotnet-gcdump` version 3.1.57502 and later versions

Install

There are two ways to download and install `dotnet-gcdump`:

- **dotnet global tool:**

To install the latest release version of the `dotnet-gcdump` NuGet package, use the [dotnet tool install](#) command:

```
dotnet tool install --global dotnet-gcdump
```

- **Direct download:**

Download the tool executable that matches your platform:

OS	Platform
Windows	x86 x64 Arm Arm-x64
macOS	x64
Linux	x64 Arm Arm64 musl-x64 musl-Arm64

NOTE

To use `dotnet-gcdump` on an x86 app, you need a corresponding x86 version of the tool.

Synopsis

```
dotnet-gcdump [-h|--help] [--version] <command>
```

Description

The `dotnet-gcdump` global tool collects GC (Garbage Collector) dumps of live .NET processes using [EventPipe](#). GC dumps are created by triggering a GC in the target process, turning on special events, and regenerating the graph of object roots from the event stream. This process allows for GC dumps to be collected while the process is running and with minimal overhead. These dumps are useful for several scenarios:

- Comparing the number of objects on the heap at several points in time.
- Analyzing roots of objects (answering questions like, "what still has a reference to this type?").
- Collecting general statistics about the counts of objects on the heap.

[View the GC dump captured from dotnet-gcdump](#)

On Windows, `.gcdump` files can be viewed in [PerfView](#) for analysis or in Visual Studio. Currently, there is no way of opening a `.gcdump` on non-Windows platforms.

You can collect multiple `.gcdump`s and open them simultaneously in Visual Studio to get a comparison experience.

Options

- `--version`

Displays the version of the `dotnet-gcdump` utility.

- `-h|--help`

Shows command-line help.

Commands

COMMAND

`dotnet-gcdump collect`

`dotnet-gcdump ps`

`dotnet-gcdump report`

`dotnet-gcdump collect`

Collects a GC dump from a currently running process.

WARNING

To walk the GC heap, this command triggers a generation 2 (full) garbage collection, which can suspend the runtime for a long time, especially when the GC heap is large. Don't use this command in performance-sensitive environments when the GC heap is large.

Synopsis

```
dotnet-gcdump collect [-h|--help] [-p|--process-id <pid>] [-o|--output <gcdump-file-path>] [-v|--verbose] [-t|--timeout <timeout>] [-n|--name <name>]
```

Options

- `-h|--help`

Shows command-line help.

- `-p|--process-id <pid>`

The process ID to collect the GC dump from.

- `-o|--output <gcdump-file-path>`

The path where collected GC dumps should be written. Defaults to
`.\YYYYMMDD_HHMMSS_<pid>.gcdump`.

- `-v|--verbose`

Output the log while collecting the GC dump.

- `-t|--timeout <timeout>`

Give up on collecting the GC dump if it takes longer than this many seconds. The default value is 30.

- `-n|--name <name>`

The name of the process to collect the GC dump from.

NOTE

On Linux and macOS, this command expects the target application and `dotnet-gcdump` to share the same `TMPDIR` environment variable. Otherwise, the command will time out.

NOTE

To collect a GC dump using `dotnet-gcdump`, it needs to be run as the same user as the user running target process or as root. Otherwise, the tool will fail to establish a connection with the target process.

`dotnet-gcdump ps`

Lists the dotnet processes that GC dumps can be collected for. `dotnet-gcdump` 6.0.320703 and later, also display the command-line arguments that each process was started with, if available.

Synopsis

```
dotnet-gcdump ps [-h|--help]
```

Example

Suppose you start a long-running app using the command `dotnet run --configuration Release`. In another window, you run the `dotnet-gcdump ps` command. The output you'll see is as follows. The command-line arguments, if any, are shown using `dotnet-gcdump` version 6.0.320703 and later.

```
> dotnet-gcdump ps

 21932 dotnet      C:\Program Files\dotnet\dotnet.exe      run --configuration Release
 36656 dotnet      C:\Program Files\dotnet\dotnet.exe
```

`dotnet-gcdump report <gcdump_filename>`

Generate a report from a previously generated GC dump or from a running process, and write to `stdout`.

Synopsis

```
dotnet-gcdump report [-h|--help] [-p|--process-id <pid>] [-t|--report-type <HeapStat>]
```

Options

- `-h|--help`

Shows command-line help.

- `-p|--process-id <pid>`

The process ID to collect the GC dump from.

- `-t|--report-type <HeapStat>`

The type of report to generate. Available options: heapstat (default).

Troubleshoot

- There is no type information in the gcdump.

Prior to .NET Core 3.1, there was an issue where a type cache was not cleared between gcdumps when they were invoked with EventPipe. This resulted in the events needed for determining type information not being sent for the second and subsequent gcdumps. This was fixed in .NET Core 3.1-preview2.

- COM and static types aren't in the GC dump.

Prior to .NET Core 3.1, there was an issue where static and COM types weren't sent when the GC dump was invoked via EventPipe. This has been fixed in .NET Core 3.1.

- `dotnet-gcdump` is unable to generate a `.gcdump` file due to missing information, for example, [Error] **Exception during gcdump: System.ApplicationException: ETL file shows the start of a heap dump but not its completion..** Or, the `.gcdump` file doesn't include the entire heap.

`dotnet-gcdump` works by collecting a trace of events emitted by the garbage collector during an induced generation 2 collection. If the heap is sufficiently large, or there isn't enough memory to scale the eventing buffers, then the events required to reconstruct the heap graph from the trace may be dropped. In this case, to diagnose issues with the heap, it's recommended to collect a dump of the process.

- `dotnet-gcdump` appears to cause an Out Of Memory issue in a memory constrained environment.

`dotnet-gcdump` works by collecting a trace of events emitted by the garbage collector during an induced generation 2 collection. The buffer for event collection is owned by the target application and can grow up to 256 MB. `dotnet-gcdump` itself also uses memory. If your environment is memory constrained, be sure to account for these factors when collecting a gcdump to prevent errors.

Diagnostic monitoring and collection utility (dotnet-monitor)

9/20/2022 • 3 minutes to read • [Edit Online](#)

This article applies to: ✓ `dotnet-monitor` version 6.0.0 and later versions

Install

There are two ways to download `dotnet-monitor`:

- **dotnet global tool:**

To install the latest release version of the `dotnet-monitor` NuGet package, use the [dotnet tool install](#) command:

```
dotnet tool install --global dotnet-monitor
```

- **Docker image:**

Download a Docker image for use in multi-container environments:

```
docker pull mcr.microsoft.com/dotnet/monitor
```

Synopsis

```
dotnet-monitor [-h|--help] [--version] <command>
```

Description

The `dotnet-monitor` global tool is a way to monitor .NET applications in production environments and to collect diagnostic artifacts (for example, dumps, traces, logs, and metrics) on-demand or using automated rules for collecting under specified conditions.

Options

- `--version`

Displays the version of the `dotnet-monitor` utility.

- `-h|--help`

Shows command-line help.

Commands

`COMMAND`

```
dotnet monitor collect
```

COMMAND

```
dotnet monitor config show
```

```
dotnet monitor generatekey
```

dotnet-monitor collect

Monitor .NET applications, allow collecting diagnostic artifacts, and send the results to a chosen destination.

Synopsis

```
dotnet-monitor collect [-h|--help] [-u|--urls] [-m|--metrics] [--metricUrls] [--diagnostic-port] [--no-auth]  
[--temp-apikey] [--no-http-egress]
```

Options

- `-h|--help`

Shows command-line help.

- `-u|--urls <urls>`

Bindings for the HTTP api. Default is `https://localhost:52323`.

- `-m|--metrics [true|false]`

Enable publishing of metrics to `/metrics` route. Default is `true`

- `--metricUrls <urls>`

Bindings for the metrics HTTP api. Default is `http://localhost:52325`.

- `--diagnostic-port <path>`

The fully qualified path and filename of the diagnostic port to which runtime instances can connect.

Specifying this option places `dotnet-monitor` into 'listen' mode. When not specified, `dotnet-monitor` is in 'connect' mode.

On Windows, this must be a valid named pipe name. On Linux and macOS, this must be a valid Unix Domain Socket path.

- `--no-auth`

Disables API key authentication. Default is `false`.

It is strongly recommended that this option is not used in production environments.

- `--temp-apikey`

Generates a temporary API key for the `dotnet-monitor` instance.

- `--no-http-egress`

Disables egress of diagnostic artifacts via the HTTP response. When specified, artifacts must be egressed using an egress provider.

dotnet-monitor config show

Shows configuration, as if `dotnet-monitor collect` was executed with these parameters.

Synopsis

```
dotnet-monitor config show [-h|--help] [-u|--urls] [-m|--metrics] [--metricUrls] [--diagnostic-port] [--no-auth] [--temp-apikey] [--no-http-egress] [--level] [--show-sources]
```

Options

- `-h|--help`

Shows command-line help.

- `-u|--urls <urls>`

Bindings for the HTTP api. Default is `https://localhost:52323`.

This value is mapped into configuration as the `urls` key.

- `-m|--metrics [true|false]`

Enable publishing of metrics to `/metrics` route. Default is `true`.

This value is mapped into configuration as the `Metrics:Enabled` key.

- `--metricUrls <urls>`

Bindings for the metrics HTTP api. Default is `http://localhost:52325`.

This value is mapped into configuration as the `Metrics:Endpoints` key.

- `--diagnostic-port <path>`

The fully qualified path and filename of the diagnostic port to which runtime instances can connect.

Specifying this option places `dotnet-monitor` into 'listen' mode. When not specified, `dotnet-monitor` is in 'connect' mode.

On Windows, this must be a valid named pipe name. On Linux and macOS, this must be a valid Unix Domain Socket path.

This value is mapped into configuration as the `DiagnosticPort:EndpointName` key.

- `--no-auth`

Disables API key authentication. Default is `false`.

It is strongly recommended that this option is not used in production environments.

This value is not mapped into configuration.

- `--temp-apikey`

Generates a temporary API key for the `dotnet-monitor` instance.

This value is mapped into configuration as the `Authentication:MonitorApiKey` key.

- `--no-http-egress`

Disables egress of diagnostic artifacts via the HTTP response. When specified, artifacts must be egressed using an egress provider.

This value is not mapped into configuration.

- `--level`

Configuration level. `Full` configuration can show sensitive information. There are two levels:

- `Full` - The full configuration without any redaction of any values.
- `Redacted` - The full configuration but sensitive information, such as known secrets, is redacted.

- `--show-sources`

Identifies from which configuration source each effective configuration value is provided.

dotnet-monitor generatekey

Generate an API key and hash for HTTP authentication.

Synopsis

```
dotnet-monitor generatekey [-h|--help] [-o|--output]
```

Options

- `-h|--help`

Shows command-line help.

- `-o|--output <Cmd|Json|MachineJson|PowerShell|Shell|Text>`

The output format in which the API key information is written to standard output.

The allowable values are:

- `Cmd` - Outputs in a format usable in Windows Command Prompt or batch files.
- `Json` - Outputs in a format of a JSON object.
- `MachineJson` - Outputs in a format of a JSON object without comments and explanation. Useful for automation scenarios.
- `PowerShell` - Outputs in a format usable in PowerShell prompts and scripts.
- `Shell` - Outputs in a format usable in Linux shells such as Bash.
- `Text` - Outputs in a format that is plain text.

See Also

- [dotnet/dotnet-monitor](#)

dotnet-trace performance analysis utility

9/20/2022 • 12 minutes to read • [Edit Online](#)

This article applies to: ✓ `dotnet-trace` 3.0.47001 and later versions

Install

There are two ways to download and install `dotnet-trace`:

- **dotnet global tool:**

To install the latest release version of the `dotnet-trace` NuGet package, use the `dotnet tool install` command:

```
dotnet tool install --global dotnet-trace
```

- **Direct download:**

Download the tool executable that matches your platform:

OS	Platform
Windows	x86 x64 Arm Arm-x64
macOS	x64
Linux	x64 Arm Arm64 musl-x64 musl-Arm64

NOTE

To use `dotnet-trace` on an x86 app, you need a corresponding x86 version of the tool.

Synopsis

```
dotnet-trace [-h, --help] [--version] <command>
```

Description

The `dotnet-trace` tool:

- Is a cross-platform .NET Core tool.
- Enables the collection of .NET Core traces of a running process without a native profiler.
- Is built on `EventPipe` of the .NET Core runtime.
- Delivers the same experience on Windows, Linux, or macOS.

Options

- `-h|--help`

Shows command-line help.

- `--version`

Displays the version of the dotnet-trace utility.

Commands

COMMAND

[dotnet-trace collect](#)

[dotnet-trace convert](#)

[dotnet-trace ps](#)

[dotnet-trace list-profiles](#)

[dotnet-trace report](#)

dotnet-trace collect

Collects a diagnostic trace from a running process or launches a child process and traces it (.NET 5+ only). To have the tool run a child process and trace it from its startup, append `--` to the collect command.

Synopsis

```
dotnet-trace collect [--buffersize <size>] [--clreventlevel <clreventlevel>] [--clrevents <clrevents>]
    [--format <Chromium|NetTrace|Speedscope>] [-h|--help]
    [-n, --name <name>] [--diagnostic-port] [-o|--output <trace-file-path>] [-p|--process-id <pid>]
    [--profile <profile-name>] [--providers <list-of-comma-separated-providers>]
    [--show-child-io]
    [-- <command>] (for target applications running .NET 5 or later)
```

Options

- `--buffersize <size>`

Sets the size of the in-memory buffer, in megabytes. Default 256 MB.

NOTE

If the target process emits events faster than they can be written to disk, this buffer may overflow and some events will be dropped. You can mitigate this problem by increasing the buffer size or reducing the number of events being recorded.

- `--clreventlevel <clreventlevel>`

Verbosity of CLR events to be emitted.

- `--clrevents <clrevents>`

A list of CLR runtime provider keywords to enable separated by `+` signs. This is a simple mapping that lets you specify event keywords via string aliases rather than their hex values. For example,

```
dotnet-trace collect --providers Microsoft-Windows-DotNETRuntime:3:4
```

 requests the same set of events as

```
dotnet-trace collect --clrevents gc+gchandle --clreventlevel informational
```

. The table below shows the

list of available keywords:

KEYWORD STRING ALIAS	KEYWORD HEX VALUE
gc	0x1
gchandle	0x2
fusion	0x4
loader	0x8
jit	0x10
ngen	0x20
startenumeration	0x40
endenumeration	0x80
security	0x400
appdomainresourcemanagement	0x800
jittracing	0x1000
interop	0x2000
contention	0x4000
exception	0x8000
threading	0x10000
jittedmethoddiltnativemap	0x20000
overrideandsuppressngenevents	0x40000
type	0x80000
gcheapdump	0x100000
gcsampledobjectallocationhigh	0x200000
gcheapsurvivalandmovement	0x400000
gcheapcollect	0x800000
gcheapandtypenames	0x1000000
gcsampledobjectallocationlow	0x2000000

KEYWORD STRING ALIAS	KEYWORD HEX VALUE
perftrack	0x20000000
stack	0x40000000
threadtransfer	0x80000000
debugger	0x100000000
monitoring	0x200000000
codesymbols	0x400000000
eventsource	0x800000000
compilation	0x1000000000
compilationdiagnostic	0x2000000000
methoddiagnostic	0x4000000000
typediagnostic	0x8000000000

You can read about the CLR provider more in detail on the [.NET runtime provider reference documentation](#).

- `--format {Chromium|NetTrace|Speedscope}`

Sets the output format for the trace file conversion. The default is `NetTrace`.

- `-n, --name <name>`

The name of the process to collect the trace from.

- `--diagnostic-port <path-to-port>`

The name of the diagnostic port to create. See [Use diagnostic port to collect a trace from app startup](#) to learn how to use this option to collect a trace from app startup.

- `-o|--output <trace-file-path>`

The output path for the collected trace data. If not specified, it defaults to `trace.nettrace`.

- `-p|--process-id <PID>`

The process ID to collect the trace from.

- `--profile <profile-name>`

A named pre-defined set of provider configurations that allows common tracing scenarios to be specified succinctly. The following profiles are available:

PROFILE	DESCRIPTION
---------	-------------

PROFILE	DESCRIPTION
cpu-sampling	Useful for tracking CPU usage and general .NET runtime information. This is the default option if no profile or providers are specified.
gc-verbose	Tracks GC collections and samples object allocations.
gc-collect	Tracks GC collections only at very low overhead.

- `--providers <list-of-comma-separated-providers>`

A comma-separated list of `EventPipe` providers to be enabled. These providers supplement any providers implied by `--profile <profile-name>`. If there's any inconsistency for a particular provider, this configuration takes precedence over the implicit configuration from the profile.

This list of providers is in the form:

- `Provider[,Provider]`
- `Provider` is in the form: `KnownProviderName[:Flags[:Level][:KeyValueArgs]]`.
- `KeyValueArgs` is in the form: `[key1=value1][;key2=value2]`.

To learn more about some of the well-known providers in .NET, refer to [Well-known Event Providers](#).

- `-- <command>` (for target applications running .NET 5 only)

After the collection configuration parameters, the user can append `--` followed by a command to start a .NET application with at least a 5.0 runtime. This may be helpful when diagnosing issues that happen early in the process, such as startup performance issue or assembly loader and binder errors.

NOTE

Using this option monitors the first .NET 5 process that communicates back to the tool, which means if your command launches multiple .NET applications, it will only collect the first app. Therefore, it is recommended you use this option on self-contained applications, or using the `dotnet exec <app.dll>` option.

- `--show-child-io`

Shows the input and output streams of a launched child process in the current console.

NOTE

- Stopping the trace may take a long time (up to minutes) for large applications. The runtime needs to send over the type cache for all managed code that was captured in the trace.
- On Linux and macOS, this command expects the target application and `dotnet-trace` to share the same `TMPDIR` environment variable. Otherwise, the command will time out.
- To collect a trace using `dotnet-trace`, it needs to be run as the same user as the user running the target process or as root. Otherwise, the tool will fail to establish a connection with the target process.
- If you see an error message similar to:

[ERROR] System.ComponentModel.Win32Exception (299): A 32 bit processes cannot access modules of a 64 bit process.

, you are trying to use a version of `dotnet-trace` that has mismatched bitness against the target process.

Make sure to download the correct bitness of the tool in the [install](#) link.

- If you experience an unhandled exception while running `dotnet-trace collect`, this results in an incomplete trace. If finding the root cause of the exception is your priority, navigate to [Collect dumps on crash](#). As a result of the unhandled exception, the trace is truncated when the runtime shuts down to prevent other undesired behavior such as a hang or data corruption. Even though the trace is incomplete, you can still open it to see what happened leading up to the failure. However, it will be missing Rundown information (this happens at the end of a trace) so stacks might be unresolved (depending on what providers were turned on). Open the trace by executing PerfView with the `/ContinueOnError` flag at the command line. The logs will also contain the location the exception was fired.

dotnet-trace convert

Converts `nettrace` traces to alternate formats for use with alternate trace analysis tools.

Synopsis

```
dotnet-trace convert [<input-filename>] [--format <Chromium|NetTrace|Speedscope>] [-h|--help] [-o|--output <output-filename>]
```

Arguments

- `<input-filename>`

Input trace file to be converted. Defaults to `trace.nettrace`.

Options

- `--format <Chromium|NetTrace|Speedscope>`

Sets the output format for the trace file conversion.

- `-o|--output <output-filename>`

Output filename. Extension of target format will be added.

NOTE

Converting `nettrace` files to `chromium` or `speedscope` files is irreversible. `speedscope` and `chromium` files don't have all the information necessary to reconstruct `nettrace` files. However, the `convert` command preserves the original `nettrace` file, so don't delete this file if you plan to open it in the future.

dotnet-trace ps

Lists the dotnet processes that traces can be collected from. `dotnet-trace` 6.0.320703 and later, also display the command-line arguments that each process was started with, if available.

Synopsis

```
dotnet-trace ps [-h|--help]
```

Example

Suppose you start a long-running app using the command `dotnet run --configuration Release`. In another window, you run the `dotnet-trace ps` command. The output you'll see is as follows. The command-line arguments, if available, are shown in `dotnet-trace` version 6.0.320703 and later.

```
> dotnet-trace ps

21932 dotnet      C:\Program Files\dotnet\dotnet.exe    run --configuration Release
36656 dotnet      C:\Program Files\dotnet\dotnet.exe
```

dotnet-trace list-profiles

Lists pre-built tracing profiles with a description of what providers and filters are in each profile.

Synopsis

```
dotnet-trace list-profiles [-h|--help]
```

dotnet-trace report

Creates a report into stdout from a previously generated trace.

Synopsis

```
dotnet-trace report [-h|--help] <tracefile> [command]
```

Arguments

- `<tracefile>`

The file path for the trace being analyzed.

Commands

dotnet-trace report topN

Finds the top N methods that have been on the callstack the longest.

Synopsis

```
dotnet-trace report <tracefile> topN [-n|--number <n>] [--inclusive] [-v|--verbose] [-h|--help]
```

Options

- `-n|--number <n>`

Gives the top N methods on the callstack.

- `--inclusive`

Output the top N methods based on [inclusive](#) time. If not specified, exclusive time is used by default.

- `-v|--verbose`

Output the parameters of each method in full. If not specified, parameters will be truncated.

Collect a trace with dotnet-trace

To collect traces using `dotnet-trace`:

- Get the process identifier (PID) of the .NET Core application to collect traces from.

- On Windows, you can use Task Manager or the `tasklist` command, for example.
- On Linux, for example, the `ps` command.
- `dotnet-trace ps`
- Run the following command:

```
dotnet-trace collect --process-id <PID>
```

The preceding command generates output similar to the following:

```
Press <Enter> to exit...
Connecting to process: <Full-Path-To-Process-Being-Profiled>/dotnet.exe
Collecting to file: <Full-Path-To-Trace>/trace.nettrace
Session Id: <SessionId>
Recording trace 721.025 (KB)
```

- Stop collection by pressing the `<Enter>` key. `dotnet-trace` will finish logging events to the `trace.nettrace` file.

Launch a child application and collect a trace from its startup using dotnet-trace

IMPORTANT

This works for apps running .NET 5 or later only.

Sometimes it may be useful to collect a trace of a process from its startup. For apps running .NET 5 or later, it is possible to do this by using dotnet-trace.

This will launch `hello.exe` with `arg1` and `arg2` as its command-line arguments and collect a trace from its runtime startup:

```
dotnet-trace collect -- hello.exe arg1 arg2
```

The preceding command generates output similar to the following:

```
No profile or providers specified, defaulting to trace profile 'cpu-sampling'

Provider Name          Keywords          Level          Enabled By
Microsoft-DotNETCore-SampleProfiler 0x0000F00000000000 Informational(4)  --profile
Microsoft-Windows-DotNETRuntime     0x00000014C14FCCBD  Informational(4)  --profile

Process      : E:\temp\gcpersim\bin\Debug\net5.0\gcpersim.exe
Output File  : E:\temp\gcpersim\trace.nettrace

[00:00:00:05] Recording trace 122.244 (KB)
Press <Enter> or <Ctrl+C> to exit...
```

You can stop collecting the trace by pressing `<Enter>` or `<Ctrl + C>` key. Doing this will also exit `hello.exe`.

NOTE

Launching `hello.exe` via `dotnet-trace` will redirect its input/output and you won't be able to interact with it on the console by default. Use the `--show-child-io` switch to interact with its `stdin/stdout`. Exiting the tool via `CTRL+C` or `SIGTERM` will safely end both the tool and the child process. If the child process exits before the tool, the tool will exit as well and the trace should be safely viewable.

Use diagnostic port to collect a trace from app startup

IMPORTANT

This works for apps running .NET 5 or later only.

Diagnostic port is a runtime feature added in .NET 5 that allows you to start tracing from app startup. To do this using `dotnet-trace`, you can either use `dotnet-trace collect -- <command>` as described in the examples above, or use the `--diagnostic-port` option.

Using `dotnet-trace <collect|monitor> -- <command>` to launch the application as a child process is the simplest way to quickly trace the application from its startup.

However, when you want to gain a finer control over the lifetime of the app being traced (for example, monitor the app for the first 10 minutes only and continue executing) or if you need to interact with the app using the CLI, using `--diagnostic-port` option allows you to control both the target app being monitored and `dotnet-trace`.

1. The command below makes `dotnet-trace` create a diagnostics socket named `myport.sock` and wait for a connection.

```
dotnet-trace collect --diagnostic-port myport.sock
```

Output:

```
Waiting for connection on myport.sock
Start an application with the following environment variable:
DOTNET_DiagnosticPorts=/home/user/myport.sock
```

2. In a separate console, launch the target application with the environment variable `DOTNET_DiagnosticPorts` set to the value in the `dotnet-trace` output.

```
export DOTNET_DiagnosticPorts=/home/user/myport.sock
./my-dotnet-app arg1 arg2
```

This should then enable `dotnet-trace` to start tracing `my-dotnet-app`:

```
Waiting for connection on myport.sock
Start an application with the following environment variable: DOTNET_DiagnosticPorts=myport.sock
Starting a counter session. Press Q to quit.
```

IMPORTANT

Launching your app with `dotnet run` can be problematic because the dotnet CLI may spawn many child processes that are not your app and they can connect to `dotnet-trace` before your app, leaving your app to be suspended at run time. It is recommended you directly use a self-contained version of the app or use `dotnet exec` to launch the application.

View the trace captured from dotnet-trace

On Windows, you can view `.nettrace` files in [Visual Studio](#) or [PerfView](#) for analysis.

On Linux, you can view the trace by changing the output format of `dotnet-trace` to `speedscope`. Change the output file format by using the `-f|--format` option. You can choose between `nettrace` (the default option) and `speedscope`. The option `-f speedscope` will make `dotnet-trace` produce a `speedscope` file. `Speedscope` files can be opened at <https://www.speedscope.app>.

For traces collected on non-Windows platforms, you can also move the trace file to a Windows machine and view it in Visual Studio or PerfView.

NOTE

The .NET Core runtime generates traces in the `nettrace` format. The traces are converted to `speedscope` (if specified) after the trace is completed. Since some conversions may result in loss of data, the original `nettrace` file is preserved next to the converted file.

Use dotnet-trace to collect counter values over time

`dotnet-trace` can:

- Use `EventCounter` for basic health monitoring in performance-sensitive environments. For example, in production.
- Collect traces so they don't need to be viewed in real time.

For example, to collect runtime performance counter values, use the following command:

```
dotnet-trace collect --process-id <PID> --providers System.Runtime:0:1:EventCounterIntervalSec=1
```

The preceding command tells the runtime counters to report once every second for lightweight health monitoring. Replacing `EventCounterIntervalSec=1` with a higher value (for example, 60) allows collection of a smaller trace with less granularity in the counter data.

The following command reduces overhead and trace size more than the preceding one:

```
dotnet-trace collect --process-id <PID> --providers System.Runtime:0:1:EventCounterIntervalSec=1,Microsoft-Windows-DotNETRuntime:0:1,Microsoft-DotNETCore-SampleProfiler:0:1
```

The preceding command disables runtime events and the managed stack profiler.

Use .rsp file to avoid typing long commands

You can launch `dotnet-trace` with an `.rsp` file that contains the arguments to pass. This can be useful when enabling providers that expect lengthy arguments or when using a shell environment that strips characters.

For example, the following provider can be cumbersome to type out each time you want to trace:

```
dotnet-trace collect --providers Microsoft-Diagnostics-
DiagnosticSource:0x3:5:FilterAndPayloadSpecs="SqlClientDiagnosticListener/System.Data.SqlClient.WriteCommand
Before@Activity1Start:-
Command;Command.CommandText;ConnectionId;Operation;Command.Connection.ServerVersion;Command.CommandTimeout;C
ommand.CommandType;Command.Connection.ConnectionString;Command.Connection.Database;Command.Connection.DataSource;Command.Connection.PacketSize\r\nSqlClientDiagnosticListener/System.Data.SqlClient.WriteCommandAfter@Ac
tivity1Stop:\r\nMicrosoft.EntityFrameworkCore/Microsoft.EntityFrameworkCore.Database.Command.CommandExecutin
g@Activity2Start:-
Command;Command.CommandText;ConnectionId;IsAsync;Command.Connection.ClientConnectionId;Command.Connection.Se
rverVersion;Command.CommandTimeout;Command.CommandType;Command.Connection.ConnectionString;Command.Connectio
n.Database;Command.Connection.DataSource;Command.Connection.PacketSize\r\nMicrosoft.EntityFrameworkCore/Micr
osoft.EntityFrameworkCore.Database.Command.CommandExecuted@Activity2Stop:",OtherProvider,AnotherProvider
```

In addition, the previous example contains `"` as part of the argument. Because quotes are not handled equally by each shell, you may experience various issues when using different shells. For example, the command to enter in `zsh` is different to the command in `cmd`.

Instead of typing this each time, you can save the following text into a file called `myprofile.rsp`.

```
--providers
Microsoft-Diagnostics-
DiagnosticSource:0x3:5:FilterAndPayloadSpecs="SqlClientDiagnosticListener/System.Data.SqlClient.WriteCommand
Before@Activity1Start:-
Command;Command.CommandText;ConnectionId;Operation;Command.Connection.ServerVersion;Command.CommandTimeout;C
ommand.CommandType;Command.Connection.ConnectionString;Command.Connection.Database;Command.Connection.DataSource;Command.Connection.PacketSize\r\nSqlClientDiagnosticListener/System.Data.SqlClient.WriteCommandAfter@Ac
tivity1Stop:\r\nMicrosoft.EntityFrameworkCore/Microsoft.EntityFrameworkCore.Database.Command.CommandExecutin
g@Activity2Start:-
Command;Command.CommandText;ConnectionId;IsAsync;Command.Connection.ClientConnectionId;Command.Connection.Se
rverVersion;Command.CommandTimeout;Command.CommandType;Command.Connection.ConnectionString;Command.Connectio
n.Database;Command.Connection.DataSource;Command.Connection.PacketSize\r\nMicrosoft.EntityFrameworkCore/Micr
osoft.EntityFrameworkCore.Database.Command.CommandExecuted@Activity2Stop:",OtherProvider,AnotherProvider
```

Once you've saved `myprofile.rsp`, you can launch `dotnet-trace` with this configuration using the following command:

```
dotnet-trace @myprofile.rsp
```

See also

- [Well-known event providers from .NET](#)

Inspect managed stack traces (dotnet-stack)

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article applies to: ✓ `dotnet-stack` version 5.0.221401 and later versions

Install

There are two ways to download and install `dotnet-stack`:

- **dotnet global tool:**

To install the latest release version of the `dotnet-stack` NuGet package, use the `dotnet tool install` command:

```
dotnet tool install --global dotnet-stack
```

- **Direct download:**

Download the tool executable that matches your platform:

OS	Platform
Windows	x86 x64 Arm Arm-x64
macOS	x64
Linux	x64 Arm Arm64 musl-x64 musl-Arm64

Synopsis

```
dotnet-stack [-h, --help] [--version] <command>
```

Description

The `dotnet-stack` tool:

- Is a cross-platform .NET Core tool.
- Captures and prints the managed stacks for all threads in the target .NET process.
- Utilizes `EventPipe` tracing provided by the .NET Core runtime.

Options

- `-h|--help`

Shows command-line help.

- `--version`

Displays the version of the dotnet-stack utility.

Commands

COMMAND	DESCRIPTION
<code>dotnet-stack report</code>	Prints the stack trace for each thread in the target process.
<code>dotnet-stack ps</code>	Lists the dotnet processes that stack traces can be collected from.

dotnet-stack report

Prints the stack trace for each thread in the target process.

Synopsis

```
dotnet-stack report -p|--process-id <pid>
                    -n|--name <process-name>
                    [-h|--help]
```

Options

- `-n, --name <name>`

The name of the process to collect the trace from.

- `-p|--process-id <PID>`

The process ID to collect the trace from.

dotnet-stack ps

Lists the dotnet processes that stack traces can be collected from. `dotnet-stack` version 6.0.320703 and later versions also display the command-line arguments that each process was started with, if available.

Synopsis

```
dotnet-stack ps [-h|--help]
```

Example

Suppose you start a long-running app using the command `dotnet run --configuration Release`. In another window, you run the `dotnet-stack ps` command. The output you'll see is as follows. The command-line arguments, if any, are shown in `dotnet-stack` version 6.0.320703 and later.

```
> dotnet-stack ps

21932 dotnet      C:\Program Files\dotnet\dotnet.exe    run --configuration Release
36656 dotnet      C:\Program Files\dotnet\dotnet.exe
```

Report managed stacks with dotnet-stack

To report managed stacks using `dotnet-stack`:

- Get the process identifier (PID) of the .NET Core application to report stacks from.
 - On Windows, you can use Task Manager or the `tasklist` command, for example.

- On Linux, for example, the `ps` command.
- `dotnet-stack ps`

- Run the following command:

```
dotnet-stack report --process-id <PID>
```

The preceding command generates output similar to the following:

```
Thread (0x48839B):
[Native Frames]
System.Console!System.IO.StdInReader.ReadKey(bool&)
System.Console!System.IO.SyncTextReader.ReadKey(bool&)
System.Console!System.ConsolePal.ReadKey(bool)
System.Console!System.Console.ReadKey()
StackTracee!Tracee.Program.Main(class System.String[])
```

The output of `dotnet-stack` follows the following form:

- Comments in the output are prefixed with `#`.
- Each thread has a header that includes the native thread ID: `Thread (<thread-id>):`.
- Stack frames follow the form `Module!Method`.
- Transitions to unmanaged code are represented as `[Native Frames]` in the output.

```
# comment
Thread (0x1234):
  module!Method
  module!Method

Thread (0x5678):
  [Native Frames]
  Module!Method
  Module!Method
```

NOTE

Stopping the process can take a long time (up to several minutes) for very large applications. The runtime needs to send over the type and method information for all managed code that was captured to resolve function names.

Next steps

- [Use dotnet-trace to collect CPU samples of a .NET application](#)
- [Use dotnet-dump to collect a dump of a .NET application](#)

Symbol downloader (dotnet-symbol)

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 2.1 SDK and later versions

Install

To install the latest release version of the `dotnet-symbol` NuGet package, use the `dotnet tool install` command:

```
dotnet tool install --global dotnet-symbol
```

Synopsis

```
dotnet-symbol [-h|--help] [options] <FILES>
```

Description

The `dotnet-symbol` global tool downloads files (symbols, DAC, modules, etc.) needed for debugging core dumps and minidumps. This can be useful when debugging dumps captured on another machine. `dotnet-symbol` can download modules and symbols needed to analyze the dump.

Options

- `--microsoft-symbol-server`
Add 'http://msdl.microsoft.com/download/symbols' symbol server path (default).
- `--server-path <symbol server path>`
Add a symbol server to the server path.
- `authenticated-server-path <pat> <server path>`
Add an authenticated symbol server to the server path using a personal access token (PAT).
- `--cache-directory <file cache directory>`
Adds a cache directory.
- `--recurse-subdirectories`
Process input files in all subdirectories.
- `--host-only`
Download only the host program (that is, dotnet) that lldb needs for loading core dumps.
- `--symbols`
Download symbol files (.pdb, .dbg, .dwarf).
- `--modules`

Download the module files (.dll, .so, .dylib).

- `--debugging`

Download the special debugging modules (DAC, DBI, SOS).

- `--windows-pdbs`

Force the downloading of the Windows PDBs when Portable PDBs are also available.

- `-o, --output <output directory>`

Set the output directory. Otherwise, write next to the input file (default).

- `-d, --diagnostics`

Enable diagnostic output.

- `-h|--help`

Shows command-line help.

Download symbols

Running `dotnet-symbol` against a dump file will, by default, download all the modules, symbols, and DAC/DBI files needed to debug the dump including the managed assemblies. Because SOS can now download symbols when needed, most Linux core dumps can be analyzed using lldb with only the host (dotnet) and debugging modules. To get these files necessary for diagnosing a core dump with lldb run:

```
dotnet-symbol --host-only --debugging <dump file path>
```

Troubleshoot

- 404 Not Found while downloading symbols.

Symbol download is only supported for official .NET Core runtime versions acquired through official channels such as [the official web site](#) and the [default sources in the dotnet installation scripts](#). A 404 error while downloading debugging files may indicate that the dump was created with a .NET Core runtime from another source, such as one built from source locally or for a particular Linux distro, or from community sites like archlinux. In such cases, file necessary for debugging (dotnet, libcoreclr.so, and libmscordaccore.so) should be copied from those sources or from the environment the dump file was created in.

See also

- [Debugging with symbols](#)
- [Symbols and Portable PDBs](#)

SOS installer (dotnet-sos)

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 2.1 SDK and later versions

Install

There are two ways to download and install `dotnet-sos`:

- **dotnet global tool:**

To install the latest release version of the `dotnet-sos` NuGet package, use the `dotnet tool install` command:

```
dotnet tool install --global dotnet-sos
```

- **Direct download:**

Download the tool executable that matches your platform:

OS	Platform
Windows	x86 x64 Arm Arm-x64
macOS	x64
Linux	x64 Arm Arm64 musl-x64 musl-Arm64

Synopsis

```
dotnet-sos [-h|--help] [options] [command]
```

Description

The `dotnet-sos` global tool installs the [SOS debugger extension](#). This extension lets you inspect managed .NET Core state from native debuggers like lldb and windbg.

NOTE

Installing SOS via the `dotnet-sos` tool is only needed on Linux or macOS. It may also be needed on Windows if you're using older debugging tools. Recent versions of the [Windows Debugger](#) (>= version 10.0.18317.1001 of WinDbg or cdb) load SOS automatically from the Microsoft extension gallery.

Options

- `--version`

Displays version information.

- `-h|--help`

Shows command-line help.

dotnet-sos install

Installs the [SOS extension](#) locally for debugging .NET Core processes. On macOS and Linux, the `.lldbinit` file will be updated so that the extension automatically loads at lldb startup. If you're installing SOS on Windows with older debugging tools (prior to version 10.0.18317.1001), you will need to manually load the extension in WinDbg or cdb by running `.load %USERPROFILE%\dotnet\sos\sos.dll` in the debugger.

Synopsis

```
dotnet-sos install [--architecture <arch>]
```

Options

- `--architecture <arch>`

Specifies the processor architecture of the SOS binaries to install. By default, `dotnet-sos` installs the architecture of the host machine. Use this option when you want to install SOS for an architecture that's different from the dotnet host architecture. For example, if you're running Arm32 binaries from an Arm64 host, you will need to install SOS with `dotnet-sos install --architecture Arm`.

The following architectures are available:

- `Arm`
- `Arm64`
- `X86`
- `X64`

dotnet-sos uninstall

Uninstalls the [SOS extension](#) and, on Linux and macOS, removes it from lldb configuration.

Synopsis

```
dotnet-sos uninstall
```

dotnet-dsrouter

9/20/2022 • 8 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET 6.0 SDK and later versions

Install

There are two ways to download and install `dotnet-dsrouter`:

- **dotnet global tool:**

To install the latest release version of the `dotnet-dsrouter` NuGet package, use the `dotnet tool install` command:

```
dotnet tool install --global dotnet-dsrouter
```

Synopsis

```
dotnet-dsrouter [-?, -h, --help] [--version] <command>
```

Description

The `dotnet-dsrouter` connects diagnostic tooling like `dotnet-trace` and `dotnet-counters` to .NET applications running on Android, iOS, and tvOS, regardless of whether they're running as an emulator, simulator, or on the device itself. Diagnostic tooling uses local inter-process communication (IPC) (Named Pipe, Unix Domain Socket) to connect and communicate with a .NET runtime. .NET applications running in sandboxed environments on emulators, simulators, and devices need alternative ways to communicate. The `dotnet-dsrouter` injects itself between existing diagnostic tooling and .NET mobile applications and creates a local representation of the application. The `dotnet-dsrouter` enables diagnostic tools to communicate with a remote .NET runtime as if it has been running on the local machine.

The communication between diagnostic tooling and `dotnet-dsrouter` uses the same IPC (Named Pipe, Unix Domain Socket) as used when connecting to a local .NET runtime. `dotnet-dsrouter` uses TCP/IP in its communication with remote .NET runtime and support several different connectivity scenarios to handle different needs and requirements used by different platforms. `dotnet-dsrouter` also implements additional support to simplify connectivity configuration when running in emulator, simulator, and on physical device attached over USB.

NOTE

`dotnet-dsrouter` is intended for development and testing and it's highly recommended to run `dotnet-dsrouter` over loopback interface (for example, `127.0.0.1`, `[::1]`). The connectivity features and port forwarding capabilities of `dotnet-dsrouter` handles all scenarios using local emulator, simulator or physical device connected over USB.

WARNING

Binding TCP server endpoint to anything except loopback interface (`localhost` , `127.0.0.1` or `[::1]`) is *not* recommended. Any connections towards TCP server endpoint will be unauthenticated and unencrypted.

`dotnet-dsrouter` is intended for development use and should only be run in development and testing environments.

Detailed usage of `dotnet-dsrouter` together with mobile applications is outline by respective .NET SDKs. This document will only include a couple of examples on how to run diagnostic tools against .NET application running on Android. For in-depth details on configuration and scenarios, see [Diagnostics Tracing](#).

Options

- `-?|-h|--help`

Shows command-line help.

- `--version`

Displays the version of the `dotnet-dsrouter` utility.

Commands

COMMAND

`dotnet-dsrouter client-server`

`dotnet-dsrouter server-server`

`dotnet-dsrouter server-client`

`dotnet-dsrouter client-client`

dotnet-dsrouter client-server

Start a .NET application diagnostics server routing local IPC server and remote TCP client. The router is configured using an IPC client (connecting diagnostic tool IPC server) and a TCP/IP server (accepting runtime TCP client).

Synopsis

```
dotnet-dsrouter client-server
  [-ipcc|--ipc-client <ipcClient>]
  [-tcps|--tcp-server <tcpServer>]
  [-rt|--runtime-timeout <timeout>]
  [-v|--verbose <level>]
  [-fp|--forward-port <platform>]
```

Options

- `-ipcc, --ipc-client <ipcClient>` The diagnostic tool diagnostics server IPC address (`--diagnostic-port` argument). Router connects diagnostic tool IPC server when establishing a new route between runtime and diagnostic tool.
- `-tcps, --tcp-server <tcpServer>` The router TCP/IP address using format `[host]:[port]` . Router can bind one (`127.0.0.1` , `[::1]` , `0.0.0.0` , `[::]` , IPv4 address, IPv6 address, hostname) or all (*) interfaces.

Launch runtime using `DOTNET_DiagnosticPorts` environment variable, connecting router TCP server during startup.

- `-rt, --runtime-timeout <runtimeTimeout>` Automatically shut down router if no runtime connects to it before specified timeout (seconds). If not specified, router won't trigger an automatic shutdown.
- `-v, --verbose <verbose>` Enable verbose logging (debug|trace)
- `-fp, --forward-port <forwardPort>` Enable port forwarding, values `Android` or `ios` for `TcpClient`, and only `Android` for `TcpServer`. Make sure to set `ANDROID_SDK_ROOT` before using this option on Android.

dotnet-dsrouter server-server

Start a .NET application diagnostics server routing local IPC client and remote TCP client. The router is configured using an IPC server (connecting to by diagnostic tools) and a TCP/IP server (accepting runtime TCP client).

Synopsis

```
dotnet-dsrouter server-server
  [-ipcs|--ipc-server <ipcServer>]
  [-tcps|--tcp-server <tcpServer>]
  [-rt|--runtime-timeout <timeout>]
  [-v|--verbose <level>]
  [-fp|--forward-port <platform>]
```

Options

- `-ipcs, --ipc-server <ipcServer>` The diagnostics server IPC address to route. Router accepts IPC connections from diagnostic tools establishing a new route between runtime and diagnostic tool. If not specified router will use default IPC diagnostics server path.
- `-tcps, --tcp-server <tcpServer>` The router TCP/IP address using format `[host]:[port]`. Router can bind one (`127.0.0.1`, `[:1]`, `0.0.0.0`, `[::]`, IPv4 address, IPv6 address, hostname) or all (*) interfaces.
Launch runtime using `DOTNET_DiagnosticPorts` environment variable, connecting router TCP server during startup.
- `-rt, --runtime-timeout <runtimeTimeout>` Automatically shut down router if no runtime connects to it before specified timeout (seconds). If not specified, router won't trigger an automatic shutdown.
- `-v, --verbose <verbose>` Enable verbose logging (debug|trace)
- `-fp, --forward-port <forwardPort>` Enable port forwarding, values `Android` or `ios` for `TcpClient`, and only `Android` for `TcpServer`. Make sure to set `ANDROID_SDK_ROOT` before using this option on Android.

dotnet-dsrouter server-client

Start a .NET application diagnostics server routing local IPC client and remote TCP server. The router is configured using an IPC server (connecting to by diagnostic tools) and a TCP/IP client (connecting runtime TCP server).

Synopsis

```
dotnet-dsrouter server-client
  [-ipcs|--ipc-server <ipcServer>]
  [-tcpc|--tcp-client <tcpClient>]
  [-rt|--runtime-timeout <timeout>]
  [-v|--verbose <level>]
  [-fp|--forward-port <platform>]
```

Options

- `-ipcs, --ipc-server <ipcServer>` The diagnostics server IPC address to route. Router accepts IPC connections from diagnostic tools establishing a new route between runtime and diagnostic tool. If not specified router will use default IPC diagnostics server path.
- `-tcpc, --tcp-client <tcpClient>` The runtime TCP/IP address using format `[host]:[port]`. Router can connect `127.0.0.1`, `[:1]`, IPv4 address, IPv6 address, hostname addresses. Launch runtime using `DOTNET_DiagnosticPorts` environment variable to set up listener.
- `-rt, --runtime-timeout <runtimeTimeout>` Automatically shut down router if no runtime connects to it before specified timeout (seconds). If not specified, router won't trigger an automatic shutdown.
- `-v, --verbose <verbose>` Enable verbose logging (debug|trace)
- `-fp, --forward-port <forwardPort>` Enable port forwarding, values `Android` or `ios` for `TcpClient`, and only `Android` for `TcpServer`. Make sure to set `ANDROID_SDK_ROOT` before using this option on Android.

dotnet-dsrouter client-client

Start a .NET application diagnostics server routing local IPC server and remote TCP server. The router is configured using an IPC client (connecting diagnostic tool IPC server) and a TCP/IP client (connecting runtime TCP server).

Synopsis

```
dotnet-dsrouter client-client
  [-ipcc|--ipc-client <ipcClient>]
  [-tcpc|--tcp-client <tcpClient>]
  [-rt|--runtime-timeout <timeout>]
  [-v|--verbose <level>]
  [-fp|--forward-port <platform>]
```

Options

- `-ipcc, --ipc-client <ipcClient>` The diagnostic tool diagnostics server IPC address (`--diagnostic-port argument`). Router connects diagnostic tool IPC server when establishing a new route between runtime and diagnostic tool.
- `-tcpc, --tcp-client <tcpClient>` The runtime TCP/IP address using format `[host]:[port]`. Router can connect `127.0.0.1`, `[:1]`, IPv4 address, IPv6 address, hostname addresses. Launch runtime using `DOTNET_DiagnosticPorts` environment variable to set up listener.
- `-rt, --runtime-timeout <runtimeTimeout>` Automatically shut down router if no runtime connects to it before specified timeout (seconds). If not specified, router won't trigger an automatic shutdown.
- `-v, --verbose <verbose>` Enable verbose logging (debug|trace)
- `-fp, --forward-port <forwardPort>` Enable port forwarding, values `Android` or `ios` for `TcpClient`, and only `Android` for `TcpServer`. Make sure to set `ANDROID_SDK_ROOT` before using this option on Android.

Collect a startup trace using dotnet-trace from a .NET application running on Android

Sometimes it may be useful to collect a trace of an application from its startup. The following steps illustrate the process of doing so targeting a .NET application running on Android. Since `dotnet-dsrouter` is run using port forwarding, the same scenario works against applications running on a local emulator and on a physical device attached over USB. Make sure to set `ANDROID_SDK_ROOT` before using this option, or `dotnet-dsrouter` won't be able to find `adb` needed to set up port forwarding.

- Launch `dotnet-dsrouter` in server-server mode:

```
dotnet-dsrouter server-server -ipcs ~/mylocalport -tcps 127.0.0.1:9000 --forward-port Android &
```

- Set `DOTNET_DiagnosticPorts` environment variable using `AndroidEnvironment`:

Create a file in the same directory as the `.csproj` using a name like `app.env`, add environment variables into file, `DOTNET_DiagnosticPorts=127.0.0.1:9000,suspend` and include following `ItemGroup` into `.csproj`:

```
<ItemGroup Condition="'$(AndroidEnableProfiler)'=='true'>
  <AndroidEnvironment Include="app.env" />
</ItemGroup>
```

It's also possible to set `DOTNET_DiagnosticPorts` using `adb shell setprop`:

```
adb shell setprop debug.mono.profile '127.0.0.1:9000,suspend'
```

- Build and launch the application using .NET Android SDK, and enable tracing by passing `/p:AndroidEnableProfiler=true` to MSBuild. Since the app has been configured to suspend on startup, it will connect back to the `dotnet-dsrouter` TCP/IP listener running on `127.0.0.1:9000` and wait for diagnostic tooling to connect before resuming application execution.
- Start `dotnet-trace` in collect mode, connecting to `dotnet-dsrouter` IPC server, `~/mylocalport`:

```
dotnet-trace collect --diagnostic-port ~/mylocalport,connect
```

`dotnet-trace` will start a trace session and resume application that will now continue to execute. A stream of events will start to flow from the mobile application, through `dotnet-dsrouter` into `dotnet-trace` nettrace file. When done tracing, press `Enter` to make sure trace session is properly closed making sure nettrace file includes all needed data before application gets closed.

It's possible to run several trace sessions against the same running application over time, leave `dotnet-dsrouter` running and rerun `dotnet-trace` when a new trace session is needed.

`dotnet-dsrouter` can be left running in background and reused if an application is configured to connect using its address and port.

`dotnet-dsrouter` is tied to one running application at any time. If there are needs to trace several different applications at the same time, each application needs to use its own `dotnet-dsrouter` instance, by setting up a unique IPC, TCP/IP address pair in `dotnet-dsrouter` and configure different application instances to connect back to its unique `dotnet-dsrouter` instance.

If `dotnet-dsrouter` is run with `--forward-port` targeting Android and `adb` server, emulator or device gets restarted, all `dotnet-dsrouter` instances need to be restarted as well to restore port forwarding rules.

When done using `dotnet-dsrouter`, press Q or Ctrl + C to quit application.

NOTE

When running `dotnet-dsrouter` on Windows it will use Named Pipes for its IPC channel. Replace `~/mylocalport` with `mylocalport` in above examples when running on Windows.

NOTE

TCP/IP port 9000 is just an example. Any free TCP/IP port can be used.

NOTE

Unix Domain Socket `~/mylocalport` is just an example. Any free Unix Domain Socket file path can be used.

Collect a trace using dotnet-trace from a .NET application running on Android

If there's no need to collect a trace during application startup, it's possible to launch application in `nosuspend` mode, meaning runtime won't block at startup waiting for diagnostic tooling to connect before resuming execution. Most of the above-described scenario applies to this mode as well, just replace `suspend` with `nosuspend` in `DOTNET_DiagnosticPorts` environment variable to launch application in `nosuspend` mode.

See also

- [Android .NET Tracing](#)
- [.NET MAUI Tracing](#)
- [.NET Diagnostics Tracing on mobile](#)

Trace .NET applications with PerfCollect

9/20/2022 • 10 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 2.1 SDK and later versions

When performance problems are encountered on Linux, collecting a trace with `perfcollect` can be used to gather detailed information about what was happening on the machine at the time of the performance problem.

`perfcollect` is a bash script that uses [Linux Trace Toolkit: next generation \(LTTng\)](#) to collect events written from the runtime or any [EventSource](#), as well as `perf` to collect CPU samples of the target process.

Prepare your machine

Follow these steps to prepare your machine to collect a performance trace with `perfcollect`.

NOTE

If you are in a container environment, your container needs to have `SYS_ADMIN` capability. For more information on tracing applications inside containers using PerfCollect, see [Collect diagnostics in containers](#).

1. Download `perfcollect`.

```
curl -OL https://aka.ms/perfcollect
```

2. Make the script executable.

```
chmod +x perfcollect
```

3. Install tracing prerequisites - these are the actual tracing libraries.

```
sudo ./perfcollect install
```

This will install the following prerequisites on your machine:

- a. `perf`: the Linux Performance Events subsystem and companion user-mode collection/viewer application. `perf` is part of the Linux kernel source, but is not usually installed by default.
- b. `LTTng`: Used to capture event data emitted at run time by CoreCLR. This data is then used to analyze the behavior of various runtime components such as the GC, JIT, and thread pool.

Recent versions of .NET Core and the Linux perf tool support automatic resolution of method names for framework code. If you are working with .NET Core version 3.1 or less, an extra step is necessary. See [Resolving Framework Symbols](#) for details.

For resolving method names of native runtime DLLs (such as libcoreclr.so), `perfcollect` will resolve symbols for them when it converts the data, but only if the symbols for these binaries are present. See [Getting Symbols for](#)

the Native Runtime section for details.

Collect a trace

1. Have two shells available - one for controlling tracing, referred to as [Trace], and one for running the application, referred to as [App].
2. [Trace] Start collection.

```
sudo ./perfcollect collect sampleTrace
```

Expected Output:

```
Collection started. Press CTRL+C to stop.
```

3. [App] Set up the application shell with the following environment variables - this enables tracing configuration of CoreCLR.

```
export DOTNET_PerfMapEnabled=1  
export DOTNET_EnableEventLog=1
```

NOTE

.NET 6 standardizes on the prefix `DOTNET_` instead of `COMPlus_` for environment variables that configure .NET run-time behavior. However, the `COMPlus_` prefix will continue to work. If you're using a previous version of the .NET runtime, you should still use the `COMPlus_` prefix for environment variables.

4. [App] Run the app - let it run as long as you need to in order to capture the performance problem. The exact length can be as short as you need as long as it sufficiently captures the window of time where the performance problem you want to investigate occurs.

```
dotnet run
```

5. [Trace] Stop collection - hit CTRL+C.

```
^C
...STOPPED.

Starting post-processing. This may take some time.

Generating native image symbol files
...SKIPPED
Saving native symbols
...FINISHED
Exporting perf.data file
...FINISHED
Compressing trace files
...FINISHED
Cleaning up artifacts
...FINISHED

Trace saved to sampleTrace.trace.zip
```

The compressed trace file is now stored in the current working directory.

View a trace

There are a number of options for viewing the trace that was collected. Traces are best viewed using [PerfView](#) on Windows, but they can be viewed directly on Linux using `PerfCollect` itself or `TraceCompass`.

Use `PerfCollect` to view the trace file

You can use `perfcollect` itself to view the trace that you collected. To do this, use the following command:

```
./perfcollect view sampleTrace.trace.zip
```

By default, this will show the CPU trace of the application using `perf`.

To look at the events that were collected via `LTTng`, you can pass in the flag `-viewer lttng` to see the individual events:

```
./perfcollect view sampleTrace.trace.zip -viewer lttng
```

This will use `babeltrace` viewer to print the events payload:

```
# [01:02:18.189217659] (+0.020132603) ubuntu-xenial DotNETRuntime:ExceptionThrown_V1: { cpu_id = 0 }, {
ExceptionType = "System.Exception", ExceptionMessage = "An exception happened", ExceptionEIP =
139875671834775, ExceptionHRESULT = 2148734208, ExceptionFlags = 16, ClrInstanceID = 0 }
# [01:02:18.189250227] (+0.020165171) ubuntu-xenial DotNETRuntime:ExceptionCatchStart: { cpu_id = 0 }, {
EntryEIP = 139873639728404, MethodID = 139873626968120, MethodName = "void [helloworld]
helloworld.Program::Main(string[])", ClrInstanceID = 0 }
```

Use [PerfView](#) to open the trace file

To see an aggregate view of both the CPU sample and the events, you can use [PerfView](#) on a Windows machine.

1. Copy the trace.zip file from Linux to a Windows machine.
2. Download PerfView from <https://aka.ms/perfview>.
3. Run `PerfView.exe`

```
PerfView.exe <path to trace.zip file>
```

PerfView will display the list of views that are supported based on the data contained in the trace file.

- For CPU investigations, choose **CPU stacks**.
- For detailed GC information, choose **GCStats**.
- For per-process/module/method JIT information, choose **JITStats**.
- If there is not a view for the information you need, you can try looking for the events in the raw events view. Choose **Events**.

For more information on how to interpret views in PerfView, see help links in the view itself, or from the main window in PerfView, choose **Help->Users Guide**.

NOTE

Events written via [System.Diagnostics.Tracing.EventSource](#) API (including the events from Framework) won't show up under their provider name. Instead, they are written as `EventSourceEvent` events under `Microsoft-Windows-DotNETRuntime` provider and their payloads are JSON serialized.

Use TraceCompass to open the trace file

[Eclipse TraceCompass](#) is another option you can use to view the traces. `TraceCompass` works on Linux machines as well, so you don't need to move your trace over to a Windows machine. To use `TraceCompass` to open your trace file, you will need to unzip the file.

```
unzip myTrace.trace.zip
```

`perfcollect` will save the LTTng trace it collected into a CTF file format in a subdirectory in the `lttngTrace`. Specifically, the CTF file will be located in a directory that looks like `lttngTrace/auto-20201025-101230\ust\uid\1000\64-bit\`.

You can open the CTF trace file in `TraceCompass` by selecting `File -> Open Trace` and select the `metadata` file.

For more details, please refer to [TraceCompass documentation](#).

Resolve framework symbols

Framework symbols need to be manually generated at the time the trace is collected. They are different than app-level symbols because the framework is pre-compiled while app code is just-in-time-compiled. For framework code that was precompiled to native code, you need to call `crossgen` that knows how to generate the mapping from the native code to the name of the methods.

`perfcollect` can handle most of the details for you, but it needs to have `crossgen` available. By default it is not installed with .NET distribution. If `crossgen` is not there, `perfcollect` warns you and refers you to these instructions. To fix things you need to fetch exactly the right version of crossgen for the runtime you are using. If you place the crossgen tool in the same directory as the .NET Runtime DLLs (for example, libcoreclr.so), then `perfcollect` can find it and add the framework symbols to the trace file for you.

Normally when you create a .NET application, it just generates the DLL for the code you wrote, using a shared copy of the runtime for the rest. However you can also generate what is called a 'self-contained' version of an application and this contains all runtime DLLs. `crossgen` is part of the NuGet package that is used to create self-

contained apps, so one way of getting the right version of `crossgen` is to create a self-contained package of your application.

For example:

```
mkdir helloWorld
cd helloWorld
dotnet new console
dotnet publish --self-contained -r linux-x64
```

This creates a new Hello World application and builds it as a self-contained app.

As a side effect of creating the self-contained application the dotnet tool will download a NuGet package called `runtime.linux-x64.microsoft.netcore.app` and place it in the directory `~/.nuget/packages/runtime.linux-x64.microsoft.netcore.app/VERSION`, where `VERSION` is the version number of your .NET Core runtime (for example, `2.1.0`). Under that is a tools directory and inside there is the `crossgen` tool you need. Starting with .NET Core 3.0, the package location is `~/.nuget/packages/microsoft.netcore.app.runtime.linux-x64/VERSION`.

The `crossgen` tool needs to be put next to the runtime that is actually used by your application. Typically your app uses the shared version of .NET Core that is installed at `/usr/share/dotnet/shared/Microsoft.NETCore.App/VERSION` where `VERSION` is the version number of the .NET Runtime. This is a shared location, so you need to be super-user to modify it. If the `VERSION` is `2.1.0` the commands to update `crossgen` would be:

```
sudo bash
cp ~/.nuget/packages/runtime.linux-x64.microsoft.netcore.app/2.1.0/tools/crossgen
/usr/share/dotnet/shared/Microsoft.NETCore.App/2.1.0
```

Once you have done this, `perfcollect` will use `crossgen` to include framework symbols. The warning that `perfcollect` used to issue should go away. This only has to be one once per machine (until you update your runtime).

Alternative: Turn off use of precompiled code

If you don't have the ability to update the .NET Runtime (to add `crossgen`), or if the above procedure did not work for some reason, there is another approach to getting framework symbols. You can tell the runtime to simply not use the precompiled framework code. The code will be Just-In-Time compiled and `crossgen` is not needed.

NOTE

Choosing this approach may increase the startup time for your application.

To do this, you can add the following environment variable:

```
export DOTNET_ZapDisable=1
```

NOTE

.NET 6 standardizes on the prefix `DOTNET_` instead of `COMPlus_` for environment variables that configure .NET run-time behavior. However, the `COMPlus_` prefix will continue to work. If you're using a previous version of the .NET runtime, you should still use the `COMPlus_` prefix for environment variables.

With this change, you should get the symbols for all .NET code.

Get symbols for the native runtime

Most of the time you are interested in your own code, which `perfcollect` resolves by default. Sometimes it is useful to see what is going on inside the .NET DLLs (which is what the last section was about), but sometimes what is going on in the native runtime DLLs (typically `libcoreclr.so`), is interesting. `perfcollect` will resolve the symbols for these when it converts its data, but only if the symbols for these native DLLs are present (and are beside the library they are for).

There is a global command called `dotnet-symbol` that does this. To use `dotnet-symbol` to get native runtime symbols:

1. Install `dotnet-symbol`:

```
dotnet tool install -g dotnet-symbol
```

2. Download the symbols. If your installed version of the .NET Core runtime is 2.1.0, the command to do this is:

```
mkdir mySymbols  
dotnet symbol --symbols --output mySymbols  
/usr/share/dotnet/shared/Microsoft.NETCore.App/2.1.0/lib*.so
```

3. Copy the symbols to the correct place.

```
sudo cp mySymbols/* /usr/share/dotnet/shared/Microsoft.NETCore.App/2.1.0
```

If this cannot be done because you do not have write access to the appropriate directory, you can use `perf buildid-cache` to add the symbols.

After this, you should get symbolic names for the native DLLs when you run `perfcollect`.

Collect in a Docker container

For more information on how to use `perfcollect` in container environments, see [Collect diagnostics in containers](#).

Learn more about collection options

You can specify the following optional flags with `perfcollect` to better suit your diagnostic needs.

Collect for a specific duration

When you want to collect a trace for a specific duration, you can use `-collectsec` option followed by a number specifying the total seconds to collect a trace for.

Collect threadtime traces

Specifying `-threadtime` with `perfcollect` lets you collect per-thread CPU usage data. This lets you analyze where every thread was spending its CPU time.

Collect traces for managed memory and garbage collector performance

The following options let you specifically collect the GC events from the runtime.

- `perfcollect collect -gccollectonly`

Collect only a minimal set of GC Collection events. This is the least verbose GC eventing collection profile with the lowest impact on the target app's performance. This command is analogous to `PerfView.exe /GCCollectOnly collect` command in PerfView.

- `perfcollect collect -gconly`

Collect more verbose GC collection events with JIT, Loader, and Exception events. This requests more verbose events (such as the allocation information and GC join information) and will have more impact to the target app's performance than `-gccollectonly` option. This command is analogous to `PerfView.exe /GCOonly collect` command in PerfView.

- `perfcollect collect -gcwithheap`

Collect the most verbose GC collection events, which tracks the heap survival and movements as well. This gives in-depth analysis of the GC behavior but will incur high performance cost as each GC can take more than two times longer. It is recommended you understand the performance implication of using this trace option when tracing in production environments.

Debug a memory leak in .NET Core

9/20/2022 • 5 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 3.1 SDK and later versions

A memory leak may happen when your app references objects that it no longer needs to perform the desired task. Referencing said objects makes the garbage collector to be unable to reclaim the memory used, often resulting in performance degradation and potentially end up throwing an [OutOfMemoryException](#).

This tutorial demonstrates the tools to analyze a memory leak in a .NET Core app using the .NET diagnostics CLI tools. If you are on Windows, you may be able to [use Visual Studio's Memory Diagnostic tools](#) to debug the memory leak.

This tutorial uses a sample app, which is designed to intentionally leak memory. The sample is provided as an exercise. You can analyze an app that is unintentionally leaking memory too.

In this tutorial, you will:

- Examine managed memory usage with [dotnet-counters](#).
- Generate a dump file.
- Analyze the memory usage using the dump file.

Prerequisites

The tutorial uses:

- [.NET Core 3.1 SDK](#) or a later version.
- [dotnet-counters](#) to check managed memory usage.
- [dotnet-dump](#) to collect and analyze a dump file.
- A [sample debug target](#) app to diagnose.

The tutorial assumes the sample and tools are installed and ready to use.

Examine managed memory usage

Before you start collecting diagnostics data to help us root cause this scenario, you need to make sure you're actually seeing a memory leak (memory growth). You can use the [dotnet-counters](#) tool to confirm that.

Open a console window and navigate to the directory where you downloaded and unzipped the [sample debug target](#). Run the target:

```
dotnet run
```

From a separate console, find the process ID:

```
dotnet-counters ps
```

The output should be similar to:

```
4807 DiagnosticScena  
/home/user/git/samples/core/diagnostics/DiagnosticScenarios/bin/Debug/netcoreapp3.0/DiagnosticScenarios
```

Now, check managed memory usage with the `dotnet-counters` tool. The `--refresh-interval` specifies the number of seconds between refreshes:

```
dotnet-counters monitor --refresh-interval 1 -p 4807
```

The live output should be similar to:

```
Press p to pause, r to resume, q to quit.  
Status: Running  
  
[System.Runtime]  
# of Assemblies Loaded 118  
% Time in GC (since last GC) 0  
Allocation Rate (Bytes / sec) 37,896  
CPU Usage (%) 0  
Exceptions / sec 0  
GC Heap Size (MB) 4  
Gen 0 GC / sec 0  
Gen 0 Size (B) 0  
Gen 1 GC / sec 0  
Gen 1 Size (B) 0  
Gen 2 GC / sec 0  
Gen 2 Size (B) 0  
LOH Size (B) 0  
Monitor Lock Contention Count / sec 0  
Number of Active Timers 1  
ThreadPool Completed Work Items / sec 10  
ThreadPool Queue Length 0  
ThreadPool Threads Count 1  
Working Set (MB) 83
```

Focusing on this line:

GC Heap Size (MB)	4
-------------------	---

You can see that the managed heap memory is 4 MB right after startup.

Now, hit the URL `https://localhost:5001/api/diagscenario/memleak/20000`.

Observe that the memory usage has grown to 30 MB.

GC Heap Size (MB)	30
-------------------	----

By watching the memory usage, you can safely say that memory is growing or leaking. The next step is to collect the right data for memory analysis.

Generate memory dump

When analyzing possible memory leaks, you need access to the app's memory heap. Then you can analyze the memory contents. Looking at relationships between objects, you create theories on why memory isn't being freed. A common diagnostics data source is a memory dump on Windows or the equivalent core dump on Linux. To generate a dump of a .NET Core application, you can use the `dotnet-dump` tool.

Using the `sample debug target` previously started, run the following command to generate a Linux core dump:

```
dotnet-dump collect -p 4807
```

The result is a core dump located in the same folder.

```
Writing minidump with heap to ./core_20190430_185145
Complete
```

Restart the failed process

Once the dump is collected, you should have sufficient information to diagnose the failed process. If the failed process is running on a production server, now it's the ideal time for short-term remediation by restarting the process.

In this tutorial, you're now done with the [Sample debug target](#) and you can close it. Navigate to the terminal that started the server, and press **Ctrl+C**.

Analyze the core dump

Now that you have a core dump generated, use the [dotnet-dump](#) tool to analyze the dump:

```
dotnet-dump analyze core_20190430_185145
```

Where `core_20190430_185145` is the name of the core dump you want to analyze.

NOTE

If you see an error complaining that `/libdl.so` cannot be found, you may have to install the `/libc6-dev` package. For more information, see [Prerequisites for .NET Core on Linux](#).

You'll be presented with a prompt where you can enter SOS commands. Commonly, the first thing you want to look at is the overall state of the managed heap:

```
> dumpheap -stat

Statistics:
    MT      Count    TotalSize Class Name
...
00007f6c1eefba8      576        59904 System.Reflection.RuntimeMethodInfo
00007f6c1dc021c8     1749       95696 System.SByte[]
00000000008c9db0     3847      116080     Free
00007f6c1e784a18     175        128640 System.Char[]
00007f6c1dbf5510     217        133504 System.Object[]
00007f6c1dc014c0     467        416464 System.Byte[]
00007f6c21625038      6        4063376 testwebapi.Controllers.Customer[]
00007f6c20a67498    200000      4800000 testwebapi.Controllers.Customer
00007f6c1dc00f90    206770      19494060 System.String
Total 428516 objects
```

Here you can see that most objects are either `String` or `Customer` objects.

You can use the `dumpheap` command again with the method table (MT) to get a list of all the `String` instances:

```
> dumpheap -mt 00007faddaa50f90

Address          MT      Size
...
00007f6ad09421f8 00007faddaa50f90      94
...
00007f6ad0965b20 00007f6c1dc00f90      80
00007f6ad0965c10 00007f6c1dc00f90      80
00007f6ad0965d00 00007f6c1dc00f90      80
00007f6ad0965df0 00007f6c1dc00f90      80
00007f6ad0965ee0 00007f6c1dc00f90      80

Statistics:
MT      Count      TotalSize Class Name
00007f6c1dc00f90    206770      19494060 System.String
Total 206770 objects
```

You can now use the `gcroot` command on a `System.String` instance to see how and why the object is rooted. Be patient because this command takes several minutes with a 30-MB heap:

```
> gcroot -all 00007f6ad09421f8

Thread 3f68:
00007F6795BB58A0 00007F6C1D7D0745 System.Diagnostics.Tracing.CounterGroup.PollForValues()
[/_/src/System.Private.CoreLib/shared/System/Diagnostics/Tracing/CounterGroup.cs @ 260]
rbx: (interior)
-> 00007F6BDFFFF038 System.Object[]
-> 00007F69D0033570 testwebapi.Controllers.Processor
-> 00007F69D0033588 testwebapi.Controllers.CustomerCache
-> 00007F69D00335A0 System.Collections.Generic.List`1[[testwebapi.Controllers.Customer,
DiagnosticScenarios]]
-> 00007F6C000148A0 testwebapi.Controllers.Customer[]
-> 00007F6AD0942258 testwebapi.Controllers.Customer
-> 00007F6AD09421F8 System.String

HandleTable:
00007F6C98BB15F8 (pinned handle)
-> 00007F6BDFFFF038 System.Object[]
-> 00007F69D0033570 testwebapi.Controllers.Processor
-> 00007F69D0033588 testwebapi.Controllers.CustomerCache
-> 00007F69D00335A0 System.Collections.Generic.List`1[[testwebapi.Controllers.Customer,
DiagnosticScenarios]]
-> 00007F6C000148A0 testwebapi.Controllers.Customer[]
-> 00007F6AD0942258 testwebapi.Controllers.Customer
-> 00007F6AD09421F8 System.String

Found 2 roots.
```

You can see that the `String` is directly held by the `Customer` object and indirectly held by a `CustomerCache` object.

You can continue dumping out objects to see that most `String` objects follow a similar pattern. At this point, the investigation provided sufficient information to identify the root cause in your code.

This general procedure allows you to identify the source of major memory leaks.

Clean up resources

In this tutorial, you started a sample web server. This server should have been shut down as explained in the [Restart the failed process](#) section.

You can also delete the dump file that was created.

See also

- [dotnet-trace](#) to list processes
- [dotnet-counters](#) to check managed memory usage
- [dotnet-dump](#) to collect and analyze a dump file
- [dotnet/diagnostics](#)
- [Use Visual Studio to debug memory leaks](#)

Next steps

[Debug high CPU in .NET Core](#)

Debug high CPU usage in .NET Core

9/20/2022 • 6 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 3.1 SDK and later versions

In this tutorial, you'll learn how to debug an excessive CPU usage scenario. Using the provided example [ASP.NET Core web app](#) source code repository, you can cause a deadlock intentionally. The endpoint will stop responding and experience thread accumulation. You'll learn how you can use various tools to diagnose this scenario with several key pieces of diagnostics data.

In this tutorial, you will:

- Investigate high CPU usage
- Determine CPU usage with [dotnet-counters](#)
- Use [dotnet-trace](#) for trace generation
- Profile performance in PerfView
- Diagnose and solve excessive CPU usage

Prerequisites

The tutorial uses:

- [.NET Core 3.1 SDK](#) or a later version.
- [Sample debug target](#) to trigger the scenario.
- [dotnet-trace](#) to list processes and generate a profile.
- [dotnet-counters](#) to monitor cpu usage.

CPU counters

Before attempting to collect diagnostics data, you need to observe a high CPU condition. Run the [sample application](#) using the following command from the project root directory.

```
dotnet run
```

To find the process ID, use the following command:

```
dotnet-trace ps
```

Take note of the process ID from your command output. Our process ID was `22884`, but yours will be different.

To check the current CPU usage, use the [dotnet-counters](#) tool command:

```
dotnet-counters monitor --refresh-interval 1 -p 22884
```

The `refresh-interval` is the number of seconds between the counter polling CPU values. The output should be similar to the following:

```
Press p to pause, r to resume, q to quit.
```

```
Status: Running
```

```
[System.Runtime]
```

% Time in GC since last GC (%)	0
Allocation Rate / 1 sec (B)	0
CPU Usage (%)	0
Exception Count / 1 sec	0
GC Heap Size (MB)	4
Gen 0 GC Count / 60 sec	0
Gen 0 Size (B)	0
Gen 1 GC Count / 60 sec	0
Gen 1 Size (B)	0
Gen 2 GC Count / 60 sec	0
Gen 2 Size (B)	0
LOH Size (B)	0
Monitor Lock Contention Count / 1 sec	0
Number of Active Timers	1
Number of Assemblies Loaded	140
ThreadPool Completed Work Item Count / 1 sec	3
ThreadPool Queue Length	0
ThreadPool Thread Count	7
Working Set (MB)	63

With the web app running, immediately after startup, the CPU isn't being consumed at all and is reported at `0%`. Navigate to the `api/diagscenario/highcpu` route with `60000` as the route parameter:

```
https://localhost:5001/api/diagscenario/highcpu/60000
```

Now, rerun the `dotnet-counters` command. If interested in monitoring just the `cpu-usage` counter, add '`--counters System.Runtime[cpu-usage]`' to the previous command. We are unsure if the CPU is being consumed, so we will monitor the same list of counters as above to verify counter values are within expected range for our application.

```
dotnet-counters monitor -p 22884 --refresh-interval 1
```

You should see an increase in CPU usage as shown below (depending on the host machine, expect varying CPU usage):

```
Press p to pause, r to resume, q to quit.
```

```
Status: Running
```

[System.Runtime]	
% Time in GC since last GC (%)	0
Allocation Rate / 1 sec (B)	0
CPU Usage (%)	25
Exception Count / 1 sec	0
GC Heap Size (MB)	4
Gen 0 GC Count / 60 sec	0
Gen 0 Size (B)	0
Gen 1 GC Count / 60 sec	0
Gen 1 Size (B)	0
Gen 2 GC Count / 60 sec	0
Gen 2 Size (B)	0
LOH Size (B)	0
Monitor Lock Contention Count / 1 sec	0
Number of Active Timers	1
Number of Assemblies Loaded	140
ThreadPool Completed Work Item Count / 1 sec	3
ThreadPool Queue Length	0
ThreadPool Thread Count	7
Working Set (MB)	63

Throughout the duration of the request, the CPU usage will hover around the increased percentage.

TIP

To visualize an even higher CPU usage, you can exercise this endpoint in multiple browser tabs simultaneously.

At this point, you can safely say the CPU is running higher than you expect. Identifying the effects of a problem is key to finding the cause. We will use the effect of high CPU consumption in addition to diagnostic tools to find the cause of the problem.

Analyze High CPU with Profiler

When analyzing an app with high CPU usage, you need a diagnostics tool that can provide insights into what the code is doing. The usual choice is a profiler, and there are different profiler options to choose from.

`dotnet-trace` can be used on all operating systems, however, its limitations of safe-point bias and managed-only callstacks result in more general information compared to a kernel-aware profiler like 'perf' for Linux or ETW for Windows. If your performance investigation involves only managed code, generally `dotnet-trace` will be sufficient.

- [Linux](#)
- [Windows](#)

The `perf` tool can be used to generate .NET Core app profiles. We will demonstrate this tool, although `dotnet-trace` could be used as well. Exit the previous instance of the [sample debug target](#).

Set the `DOTNET_PerfMapEnabled` environment variable to cause the .NET app to create a `map` file in the `/tmp` directory. This `map` file is used by `perf` to map CPU addresses to JIT-generated functions by name. For more information, see [Export perf maps](#).

NOTE

.NET 6 standardizes on the prefix `DOTNET_` instead of `COMPlus_` for environment variables that configure .NET run-time behavior. However, the `COMPlus_` prefix will continue to work. If you're using a previous version of the .NET runtime, you should still use the `COMPlus_` prefix for environment variables.

Run the [sample debug target](#) in the same terminal session.

```
export DOTNET_PerfMapEnabled=1  
dotnet run
```

Exercise the high CPU API endpoint (<https://localhost:5001/api/diagscenario/highcpu/60000>) again. While it's running within the 1-minute request, run the `perf` command with your process ID:

```
sudo perf record -p 2266 -g
```

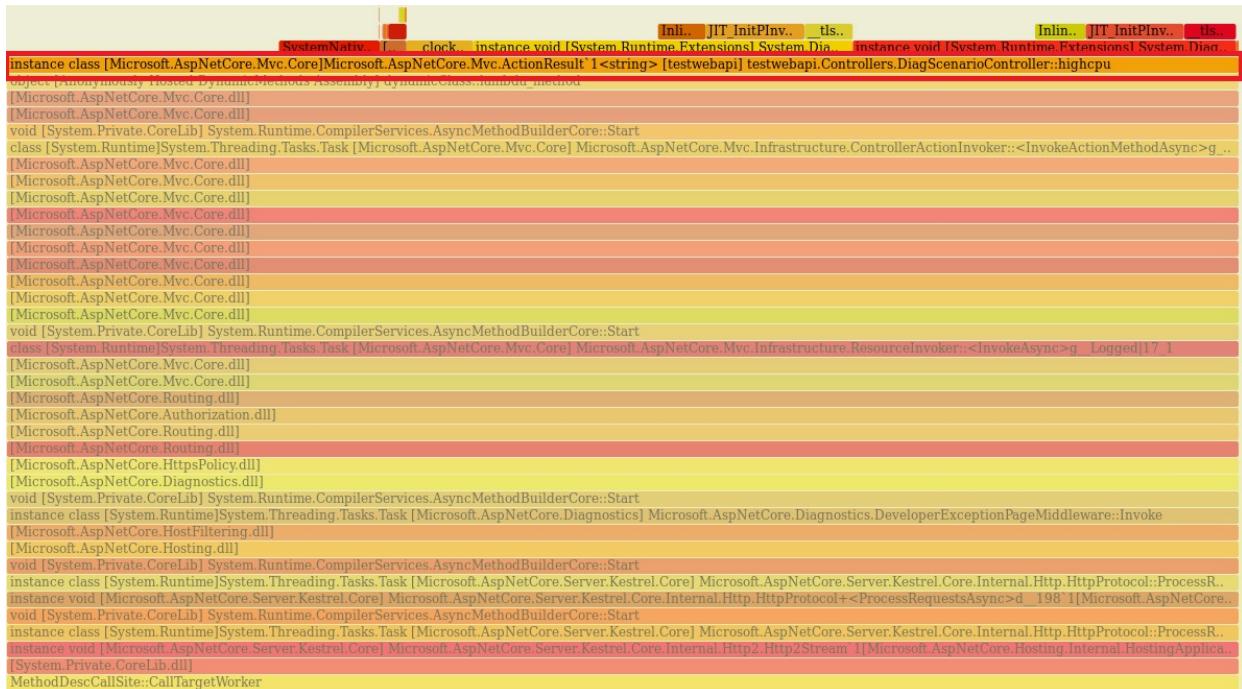
The `perf` command starts the performance collection process. Let it run for about 20-30 seconds, then press `Ctrl+C` to exit the collection process. You can use the same `perf` command to see the output of the trace.

```
sudo perf report -f
```

You can also generate a *flame-graph* by using the following commands:

```
git clone --depth=1 https://github.com/BrendanGregg/FlameGraph  
sudo perf script | FlameGraph/stackcollapse-perf.pl | FlameGraph/flamegraph.pl > flamegraph.svg
```

This command generates a `flamegraph.svg` that you can view in the browser to investigate the performance problem:



Analyzing High CPU Data with Visual Studio

All *.nettrace files can be analyzed in Visual Studio. To analyze a Linux *.nettrace file in Visual Studio, transfer the

*.nettrace file, in addition to the other necessary documents, to a Windows machine, and then open the *.nettrace file in Visual Studio. For more information, see [Analyze CPU Usage Data](#).

See also

- [dotnet-trace](#) to list processes
- [dotnet-counters](#) to check managed memory usage
- [dotnet-dump](#) to collect and analyze a dump file
- [dotnet/diagnostics](#)

Next steps

[Debug a deadlock in .NET Core](#)

Debug a deadlock in .NET Core

9/20/2022 • 6 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 3.1 SDK and later versions

In this tutorial, you'll learn how to debug a deadlock scenario. Using the provided example [ASP.NET Core web app](#) source code repository, you can cause a deadlock intentionally. The endpoint will stop responding and experience thread accumulation. You'll learn how you can use various tools to analyze the problem, such as core dumps, core dump analysis, and process tracing.

In this tutorial, you will:

- Investigate an app that has stopped responding
- Generate a core dump file
- Analyze process threads in the dump file
- Analyze callstacks and sync blocks
- Diagnose and solve a deadlock

Prerequisites

The tutorial uses:

- [.NET Core 3.1 SDK](#) or a later version
- [Sample debug target - web app](#) to trigger the scenario
- [dotnet-trace](#) to list processes
- [dotnet-dump](#) to collect, and analyze a dump file

Core dump generation

To investigate application unresponsiveness, a core dump or memory dump allows you to inspect the state of its threads and any possible locks that may have contention issues. Run the [sample debug](#) application using the following command from the sample root directory:

```
dotnet run
```

To find the process ID, use the following command:

```
dotnet-trace ps
```

Take note of the process ID from your command output. Our process ID was `4807`, but yours will be different. Navigate to the following URL, which is an API endpoint on the sample site:

```
https://localhost:5001/api/diagscenario/deadlock
```

The API request to the site will stop responding. Let the request run for about 10-15 seconds. Then create the core dump using the following command:

- [Linux](#)
- [Windows](#)

```
sudo dotnet-dump collect -p 4807
```

Analyze the core dump

To start the core dump analysis, open the core dump using the following `dotnet-dump analyze` command. The argument is the path to the core dump file that was collected earlier.

```
dotnet-dump analyze ~/.dotnet/tools/core_20190513_143916
```

Since you're looking at a potentially unresponsive application, you want an overall feel for the thread activity in the process. You can use the `threads` command as shown below:

```
> threads
*0 0x1DBFF (121855)
 1 0x1DC01 (121857)
 2 0x1DC02 (121858)
 3 0x1DC03 (121859)
 4 0x1DC04 (121860)
 5 0x1DC05 (121861)
 6 0x1DC06 (121862)
 7 0x1DC07 (121863)
 8 0x1DC08 (121864)
 9 0x1DC09 (121865)
10 0x1DC0A (121866)
11 0x1DC0D (121869)
12 0x1DC0E (121870)
13 0x1DC10 (121872)
14 0x1DC11 (121873)
15 0x1DC12 (121874)
16 0x1DC13 (121875)
17 0x1DC14 (121876)
18 0x1DC15 (121877)
19 0x1DC1C (121884)
20 0x1DC1D (121885)
21 0x1DC1E (121886)
22 0x1DC21 (121889)
23 0x1DC22 (121890)
24 0x1DC23 (121891)
25 0x1DC24 (121892)
26 0x1DC25 (121893)
...
...
317 0x1DD48 (122184)
318 0x1DD49 (122185)
319 0x1DD4A (122186)
320 0x1DD4B (122187)
321 0x1DD4C (122188)
```

The output shows all the threads currently running in the process with their associated debugger thread ID and operating system thread ID. Based on the output, there are over 300 threads.

The next step is to get a better understanding of what the threads are currently doing by getting each thread's callstack. The `clrstack` command can be used to output callstacks. It can either output a single callstack or all the callstacks. Use the following command to output all the callstacks for all the threads in the process:

```
clrstack -all
```

A representative portion of the output looks like:

...

...

...

Child SP	IP Call Site
00007F2AE37B5680 00007f305abc6360 [GCFrame: 00007f2ae37b5680]	
00007F2AE37B5770 00007f305abc6360 [GCFrame: 00007f2ae37b5770]	
00007F2AE37B57D0 00007f305abc6360 [HelperMethodFrame_1OBJ: 00007f2ae37b57d0]	
System.Threading.Monitor.ReliableEnter(System.Object, Boolean ByRef)	
00007F2AE37B5920 00007F2FE392B31F testwebapi.Controllers.DiagScenarioController.<deadlock>b__3_1()	
[/home/marioh/webapi/Controllers/diagscenario.cs @ 36]	
00007F2AE37B5950 00007F2FE392B46D	
System.Threading.ExecutionContext.RunInternal(System.Threading.ExecutionContext,	
System.Threading.ContextCallback, System.Object)	
[/_w/3/s/src/System.Private.CoreLib/shared/System/Threading/ExecutionContext.cs @ 201]	
00007F2AE37B5CA0 00007f30593044af [GCFrame: 00007f2ae37b5ca0]	
00007F2AE37B5D70 00007f30593044af [DebuggerU2MCatchHandlerFrame: 00007f2ae37b5d70]	
OS Thread Id: 0x1dc82	
Child SP	IP Call Site
00007F2AE2FB4680 00007f305abc6360 [GCFrame: 00007f2ae2fb4680]	
00007F2AE2FB4770 00007f305abc6360 [GCFrame: 00007f2ae2fb4770]	
00007F2AE2FB47D0 00007f305abc6360 [HelperMethodFrame_1OBJ: 00007f2ae2fb47d0]	
System.Threading.Monitor.ReliableEnter(System.Object, Boolean ByRef)	
00007F2AE2FB4920 00007F2FE392B31F testwebapi.Controllers.DiagScenarioController.<deadlock>b__3_1()	
[/home/marioh/webapi/Controllers/diagscenario.cs @ 36]	
00007F2AE2FB4950 00007F2FE392B46D	
System.Threading.ExecutionContext.RunInternal(System.Threading.ExecutionContext,	
System.Threading.ContextCallback, System.Object)	
[/_w/3/s/src/System.Private.CoreLib/shared/System/Threading/ExecutionContext.cs @ 201]	
00007F2AE2FB4CA0 00007f30593044af [GCFrame: 00007f2ae2fb4ca0]	
00007F2AE2FB4D70 00007f30593044af [DebuggerU2MCatchHandlerFrame: 00007f2ae2fb4d70]	
OS Thread Id: 0x1dc83	
Child SP	IP Call Site
00007F2AE27B3680 00007f305abc6360 [GCFrame: 00007f2ae27b3680]	
00007F2AE27B3770 00007f305abc6360 [GCFrame: 00007f2ae27b3770]	
00007F2AE27B37D0 00007f305abc6360 [HelperMethodFrame_1OBJ: 00007f2ae27b37d0]	
System.Threading.Monitor.ReliableEnter(System.Object, Boolean ByRef)	
00007F2AE27B3920 00007F2FE392B31F testwebapi.Controllers.DiagScenarioController.<deadlock>b__3_1()	
[/home/marioh/webapi/Controllers/diagscenario.cs @ 36]	
00007F2AE27B3950 00007F2FE392B46D	
System.Threading.ExecutionContext.RunInternal(System.Threading.ExecutionContext,	
System.Threading.ContextCallback, System.Object)	
[/_w/3/s/src/System.Private.CoreLib/shared/System/Threading/ExecutionContext.cs @ 201]	
00007F2AE27B3CA0 00007f30593044af [GCFrame: 00007f2ae27b3ca0]	
00007F2AE27B3D70 00007f30593044af [DebuggerU2MCatchHandlerFrame: 00007f2ae27b3d70]	
OS Thread Id: 0x1dc84	
Child SP	IP Call Site
00007F2AE1FB2680 00007f305abc6360 [GCFrame: 00007f2ae1fb2680]	
00007F2AE1FB2770 00007f305abc6360 [GCFrame: 00007f2ae1fb2770]	
00007F2AE1FB27D0 00007f305abc6360 [HelperMethodFrame_1OBJ: 00007f2ae1fb27d0]	
System.Threading.Monitor.ReliableEnter(System.Object, Boolean ByRef)	
00007F2AE1FB2920 00007F2FE392B31F testwebapi.Controllers.DiagScenarioController.<deadlock>b__3_1()	
[/home/marioh/webapi/Controllers/diagscenario.cs @ 36]	
00007F2AE1FB2950 00007F2FE392B46D	
System.Threading.ExecutionContext.RunInternal(System.Threading.ExecutionContext,	
System.Threading.ContextCallback, System.Object)	
[/_w/3/s/src/System.Private.CoreLib/shared/System/Threading/ExecutionContext.cs @ 201]	
00007F2AE1FB2CA0 00007f30593044af [GCFrame: 00007f2ae1fb2ca0]	
00007F2AE1FB2D70 00007f30593044af [DebuggerU2MCatchHandlerFrame: 00007f2ae1fb2d70]	
OS Thread Id: 0x1dc85	
Child SP	IP Call Site
00007F2AE17B1680 00007f305abc6360 [GCFrame: 00007f2ae17b1680]	
00007F2AE17B1770 00007f305abc6360 [GCFrame: 00007f2ae17b1770]	
00007F2AE17B17D0 00007f305abc6360 [HelperMethodFrame_1OBJ: 00007f2ae17b17d0]	
System.Threading.Monitor.ReliableEnter(System.Object, Boolean ByRef)	
00007F2AE17B1920 00007F2FE392B31F testwebapi.Controllers.DiagScenarioController.<deadlock>b__3_1()	
[/home/marioh/webapi/Controllers/diagscenario.cs @ 36]	
00007F2AE17B1950 00007F2FE392B46D	
System.Threading.ExecutionContext.RunInternal(System.Threading.ExecutionContext,	

```

System.Threading.ContextCallback, System.Object)
[/_w/3/s/src/System.Private.CoreLib/shared/System/Threading/ExecutionContext.cs @ 201]
00007F2AE17B1CA0 00007f30593044af [GCFrame: 00007f2ae17b1ca0]
00007F2AE17B1D70 00007f30593044af [DebuggerU2MCatchHandlerFrame: 00007f2ae17b1d70]
OS Thread Id: 0x1dc86
    Child SP          IP Call Site
00007F2AE0FB0680 00007f305abc6360 [GCFrame: 00007f2ae0fb0680]
00007F2AE0FB0770 00007f305abc6360 [GCFrame: 00007f2ae0fb0770]
00007F2AE0FB07D0 00007f305abc6360 [HelperMethodFrame_1OBJ: 00007f2ae0fb07d0]
System.Threading.Monitor.ReliableEnter(System.Object, Boolean ByRef)
00007F2AE0FB0920 00007F2FE392B31F testwebapi.Controllers.DiagScenarioController.<deadlock>b__3_1()
[/home/marioh/webapi/Controllers/diagscenario.cs @ 36]
00007F2AE0FB0950 00007F2FE392B46D
System.Threading.ExecutionContext.RunInternal(System.Threading.ExecutionContext,
System.Threading.ContextCallback, System.Object)
[/_w/3/s/src/System.Private.CoreLib/shared/System/Threading/ExecutionContext.cs @ 201]
00007F2AE0FB0CA0 00007f30593044af [GCFrame: 00007f2ae0fb0ca0]
00007F2AE0FB0D70 00007f30593044af [DebuggerU2MCatchHandlerFrame: 00007f2ae0fb0d70]
OS Thread Id: 0x1dc87
    Child SP          IP Call Site
00007F2AE07AF680 00007f305abc6360 [GCFrame: 00007f2ae07af680]
00007F2AE07AF770 00007f305abc6360 [GCFrame: 00007f2ae07af770]
00007F2AE07AF7D0 00007f305abc6360 [HelperMethodFrame_1OBJ: 00007f2ae07af7d0]
System.Threading.Monitor.ReliableEnter(System.Object, Boolean ByRef)
00007F2AE07AF920 00007F2FE392B31F testwebapi.Controllers.DiagScenarioController.<deadlock>b__3_1()
[/home/marioh/webapi/Controllers/diagscenario.cs @ 36]
00007F2AE07AF950 00007F2FE392B46D
System.Threading.ExecutionContext.RunInternal(System.Threading.ExecutionContext,
System.Threading.ContextCallback, System.Object)
[/_w/3/s/src/System.Private.CoreLib/shared/System/Threading/ExecutionContext.cs @ 201]
00007F2AE07AFCA0 00007f30593044af [GCFrame: 00007f2ae07afca0]
00007F2AE07AFD70 00007f30593044af [DebuggerU2MCatchHandlerFrame: 00007f2ae07afd70]
OS Thread Id: 0x1dc88
    Child SP          IP Call Site
00007F2ADFFAE680 00007f305abc6360 [GCFrame: 00007f2adffae680]
00007F2ADFFAE770 00007f305abc6360 [GCFrame: 00007f2adffae770]
00007F2ADFFAE7D0 00007f305abc6360 [HelperMethodFrame_1OBJ: 00007f2adffae7d0]
System.Threading.Monitor.ReliableEnter(System.Object, Boolean ByRef)
00007F2ADFFAE920 00007F2FE392B31F testwebapi.Controllers.DiagScenarioController.<deadlock>b__3_1()
[/home/marioh/webapi/Controllers/diagscenario.cs @ 36]
00007F2ADFFAE950 00007F2FE392B46D
System.Threading.ExecutionContext.RunInternal(System.Threading.ExecutionContext,
System.Threading.ContextCallback, System.Object)
[/_w/3/s/src/System.Private.CoreLib/shared/System/Threading/ExecutionContext.cs @ 201]
00007F2ADFFAEC0 00007f30593044af [GCFrame: 00007f2adffaeca0]
00007F2ADFFAED70 00007f30593044af [DebuggerU2MCatchHandlerFrame: 00007f2adffaed70]
...
...

```

Observing the callstacks for all 300+ threads shows a pattern where a majority of the threads share a common callstack:

```

OS Thread Id: 0x1dc88
    Child SP          IP Call Site
00007F2ADFFAE680 00007f305abc6360 [GCFrame: 00007f2adffae680]
00007F2ADFFAE770 00007f305abc6360 [GCFrame: 00007f2adffae770]
00007F2ADFFAE7D0 00007f305abc6360 [HelperMethodFrame_10B]: 00007f2adffae7d0]
System.Threading.Monitor.ReliableEnter(System.Object, Boolean ByRef)
00007F2ADFFAE920 00007F2FE392B31F testwebapi.Controllers.DiagScenarioController.<deadlock>b__3_1()
[/home/marioh/webapi/Controllers/diagscenario.cs @ 36]
00007F2ADFFAE950 00007F2FE392B46D
System.Threading.ExecutionContext.RunInternal(System.Threading.ExecutionContext,
System.Threading.ContextCallback, System.Object)
[/_w/3/s/src/System.Private.CoreLib/shared/System/Threading/ExecutionContext.cs @ 201]
00007F2ADFFAECA0 00007f30593044af [GCFrame: 00007f2adffaec0]
00007F2ADFFAED70 00007f30593044af [DebuggerU2MCatchHandlerFrame: 00007f2adffaed70]

```

The callstack seems to show that the request arrived in our deadlock method that in turn makes a call to `Monitor.ReliableEnter`. This method indicates that the threads are trying to enter a monitor lock. They're waiting on the availability of the lock. It's likely locked by a different thread.

The next step then is to find out which thread is actually holding the monitor lock. Since monitors typically store lock information in the sync block table, we can use the `syncblk` command to get more information:

```

> syncblk
Index      SyncBlock MonitorHeld Recursion Owning Thread Info      SyncBlock Owner
43 00000246E51268B8      603      1 0000024B713F4E30 5634 28  00000249654b14c0 System.Object
44 00000246E5126908      3      1 0000024B713F47E0 51d4 29  00000249654b14d8 System.Object
-----
Total      344
CCW       1
RCW       2
ComClassFactory 0
Free      0

```

The two interesting columns are **MonitorHeld** and **Owning Thread Info**. The **MonitorHeld** column shows whether a monitor lock is acquired by a thread and the number of waiting threads. The **Owning Thread Info** column shows which thread currently owns the monitor lock. The thread info has three different subcolumns. The second subcolumn shows operating system thread ID.

At this point, we know two different threads (0x5634 and 0x51d4) hold a monitor lock. The next step is to take a look at what those threads are doing. We need to check if they're stuck indefinitely holding the lock. Let's use the `setthread` and `clrstack` commands to switch to each of the threads and display the callstacks.

To look at the first thread, run the `setthread` command, and find the index of the 0x5634 thread (our index was 28). The deadlock function is waiting to acquire a lock, but it already owns the lock. It's in deadlock waiting for the lock it already holds.

```
> setthread 28
> clrstack
OS Thread Id: 0x5634 (28)
      Child SP          IP Call Site
0000004E46AFEEA8 00007fff43a5cbc4 [GCFrame: 0000004e46afeaa8]
0000004E46AFEC18 00007fff43a5cbc4 [GCFrame: 0000004e46afec18]
0000004E46AFEC68 00007fff43a5cbc4 [HelperMethodFrame_1OBJ: 0000004e46afec68]
System.Threading.Monitor.Enter(System.Object)
0000004E46AFEDC0 00007FFE5EAFC80 testwebapi.Controllers.DiagScenarioController.DeadlockFunc()
[C:\Users\dapine\Downloads\Diagnostic_scenarios_sample_debug_target\Controllers\DiagnosticScenarios.cs @ 58]
0000004E46AFEE30 00007FFE5EAFC8C testwebapi.Controllers.DiagScenarioController.<deadlock>b__3_0()
[C:\Users\dapine\Downloads\Diagnostic_scenarios_sample_debug_target\Controllers\DiagnosticScenarios.cs @ 26]
0000004E46AFEE80 00007FFEBB251A5B System.Threading.ThreadHelper.ThreadStart_Context(System.Object)
[/_/src/System.Private.CoreLib/src/System/Threading/Thread.CoreCLR.cs @ 44]
0000004E46AFEEB0 00007FFE5EAEEEC
System.Threading.ExecutionContext.RunInternal(System.Threading.ExecutionContext,
System.Threading.ContextCallback, System.Object)
[/_/src/System.Private.CoreLib/shared/System/Threading/ExecutionContext.cs @ 201]
0000004E46AFEF20 00007FFEBB234EAB System.Threading.ThreadHelper.ThreadStart()
[/_/src/System.Private.CoreLib/src/System/Threading/Thread.CoreCLR.cs @ 93]
0000004E46AFF138 00007ffebdcc6b63 [GCFrame: 0000004e46aff138]
0000004E46AFF3A0 00007ffebdcc6b63 [DebuggerU2MCatchHandlerFrame: 0000004e46aff3a0]
```

The second thread is similar. It's also trying to acquire a lock that it already owns. The remaining 300+ threads that are all waiting are most likely also waiting on one of the locks that caused the deadlock.

See also

- [dotnet-trace](#) to list processes
- [dotnet-counters](#) to check managed memory usage
- [dotnet-dump](#) to collect and analyze a dump file
- [dotnet/diagnostics](#)

Next steps

[What diagnostic tools are available in .NET Core](#)

Debug ThreadPool Starvation

9/20/2022 • 9 minutes to read • [Edit Online](#)

This article applies to: ✓ .NET Core 3.1 and later versions

In this tutorial, you'll learn how to debug a ThreadPool starvation scenario. ThreadPool starvation occurs when the pool has no available threads to process new work items and it often causes applications to respond slowly. Using the provided example [ASP.NET Core web app](#), you can cause ThreadPool starvation intentionally and learn how to diagnose it.

In this tutorial, you will:

- Investigate an app that is responding to requests slowly
- Use the `dotnet-counters` tool to identify ThreadPool starvation is likely occurring
- Use the `dotnet-stack` tool to determine what work is keeping the ThreadPool threads busy

Prerequisites

The tutorial uses:

- [.NET Core 6.0 SDK](#) to build and run the sample app
- [Sample web app](#) to demonstrate ThreadPool starvation behavior
- [Bombardier](#) to generate load for the sample web app
- [dotnet-counters](#) to observe performance counters
- [dotnet-stack](#) to examine thread stacks

Running the sample app

1. Download the code for the [sample app](#) and build it using the .NET SDK:

```
E:\demo\DiagnosticScenarios>dotnet build
Microsoft (R) Build Engine version 17.1.1+a02f73656 for .NET
Copyright (C) Microsoft Corporation. All rights reserved.

Determining projects to restore...
All projects are up-to-date for restore.
DiagnosticScenarios -> E:\demo\DiagnosticScenarios\bin\Debug\net6.0\DiagnosticScenarios.dll

Build succeeded.
  0 Warning(s)
  0 Error(s)

Time Elapsed 00:00:01.26
```

2. Run the app:

```
E:\demo\DiagnosticScenarios>bin\Debug\net6.0\DiagnosticScenarios.exe
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://localhost:5000
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: https://localhost:5001
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Production
info: Microsoft.Hosting.Lifetime[0]
      Content root path: E:\demo\DiagnosticScenarios
```

If you use a web browser and send requests to `https://localhost:5001/api/diagscenario/taskwait`, you should see the response `success:taskwait` returned after about 500 ms. This shows that the web server is serving traffic as expected.

Observing slow performance

The demo web server has several endpoints which mock doing a database request and then returning a response to the user. Each of these endpoints has a delay of approximately 500 ms when serving requests one at a time but the performance is much worse when the web server is subjected to some load. Download the [Bombardier](#) load testing tool and observe the difference in latency when 125 concurrent requests are sent to each endpoint.

```
bombardier-windows-amd64.exe https://localhost:5001/api/diagscenario/taskwait
Bombarding https://localhost:5001/api/diagscenario/taskwait for 10s using 125 connection(s)
[=====] 10s
Done!
Statistics      Avg       Stdev       Max
  Req/sec      33.06     234.67    3313.54
  Latency       3.48s     1.39s     10.79s
HTTP codes:
  1xx - 0, 2xx - 454, 3xx - 0, 4xx - 0, 5xx - 0
  others - 0
Throughput:    75.37KB/s
```

This second endpoint uses a code pattern that performs even worse:

```
bombardier-windows-amd64.exe https://localhost:5001/api/diagscenario/tasksleepwait
Bombarding https://localhost:5001/api/diagscenario/tasksleepwait for 10s using 125 connection(s)
[=====] 10s
Done!
Statistics      Avg       Stdev       Max
  Req/sec      1.61      35.25    788.91
  Latency      15.42s    2.18s     18.30s
HTTP codes:
  1xx - 0, 2xx - 140, 3xx - 0, 4xx - 0, 5xx - 0
  others - 0
Throughput:    36.57KB/s
```

Both of these endpoints show dramatically more than the 500-ms average latency when load is high (3.48 s and 15.42 s respectively). If you run this example on an older version of .NET Core, you're likely to see both examples perform equally badly. .NET 6 has updated ThreadPool heuristics that reduce the performance impact of the bad coding pattern used in the first example.

Detecting ThreadPool starvation

If you observed the behavior above on a real world service, you would know it's responding slowly under load

but you wouldn't know the cause. [dotnet-counters](#) is a tool that can show live performance counters. These counters can provide clues about certain problems and are often easy to get. In production environments, you might have similar counters provided by remote monitoring tools and web dashboards. Install dotnet-counters and begin monitoring the web service:

```
dotnet-counters monitor -n DiagnosticScenarios
Press p to pause, r to resume, q to quit.
Status: Running

[System.Runtime]
% Time in GC since last GC (%) 0
Allocation Rate (B / 1 sec) 0
CPU Usage (%) 0
Exception Count (Count / 1 sec) 0
GC Committed Bytes (MB) 0
GC Fragmentation (%) 0
GC Heap Size (MB) 34
Gen 0 GC Count (Count / 1 sec) 0
Gen 0 Size (B) 0
Gen 1 GC Count (Count / 1 sec) 0
Gen 1 Size (B) 0
Gen 2 GC Count (Count / 1 sec) 0
Gen 2 Size (B) 0
IL Bytes Jitted (B) 279,021
LOH Size (B) 0
Monitor Lock Contention Count (Count / 1 sec) 0
Number of Active Timers 0
Number of Assemblies Loaded 121
Number of Methods Jitted 3,223
POH (Pinned Object Heap) Size (B) 0
ThreadPool Completed Work Item Count (Count / 1 sec) 0
ThreadPool Queue Length 0
ThreadPool Thread Count 1
Time spent in JIT (ms / 1 sec) 0.387
Working Set (MB) 87
```

The counters above are an example while the web server wasn't serving any requests. Run Bombardier again with the `api/diagscenario/tasksleepwait` endpoint and sustained load for 2 minutes so there's plenty of time to observe what happens to the performance counters.

```
bombardier-windows-amd64.exe https://localhost:5001/api/diagscenario/tasksleepwait -d 120s
```

ThreadPool starvation occurs when there are no free threads to handle the queued work items and the runtime responds by increasing the number of ThreadPool threads. You should observe the `ThreadPool Thread Count` rapidly increase to 2-3x the number of processor cores on your machine and then further threads are added 1-2 per second until eventually stabilizing somewhere above 125. The slow and steady increase of ThreadPool threads combined with CPU Usage much less than 100% are the key signals that ThreadPool starvation is currently a performance bottleneck. The thread count increase will continue until either the pool hits the maximum number of threads, enough threads have been created to satisfy all the incoming work items, or the CPU has been saturated. Often, but not always, ThreadPool starvation will also show large values for `ThreadPool Queue Length` and low values for `ThreadPool Completed Work Item Count`, meaning that there's a large amount of pending work and little work being completed. Here's an example of the counters while the thread count is still rising:

```
Press p to pause, r to resume, q to quit.
```

```
Status: Running
```

[System.Runtime]	
% Time in GC since last GC (%)	0
Allocation Rate (B / 1 sec)	24,480
CPU Usage (%)	0
Exception Count (Count / 1 sec)	0
GC Committed Bytes (MB)	56
GC Fragmentation (%)	40.603
GC Heap Size (MB)	89
Gen 0 GC Count (Count / 1 sec)	0
Gen 0 Size (B)	6,306,160
Gen 1 GC Count (Count / 1 sec)	0
Gen 1 Size (B)	8,061,400
Gen 2 GC Count (Count / 1 sec)	0
Gen 2 Size (B)	192
IL Bytes Jitted (B)	279,263
LOH Size (B)	98,576
Monitor Lock Contention Count (Count / 1 sec)	0
Number of Active Timers	124
Number of Assemblies Loaded	121
Number of Methods Jitted	3,227
POH (Pinned Object Heap) Size (B)	1,197,336
ThreadPool Completed Work Item Count (Count / 1 sec)	2
ThreadPool Queue Length	29
ThreadPool Thread Count	96
Time spent in JIT (ms / 1 sec)	0
Working Set (MB)	152

Once the count of ThreadPool threads stabilizes, the pool is no longer starving. But if it stabilizes at a high value (more than about three times the number of processor cores), that usually indicates the application code is blocking some ThreadPool threads and the ThreadPool is compensating by running with more threads. Running steady at high thread counts won't necessarily have large impacts on request latency, but if load varies dramatically over time or the app will be periodically restarted, then each time the ThreadPool is likely to enter a period of starvation where it's slowly increasing threads and delivering poor request latency. Each thread also consumes memory, so reducing the total number of threads needed provides another benefit.

Starting in .NET 6, ThreadPool heuristics were modified to scale up the number of ThreadPool threads much faster in response to certain blocking Task APIs. ThreadPool starvation can still occur with these APIs, but the duration is much briefer than it was with older .NET versions because the runtime responds more quickly. Run Bombardier again with the `api/diagscenario/taskwait` endpoint:

```
bombardier-windows-amd64.exe https://localhost:5001/api/diagscenario/taskwait -d 120s
```

On .NET 6 you should observe the pool increase the thread count more quickly than before and then stabilize at a high number of threads. ThreadPool starvation is occurring while the thread count is climbing.

Resolving ThreadPool starvation

To eliminate ThreadPool starvation, ThreadPool threads need to remain unblocked so that they're available to handle incoming work items. There are two ways to determine what each thread was doing, either using the `dotnet-stack` tool or capturing a dump with `dotnet-dump` that can be viewed in [Visual Studio](#). `dotnet-stack` can be faster because it shows the thread stacks immediately on the console, but Visual Studio dump debugging offers better visualizations that map frames to source, Just My Code can filter out runtime implementation frames, and the Parallel Stacks feature can help group large numbers of threads with similar stacks. This tutorial shows the `dotnet-stack` option. See the [diagnosing ThreadPool starvation tutorial video](#) for an example of investigating the thread stacks using Visual Studio.

Run Bombardier again to put the web server under load:

```
bombardier-windows-amd64.exe https://localhost:5001/api/diagscenario/taskwait -d 120s
```

Then run dotnet-stack to see the thread stack traces:

```
dotnet-stack report -n DiagnosticScenarios
```

You should see a long output containing a large number of stacks, many of which look like this:

```
Thread (0x25968):
[Native Frames]
System.Private.CoreLib.il!System.Threading.ManualResetEventSlim.Wait(int32,value class
System.Threading.CancellationToken)
System.Private.CoreLib.il!System.Threading.Tasks.Task.SpinThenBlockingWait(int32,value class
System.Threading.CancellationToken)
System.Private.CoreLib.il!System.Threading.Tasks.Task.InternalWaitCore(int32,value class
System.Threading.CancellationToken)
System.Private.CoreLib.il!System.Threading.Tasks.Task`1[System.__Canon].GetResultCore(bool)
DiagnosticScenarios!testwebapi.Controllers.DiagScenarioController.TaskWait()
Anonymously Hosted DynamicMethods Assembly!dynamicClass.lambda_method1(pMT: 00007FF7A8CBF658,class
System.Object,class System.Object[])
Microsoft.AspNetCore.Mvc.Core.il!Microsoft.AspNetCore.Mvc.Infrastructure.ActionMethodExecutor+SyncObjectResu
ltExecutor.Execute(class Microsoft.AspNetCore.Mvc.Infrastructure.IActionResultTypeMapper,class
Microsoft.Extensions.Internal.ObjectMethodExecutor,class System.Object,class System.Object[])
Microsoft.AspNetCore.Mvc.Core.il!Microsoft.AspNetCore.Mvc.Infrastructure.ControllerActionInvoker.InvokeActio
nMethodAsync()

Microsoft.AspNetCore.Mvc.Core.il!Microsoft.AspNetCore.Mvc.Infrastructure.ControllerActionInvoker.Next(value
class State&,value class Scope&,class System.Object&,bool&)

Microsoft.AspNetCore.Mvc.Core.il!Microsoft.AspNetCore.Mvc.Infrastructure.ControllerActionInvoker.InvokeNextA
ctionFilterAsync()

Microsoft.AspNetCore.Mvc.Core.il!Microsoft.AspNetCore.Mvc.Infrastructure.ControllerActionInvoker.Next(value
class State&,value class Scope&,class System.Object&,bool&)

Microsoft.AspNetCore.Mvc.Core.il!Microsoft.AspNetCore.Mvc.Infrastructure.ControllerActionInvoker.InvokeNextA
ctionFilterAsync()

Microsoft.AspNetCore.Mvc.Core.il!Microsoft.AspNetCore.Mvc.Infrastructure.ControllerActionInvoker.Next(value
class State&,value class Scope&,class System.Object&,bool&)

Microsoft.AspNetCore.Mvc.Core.il!Microsoft.AspNetCore.Mvc.Infrastructure.ControllerActionInvoker.InvokeInner
FilterAsync()
    Microsoft.AspNetCore.Mvc.Core.il!Microsoft.AspNetCore.Mvc.Infrastructure.ResourceInvoker.Next(value class
State&,value class Scope&,class System.Object&,bool&)

Microsoft.AspNetCore.Mvc.Core.il!Microsoft.AspNetCore.Mvc.Infrastructure.ResourceInvoker.InvokeFilterPipelin
eAsync()
    Microsoft.AspNetCore.Mvc.Core.il!Microsoft.AspNetCore.Mvc.Infrastructure.ResourceInvoker.InvokeAsync()
    Microsoft.AspNetCore.Mvc.Core.il!Microsoft.AspNetCore.Mvc.Routing.ControllerRequestDelegateFactory+
<>c__DisplayClass10_0.<CreateRequestDelegate>b__0(class Microsoft.AspNetCore.Http.HttpContext)
    Microsoft.AspNetCore.Routing.il!Microsoft.AspNetCore.Routing.EndpointMiddleware.Invoke(class
Microsoft.AspNetCore.Http.HttpContext)
    Microsoft.AspNetCore.Authorization.Policy.il!Microsoft.AspNetCore.Authorization.AuthorizationMiddleware+
<Invoke>d__6.MoveNext()
        System.Private.CoreLib.il!System.Runtime.CompilerServices.AsyncMethodBuilderCore.Start(!!0&)

Microsoft.AspNetCore.Authorization.Policy.il!Microsoft.AspNetCore.Authorization.AuthorizationMiddleware.Invo
ke(class Microsoft.AspNetCore.Http.HttpContext)
```

```

Microsoft.AspNetCore.HttpsPolicy.il!Microsoft.AspNetCore.HttpsPolicy.HttpsRedirectionMiddleware.Invoke(class
Microsoft.AspNetCore.Http.HttpContext)
    Microsoft.AspNetCore.HttpsPolicy.il!Microsoft.AspNetCore.HttpsPolicy.HstsMiddleware.Invoke(class
Microsoft.AspNetCore.Http.HttpContext)

Microsoft.AspNetCore.HostFiltering.il!Microsoft.AspNetCore.HostFiltering.HostFilteringMiddleware.Invoke(clas
s Microsoft.AspNetCore.Http.HttpContext)

Microsoft.AspNetCore.Server.Kestrel.Core.il!Microsoft.AspNetCore.Server.Kestrel.Core.Internal.Http.HttpProto
col+<ProcessRequests>d__223`1[System._Canon].MoveNext()
    System.Private.CoreLib.il!System.Threading.ExecutionContext.RunInternal(class
System.Threading.ExecutionContext, class System.Threading.ContextCallback, class System.Object)

System.Private.CoreLib.il!System.Runtime.CompilerServices.AsyncTaskMethodBuilder`1+AsyncStateMachineBox`1[Sy
stem.Threading.Tasks.VoidTaskResult, Microsoft.AspNetCore.Server.Kestrel.Core.Internal.Http.HttpProtocol+
<ProcessRequests>d__223`1[System._Canon]].MoveNext(class System.Threading.Thread)
    System.Private.CoreLib.il!System.Threading.Tasks.AwaitTaskContinuation.RunOrScheduleAction(class
System.Runtime.CompilerServices.IAsyncStateMachineBox, bool)
    System.Private.CoreLib.il!System.Threading.Tasks.Task.RunContinuations(class System.Object)
    System.IO.Pipelines.il!System.IO.Pipelines.StreamPipeReader+<<ReadAsync>g__Core|36_0>d.MoveNext()
    System.Private.CoreLib.il!System.Threading.ExecutionContext.RunInternal(class
System.Threading.ExecutionContext, class System.Threading.ContextCallback, class System.Object)

System.Private.CoreLib.il!System.Runtime.CompilerServices.AsyncTaskMethodBuilder`1+AsyncStateMachineBox`1[Sy
stem.IO.Pipelines.ReadResult, System.IO.Pipelines.StreamPipeReader+<<ReadAsync>g__Core|36_0>d].MoveNext(class
System.Threading.Thread)
    System.Private.CoreLib.il!System.Threading.Tasks.AwaitTaskContinuation.RunOrScheduleAction(class
System.Runtime.CompilerServices.IAsyncStateMachineBox, bool)
    System.Private.CoreLib.il!System.Threading.Tasks.Task.RunContinuations(class System.Object)

System.Private.CoreLib.il!System.Runtime.CompilerServices.AsyncTaskMethodBuilder`1[System.Int32].SetExisting
TaskResult(class System.Threading.Tasks.Task`1<!0>, !0)
    System.Net.Security.il!System.Net.Security.SslStream+
<ReadAsyncInternal>d__186`1[System.Net.Security.AsyncReadWriteAdapter].MoveNext()
    System.Private.CoreLib.il!System.Threading.ExecutionContext.RunInternal(class
System.Threading.ExecutionContext, class System.Threading.ContextCallback, class System.Object)

System.Private.CoreLib.il!System.Runtime.CompilerServices.AsyncTaskMethodBuilder`1+AsyncStateMachineBox`1[Sy
stem.Int32, System.Net.Security.SslStream+
<ReadAsyncInternal>d__186`1[System.Net.Security.AsyncReadWriteAdapter]].MoveNext(class
System.Threading.Thread)

Microsoft.AspNetCore.Server.Kestrel.Core.il!Microsoft.AspNetCore.Server.Kestrel.Core.Internal.DuplexPipeStre
am+<ReadAsyncInternal>d__27.MoveNext()
    System.Private.CoreLib.il!System.Threading.ExecutionContext.RunInternal(class
System.Threading.ExecutionContext, class System.Threading.ContextCallback, class System.Object)
    System.Private.CoreLib.il!System.Threading.ThreadPoolWorkQueue.Dispatch()
    System.Private.CoreLib.il!System.Threading.PortableThreadPool+WorkerThread.WorkerThreadStart()

```

The frames at the bottom of these stacks indicate that these threads are ThreadPool threads:

```

System.Private.CoreLib.il!System.Threading.ThreadPoolWorkQueue.Dispatch()
System.Private.CoreLib.il!System.Threading.PortableThreadPool+WorkerThread.WorkerThreadStart()

```

And the frames near the top reveal that the thread is blocked on a call to `GetResultCore(bool)` from the `DiagnosticScenarioController.TaskWait()` function:

```
Thread (0x25968):
[Native Frames]
System.Private.CoreLib.il!System.Threading.ManualResetEventSlim.Wait(int32,value class
System.Threading.CancellationToken)
System.Private.CoreLib.il!System.Threading.Tasks.Task.SpinThenBlockingWait(int32,value class
System.Threading.CancellationToken)
System.Private.CoreLib.il!System.Threading.Tasks.Task.InternalWaitCore(int32,value class
System.Threading.CancellationToken)
System.Private.CoreLib.il!System.Threading.Tasks.Task`1[System.__Canon].GetResultCore(bool)
DiagnosticScenarios!testwebapi.Controllers.DiagScenarioController.TaskWait()
```

Now you can navigate to the code for this controller in the sample app's *Controllers/DiagnosticScenarios.cs* file to see that it's calling an async API without using `await`. This is the [sync-over-async](#) code pattern, which is known to block threads and is the most common cause of ThreadPool starvation.

```
public ActionResult<string> TaskWait()
{
    // ...
    Customer c = PretendQueryCustomerFromDbAsync("Dana").Result;
    return "success:taskwait";
}
```

In this case the code can be readily changed to use the `async/await` instead as shown in the [TaskAsyncWait\(\)](#) endpoint. Using `await` allows the current thread to service other workitems while the database query is in progress. When the database lookup is complete a ThreadPool thread will resume execution. This way no thread is blocked in the code during each request:

```
public async Task<ActionResult<string>> TaskAsyncWait()
{
    // ...
    Customer c = await PretendQueryCustomerFromDbAsync("Dana");
    return "success:taskasyncwait";
}
```

Running Bombadier to send load to the [api/diagscenario/taskasyncwait](#) endpoint shows that the ThreadPool thread count stays much lower and average latency remains near 500ms when using the `async/await` approach:

```
>bombardier-windows-amd64.exe https://localhost:5001/api/diagscenario/taskasyncwait
Bombarding https://localhost:5001/api/diagscenario/taskasyncwait for 10s using 125 connection(s)
[=====] 10s
Done!
Statistics          Avg      Stdev      Max
  Req/sec       227.92     274.27   1263.48
  Latency       532.58ms    58.64ms    1.14s
HTTP codes:
  1xx - 0, 2xx - 2390, 3xx - 0, 4xx - 0, 5xx - 0
  others - 0
Throughput:    98.81KB/s
```

Debug StackOverflow errors

9/20/2022 • 2 minutes to read • [Edit Online](#)

A [StackOverflowException](#) is thrown when the execution stack overflows because it contains too many nested method calls.

For example, suppose you have an app as follows:

```
using System;

namespace temp
{
    class Program
    {
        static void Main(string[] args)
        {
            Main(args); // Oops, this recursion won't stop.
        }
    }
}
```

The `Main` method will continuously call itself until there is no more stack space. Once there is no more stack space, execution cannot continue and so it will throw a [StackOverflowException](#).

```
> dotnet run
Stack overflow.
```

NOTE

On .NET 5 and later, the callstack is output to the console.

NOTE

This article describes how to debug a stack overflow with lldb. If you are running on Windows, we suggest debugging the app with [Visual Studio](#) or [Visual Studio Code](#).

Example

1. Run the app with it configured to collect a dump on crash

```
> export DOTNET_DbEnableMiniDump=1
> dotnet run
Stack overflow.
Writing minidump with heap to file /tmp/coredump.6412
Written 58191872 bytes (14207 pages) to core file
```

NOTE

.NET 6 standardizes on the prefix `DOTNET_` instead of `COMPlus_` for environment variables that configure .NET run-time behavior. However, the `COMPlus_` prefix will continue to work. If you're using a previous version of the .NET runtime, you should still use the `COMPlus_` prefix for environment variables.

2. Install the SOS extension using `dotnet-sos`

```
dotnet-sos install
```

3. Debug the dump in lldb to see the failing stack

```
lldb --core /temp/coredump.6412
(lldb) bt
...
frame #261930: 0x00007f59b40900cc
frame #261931: 0x00007f59b40900cc
frame #261932: 0x00007f59b40900cc
frame #261933: 0x00007f59b40900cc
frame #261934: 0x00007f59b40900cc
frame #261935: 0x00007f5a2d4a080f libcoreclr.so`CallDescrWorkerInternal at
unixasmmacrosamd64.inc:867
    frame #261936: 0x00007f5a2d3cc4c3 libcoreclr.so`MethodDescCallSite::CallTargetWorker(unsigned
long const*, unsigned long*, int) at callhelpers.cpp:70
    frame #261937: 0x00007f5a2d3cc468 libcoreclr.so`MethodDescCallSite::CallTargetWorker(this=
<unavailable>, pArguments=0x00007ffe8222e7b0, pReturnValue=0x0000000000000000, cbReturnValue=0) at
callhelpers.cpp:604
    frame #261938: 0x00007f5a2d4b6182 libcoreclr.so`RunMain(MethodDesc*, short, int*, PtrArray**)
[inlined] MethodDescCallSite::Call(this=<unavailable>, pArguments=<unavailable>) at callhelpers.h:468
...
...
```

4. The top frame `0x00007f59b40900cc` is repeated several times. Use the `SOS ip2md` command to figure out what method is located at the `0x00007f59b40900cc` address

```
(lldb) ip2md 0x00007f59b40900cc
MethodDesc: 00007f59b413ffa8
Method Name: temp.Program.Main(System.String[])
Class: 00007f59b4181d40
MethodTable: 00007f59b4190020
mdToken: 0000000006000001
Module: 00007f59b413dbf8
IsJitted: yes
Current CodeAddr: 00007f59b40900a0
Version History:
    ILCodeVersion: 0000000000000000
    ReJIT ID: 0
    IL Addr: 0000000000000000
        CodeAddr: 00007f59b40900a0 (MinOptJitted)
        NativeCodeVersion: 0000000000000000
Source file: /temp/Program.cs @ 9
```

5. Go look at the indicated method `temp.Program.Main(System.String[])` and source "`/temp/Program.cs @ 9`" to see if you can figure out what you did wrong. If it still wasn't clear you could add logging in that area of the code.

See Also

- [An introduction to dumps in .NET](#)

- Debug Linux dumps
- SOS Debugging Extension for .NET

Overview of .NET source code analysis

9/20/2022 • 7 minutes to read • [Edit Online](#)

.NET compiler platform (Roslyn) analyzers inspect your C# or Visual Basic code for code quality and style issues. Starting in .NET 5, these analyzers are included with the .NET SDK and you don't need to install them separately. If your project targets .NET 5 or later, code analysis is enabled by default. If your project targets a different .NET implementation, for example, .NET Core, .NET Standard, or .NET Framework, you must manually enable code analysis by setting the `EnableNETAnalyzers` property to `true`.

If you don't want to move to the .NET 5+ SDK, have a non-SDK-style .NET Framework project, or prefer a NuGet package-based model, the analyzers are also available in the [Microsoft.CodeAnalysis.NetAnalyzers NuGet package](#). You might prefer a package-based model for on-demand version updates.

NOTE

.NET analyzers are target-framework agnostic. That is, your project does not need to target a specific .NET implementation. The analyzers work for projects that target .NET 5+ as well as earlier .NET versions, such as .NET Core 3.1 and .NET Framework 4.7.2. However, to enable code analysis using the `EnableNETAnalyzers` property, your project must reference a [project SDK](#).

If rule violations are found by an analyzer, they're reported as a suggestion, warning, or error, depending on how each rule is [configured](#). Code analysis violations appear with the prefix "CA" or "IDE" to differentiate them from compiler errors.

Code quality analysis

Code quality analysis ("CAxxxx") rules inspect your C# or Visual Basic code for security, performance, design and other issues. Analysis is enabled, by default, for projects that target .NET 5 or later. You can enable code analysis on projects that target earlier .NET versions by setting the `EnableNETAnalyzers` property to `true`. You can also disable code analysis for your project by setting `EnableNETAnalyzers` to `false`.

TIP

If you're using Visual Studio, many analyzer rules have associated *code fixes* that you can apply to correct the problem. Code fixes are shown in the light bulb icon menu.

Enabled rules

The following rules are enabled, by default, in .NET 6.

DIAGNOSTIC ID	CATEGORY	SEVERITY	DESCRIPTION
CA1416	Interoperability	Warning	Platform compatibility analyzer
CA1417	Interoperability	Warning	Do not use <code>OutAttribute</code> on string parameters for P/Invokes
CA1418	Interoperability	Warning	Use valid platform string
CA1831	Performance	Warning	Use <code>AsSpan</code> instead of range-based indexers for string when appropriate

DIAGNOSTIC ID	CATEGORY	SEVERITY	DESCRIPTION
CA2013	Reliability	Warning	Do not use <code>ReferenceEquals</code> with value types
CA2014	Reliability	Warning	Do not use <code>stackalloc</code> in loops
CA2015	Reliability	Warning	Do not define finalizers for types derived from <code>MemoryManager<T></code>
CA2017	Reliability	Warning	Parameter count mismatch
CA2018	Reliability	Warning	The <code>count</code> argument to <code>Buffer.BlockCopy</code> should specify the number of bytes to copy
CA2200	Usage	Warning	Rethrow to preserve stack details
CA2252	Usage	Error	Opt in to preview features
CA2247	Usage	Warning	Argument passed to <code>TaskCompletionSource</code> constructor should be <code>TaskCreationOptions</code> enum instead of <code>TaskContinuationOptions</code>
CA2255	Usage	Warning	The <code>ModuleInitializer</code> attribute should not be used in libraries
CA2256	Usage	Warning	All members declared in parent interfaces must have an implementation in a <code>DynamicInterfaceCastableImplementation</code> -attributed interface
CA2257	Usage	Warning	Members defined on an interface with the <code>DynamicInterfaceCastableImplementationAttribute</code> should be <code>static</code>
CA2258	Usage	Warning	Providing a <code>DynamicInterfaceCastableImplementation</code> interface in Visual Basic is unsupported

You can change the severity of these rules to disable them or elevate them to errors. You can also [enable more rules](#).

- For a list of rules that are included with each .NET SDK version, see [Analyzer releases](#).
- For a list of all the code quality rules, see [Code quality rules](#).

Enable additional rules

Analysis mode refers to a predefined code analysis configuration where none, some, or all rules are enabled. In the default analysis mode, only a small number of rules are [enabled as build warnings](#). You can change the analysis mode for your project by setting the `<AnalysisMode>` property in the project file. The allowable values are:

- None
- Default
- Minimum
- Recommended
- All

Starting in .NET 6, you can omit `<AnalysisMode>` in favor of a compound value for the `<AnalysisLevel>` property. For example, the following value enables the recommended set of rules for the latest release:

`<AnalysisLevel>latest-Recommended</AnalysisLevel>`. For more information, see [AnalysisLevel](#).

To find the default severity for each available rule and whether or not the rule is enabled in the default analysis mode, see the [full list of rules](#).

Treat warnings as errors

If you use the `-warnaserror` flag when you build your projects, all code analysis warnings are also treated as errors. If you do not want code quality warnings (CAxxxx) to be treated as errors in presence of `-warnaserror`, you can set the `CodeAnalysisTreatWarningsAsErrors` MSBuild property to `false` in your project file.

```
<PropertyGroup>
  <CodeAnalysisTreatWarningsAsErrors>false</CodeAnalysisTreatWarningsAsErrors>
</PropertyGroup>
```

You'll still see any code analysis warnings, but they won't break your build.

Latest updates

By default, you'll get the latest code analysis rules and default rule severities as you upgrade to newer versions of the .NET SDK. If you don't want this behavior, for example, if you want to ensure that no new rules are enabled or disabled, you can override it in one of the following ways:

- Set the `AnalysisLevel` MSBuild property to a specific value to lock the warnings to that set. When you upgrade to a newer SDK, you'll still get bug fixes for those warnings, but no new warnings will be enabled and no existing warnings will be disabled. For example, to lock the set of rules to those that ship with version 5.0 of the .NET SDK, add the following entry to your project file.

```
<PropertyGroup>
  <AnalysisLevel>5.0</AnalysisLevel>
</PropertyGroup>
```

TIP

The default value for the `AnalysisLevel` property is `latest`, which means you always get the latest code analysis rules as you move to newer versions of the .NET SDK.

For more information, and to see a list of possible values, see [AnalysisLevel](#).

- Install the [Microsoft.CodeAnalysis.NetAnalyzers NuGet package](#) to decouple rule updates from .NET SDK updates. For projects that target .NET 5+, installing the package turns off the built-in SDK analyzers. You'll get a build warning if the SDK contains a newer analyzer assembly version than that of the NuGet package. To disable the warning, set the `_SkipUpgradeNetAnalyzersNuGetWarning` property to `true`.

NOTE

If you install the Microsoft.CodeAnalysis.NetAnalyzers NuGet package, you should not add the `EnableNETAnalyzers` property to either your project file or a `Directory.Build.props` file. When the NuGet package is installed and the `EnableNETAnalyzers` property is set to `true`, a build warning is generated.

Code-style analysis

Code-style analysis ("IDExxxx") rules enable you to define and maintain consistent code style in your codebase. The default enablement settings are:

- Command-line build: Code-style analysis is disabled, by default, for all .NET projects on command-line builds.

Starting in .NET 5, you can [enable code-style analysis on build](#), both at the command line and inside Visual Studio. Code style violations appear as warnings or errors with an "IDE" prefix. This enables you to enforce consistent code styles at build time.
- Visual Studio: Code-style analysis is enabled, by default, for all .NET projects inside Visual Studio as [code refactoring quick actions](#).

For a full list of code-style analysis rules, see [Code style rules](#).

Enable on build

With the .NET 5 SDK and later versions, you can enable code-style analysis when building from the command-line and in Visual Studio. (However, for performance reasons, [a handful of code-style rules](#) will still apply only in the Visual Studio IDE.)

Follow these steps to enable code-style analysis on build:

1. Set the MSBuild property `EnforceCodeStyleInBuild` to `true`.
2. In an `.editorconfig` file, [configure](#) each "IDE" code style rule that you wish to run on build as a warning or an error. For example:

```
[*.{cs,vb}]
# IDE0040: Accessibility modifiers required (escalated to a build warning)
dotnet_diagnostic.IDE0040.severity = warning
```

Alternatively, you can configure an entire category to be a warning or error, by default, and then selectively turn off rules in that category that you don't want to run on build. For example:

```
[*.{cs,vb}]
# Default severity for analyzer diagnostics with category 'Style' (escalated to build warnings)
dotnet_analyzer_diagnostic.category-Style.severity = warning

# IDE0040: Accessibility modifiers required (disabled on build)
dotnet_diagnostic.IDE0040.severity = silent
```

NOTE

The code-style analysis feature is experimental and may change between the .NET 5 and .NET 6 releases.

Suppress a warning

One way to suppress a rule violation is to set the severity option for that rule ID to `none` in an EditorConfig file. For example:

```
dotnet_diagnostic.CA1822.severity = none
```

For more information and other ways to suppress warnings, see [How to suppress code analysis warnings](#).

Run code analysis as a GitHub Action

The [dotnet/code-analysis](#) GitHub Action lets you run .NET code analyzers as part of continuous integration (CI) in an offline mode. For more information, see [.NET code analysis GitHub Action](#).

Third-party analyzers

In addition to the official .NET analyzers, you can also install third party analyzers, such as [StyleCop](#), [Roslynator](#), [XUnit Analyzers](#), and [Sonar Analyzer](#).

See also

- [Code quality analysis rule reference](#)
- [Code style analysis rule reference](#)
- [Code analysis in Visual Studio](#)
- [.NET Compiler Platform SDK](#)
- [Tutorial: Write your first analyzer and code fix](#)
- [Code analysis in ASP.NET Core apps](#)

Configuration options for code analysis

9/20/2022 • 5 minutes to read • [Edit Online](#)

Code analysis rules have various configuration options. Some of these options are specified as key-value pairs in an [analyzer configuration file](#) using the syntax `<option key> = <option value>`. Other options, which configure code analysis as a whole, are available as properties in your project file.

The most common option you'll configure is a [rule's severity](#). You can configure the severity level for any rule, including [code quality rules](#) and [code style rules](#). For example, to enable a rule as a warning, add the following key-value pair to an [analyzer configuration file](#):

```
dotnet_diagnostic.<rule ID>.severity = warning
```

You can also configure additional options to customize rule behavior:

- Code quality rules have [options](#) to configure behavior, such as which method names a rule should apply to.
- Code style rules have [options](#) to define style preferences, such as where new lines are desirable.
- Third-party analyzer rules can define their own configuration options, with custom key names and value formats.

General options

These options apply to code analysis as a whole. They cannot be applied only to a specific rule.

- [Analysis mode](#)
- [Enable code analysis](#)
- [Exclude generated code](#)

For additional options, see [Code analysis properties](#).

Enable code analysis

Code analysis is enabled by default for projects that target .NET 5 and later versions. If you have the .NET 5+ SDK but your project targets a different .NET implementation, you can manually enable code analysis by setting the [EnableNETAnalyzers](#) property in your project file to `true`.

```
<PropertyGroup>
  <EnableNETAnalyzers>true</EnableNETAnalyzers>
</PropertyGroup>
```

Analysis mode

While the .NET SDK includes all code analysis rules, only some of them are enabled by default. The *analysis mode* determines which, if any, set of rules is enabled. You can choose a more aggressive analysis mode where most or all rules are enabled, or a more conservative analysis mode where most or all rules are disabled and, you can then opt in to specific rules as needed. Set your analysis mode by adding the [AnalysisMode](#) property to your project file.

```
<PropertyGroup>
  <AnalysisMode>Recommended</AnalysisMode>
</PropertyGroup>
```

Exclude generated code

.NET code analyzer warnings aren't useful on generated code files, such as designer-generated files, which users can't edit to fix any violations. In most cases, code analyzers skip generated code files and don't report violations on these files.

By default, files with certain file extensions or auto-generated file headers are treated as generated code files. For example, a file name ending with `.designer.cs` or `.generated.cs` is considered generated code. This configuration option lets you specify additional naming patterns to be treated as generated code. You configure additional files and folders by adding a `generated_code = true | false` entry to your [configuration file](#). For example, to treat all files whose name ends with `.MyGenerated.cs` as generated code, add the following entry:

```
[*.MyGenerated.cs]  
generated_code = true
```

Rule-specific options

Rule-specific options can be applied to a single rule, a set of rules, or all rules. The rule-specific options include:

- [Rule severity level](#)
- [Options specific to *code-quality* rules](#)

Severity level

The following table shows the different rule severities that you can configure for all analyzer rules, including [code quality](#) and [code style](#) rules.

SEVERITY CONFIGURATION VALUE	BUILD-TIME BEHAVIOR
<code>error</code>	Violations appear as build <i>errors</i> and cause builds to fail.
<code>warning</code>	Violations appear as build <i>warnings</i> but do not cause builds to fail (unless you have an option set to treat warnings as errors).
<code>suggestion</code>	Violations appear as build <i>messages</i> and as suggestions in the Visual Studio IDE.
<code>silent</code>	Violations aren't visible to the user.
<code>none</code>	Rule is suppressed completely.
<code>default</code>	The default severity of the rule is used. The default severities for each .NET release are listed in the roslyn-analyzers repo . In that table, "Disabled" corresponds to <code>none</code> , "Hidden" corresponds to <code>silent</code> , and "Info" corresponds to <code>suggestion</code> .

TIP

For information about how rule severities surface in Visual Studio, see [Severity levels](#).

Scope

- [Single rule](#)

To set the rule severity for a single rule, use the following syntax.

```
dotnet_diagnostic.<rule ID>.severity = <severity value>
```

- **Category of rules**

To set the default rule severity for a category of rules, use the following syntax. However, this severity setting only affects rules in that category that are enabled by default.

```
dotnet_analyzer_diagnostic.category-<rule category>.severity = <severity value>
```

The different categories are listed and described at [Rule categories](#). In addition, you can find the category for a specific rule on its reference page, for example, [CA1000](#).

- **All rules**

To set the default rule severity for all analyzer rules, use the following syntax. However, this severity setting only affects rules that are enabled by default.

```
dotnet_analyzer_diagnostic.severity = <severity value>
```

IMPORTANT

When you configure the severity level for multiple rules with a single entry, either for a *category* of rules or for *all* rules, the severity only applies to rules that are [enabled by default](#). To enable rules that are disabled by default, you must either:

- Add an explicit `dotnet_diagnostic.<rule ID>.severity = <severity>` configuration entry for each rule.
- In .NET 6+, enable a category of rules by setting `<AnalysisMode<Category>>` to `A11`.
- Enable *all* rules by setting `<AnalysisMode>` to `A11` or by setting `<AnalysisLevel>` to `latest-A11`.

NOTE

The prefix for setting severity for a single rule, `dotnet_diagnostic`, is slightly different than the prefix for configuring severity via category or for all rules, `dotnet_analyzer_diagnostic`.

Precedence

If you have multiple severity configuration entries that can be applied to the same rule ID, precedence is chosen in the following order:

- An entry for an individual rule by ID takes precedence over an entry for a category.
- An entry for a category takes precedence over an entry for all analyzer rules.

Consider the following example, where [CA1822](#) has the category "Performance":

```
[*.cs]
dotnet_diagnostic.CA1822.severity = error
dotnet_analyzer_diagnostic.category-performance.severity = warning
dotnet_analyzer_diagnostic.severity = suggestion
```

In the preceding example, all three severity entries are applicable to CA1822. However, using the specified precedence rules, the first rule ID-based entry wins over the next entries. In this example, CA1822 will have an effective severity of `error`. All other rules within the "Performance" category will have a severity of `warning`.

For information about how inter-file precedence is decided, see the [Precedence section of the Configuration files](#)

article.

Configuration files for code analysis rules

9/20/2022 • 5 minutes to read • [Edit Online](#)

Code analysis rules have various [configuration options](#). You specify these options as key-value pairs in one of the following analyzer configuration files:

- [EditorConfig](#) file: File-based or folder-based configuration options.
- [Global AnalyzerConfig](#) file: Project-level configuration options. Useful when some project files reside outside the project folder.

TIP

You can also set code analysis configuration properties in your project file. These properties configure code analysis at the bulk level, from completely turning it on or off down to category-level configuration. For more information, see [EnableNETAnalyzers](#), [AnalysisLevel](#), [AnalysisLevel<Category>](#), and [AnalysisMode](#).

EditorConfig

[EditorConfig](#) files are used to provide **options that apply to specific source files or folders**. Options are placed under section headers to identify the applicable files and folders. Add an entry for each rule you want to configure, and place it under the corresponding file extension section, for example, `[*.cs]`.

```
[*.cs]
<option_name> = <option_value>
```

In the above example, `[*.cs]` is an editorconfig section header to select all C# files with `.cs` file extension within the current folder, including subfolders. The subsequent entry, `<option_name> = <option_value>`, is an analyzer option that will be applied to all the C# files.

You can apply EditorConfig file conventions to a folder, a project, or an entire repo by placing the file in the corresponding directory. These options are applied when executing the analysis at build time and while you edit code in Visual Studio.

NOTE

EditorConfig options apply only to *source* files in a project or directory. Files that are included in a project as [AdditionalFiles](#) are not considered *source* files, and EditorConfig options aren't applied to these files. To apply a rule option to non-source files, specify the option in a [global configuration file](#).

If you have an existing `.editorconfig` file for editor settings such as indent size or whether to trim trailing whitespace, you can place your code analysis configuration options in the same file.

TIP

Visual Studio provides an `.editorconfig` item template that makes it easy to add one of these files to your project. For more information, see [Add an EditorConfig file to a project](#).

Example

Following is an example EditorConfig file to configure options and rule severity:

```
# Remove the line below if you want to inherit .editorconfig settings from higher directories
root = true

# C# files
[*.cs]

##### Core EditorConfig Options #####
# Indentation and spacing
indent_size = 4
indent_style = space
tab_width = 4

##### .NET Coding Conventions #####
# this. and Me. preferences
dotnet_style_qualification_for_method = true

##### Diagnostic configuration #####
# CA1000: Do not declare static members on generic types
dotnet_diagnostic.CA1000.severity = warning
```

Global AnalyzerConfig

Starting with the .NET 5 SDK (which is supported in Visual Studio 2019 version 16.8 and later), you can also configure analyzer options with global *AnalyzerConfig* files. These files are used to provide **options that apply to all the source files in a project**, regardless of their file names or file paths.

Unlike [EditorConfig](#) files, global configuration files can't be used to configure editor style settings for IDEs, such as indent size or whether to trim trailing whitespace. Instead, they are designed purely for specifying project-level analyzer configuration options.

Format

Unlike EditorConfig files, which must have section headers, such as `[*.cs]`, to identify the applicable files and folders, global AnalyzerConfig files don't have section headers. Instead, they require a top level entry of the form `is_global = true` to differentiate them from regular EditorConfig files. This indicates that all the options in the file apply to the entire project. For example:

```
is_global = true
<option_name> = <option_value>
```

Naming

Unlike EditorConfig files, which must be named `.editorconfig`, global config files do not need to have a specific name or extension. However, if you name these files as `.globalconfig`, then they are implicitly applied to all the C# and Visual Basic projects within the current folder, including subfolders. Otherwise, you must explicitly add the `GlobalAnalyzerConfigFiles` item to your MSBuild project file:

```
<ItemGroup>
  <GlobalAnalyzerConfigFiles Include="<path_to_global_analyzer_config>＂ />
</ItemGroup>
```

Consider the following naming recommendations:

- End users should name their global configuration files `.globalconfig`.

- NuGet package creators should name their global config files `<%Package_Name%>.globalconfig`.
- MSBuild tooling-generated global config files should be named `<%Target_Name%>_Generated.globalconfig` or similar.

NOTE

The top-level entry `is_global = true` is not required when the file is named `.globalconfig`, but it is recommended for clarity.

Example

Following is an example global AnalyzerConfig file to configure options and rule severity at the project level:

```
# Top level entry required to mark this as a global AnalyzerConfig file
is_global = true

# NOTE: No section headers for configuration entries

#### .NET Coding Conventions ####

# this. and Me. preferences
dotnet_style_qualification_for_method = true:warning

#### Diagnostic configuration ####

# CA1000: Do not declare static members on generic types
dotnet_diagnostic.CA1000.severity = warning
```

Precedence

Both EditorConfig files and global AnalyzerConfig files specify a key-value pair for each option. Conflicts arise when there are multiple entries with the same key but different values. The following precedence rules are used to resolve conflicts.

CONFLICTING ENTRY LOCATIONS	PRECEDENCE RULE
In the same configuration file	The entry that appears later in the file wins. This is true for conflicting entries within a single EditorConfig file and also within a single global AnalyzerConfig file.
In two EditorConfig files	The entry in the EditorConfig file that's deeper in the file system, and hence has a longer file path, wins.
In two global AnalyzerConfig files	<p>.NET 5: A compiler warning is reported and both entries are ignored.</p> <p>.NET 6 and later versions: The entry from the file with a higher value for <code>global_level</code> takes precedence. If <code>global_level</code> isn't explicitly defined and the file is named <code>.globalconfig</code>, the <code>global_level</code> value defaults to <code>100</code>; for all other global AnalyzerConfig files, <code>global_level</code> defaults to <code>0</code>. If the <code>global_level</code> values for the configuration files with conflicting entries are equal, a compiler warning is reported and both entries are ignored.</p>
In an EditorConfig file and a Global AnalyzerConfig file	The entry in the EditorConfig file wins.

Severity options

For [severity configuration](#) options, the following *additional* precedence rules apply:

- Severity options specified at the command line as compiler options (`-nowarn` or `-warnaserror`) always override [severity configuration](#) options specified in EditorConfig and global AnalyzerConfig files.
- The precedence rules for conflicting severity entries from a [ruleset file](#) and an EditorConfig or global AnalyzerConfig file is *undefined*.

Ruleset files are deprecated in favor of EditorConfig and global AnalyzerConfig files. We recommend that you [convert ruleset files to an equivalent EditorConfig file](#).

- For information about precedence rules for related severity options with different keys, for example, when different severities are specified for a single rule and for the category that rule falls under, see [Configuration options for code analysis](#).

See also

- [EditorConfig vs global AnalyzerConfig design issue](#)

How to suppress code analysis warnings

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article covers the various ways you can suppress warnings from code analysis when you build your .NET app. You can suppress code quality rules, code style rules, and third-party analyzer rules using the information provided here.

TIP

If you're using Visual Studio as your development environment, the *light bulb* menu provides options that generate the code to suppress warnings for you. For more information, see [Suppress violations](#).

Disable the rule

By disabling the code analysis rule that's causing the warning, you disable the rule for your entire file or project (depending on the scope of the [configuration file](#) that you use). To disable the rule, set its severity to `none` in the configuration file.

```
[*.{cs,vb}]\ndotnet_diagnostic.<rule-ID>.severity = none
```

For more information about rule severities, see [Configure rule severity](#).

Use a preprocessor directive

Use a [#pragma warning \(C#\)](#) or [Disable \(Visual Basic\)](#) directive to suppress the warning for only a specific line of code.

```
try { ... }\ncatch (Exception e)\n{\n#pragma warning disable CA2200 // Rethrow to preserve stack details\n    throw e;\n#pragma warning restore CA2200 // Rethrow to preserve stack details\n}
```

```
Try\n...\nCatch e As Exception\n#Disable Warning CA2200 ' Rethrow to preserve stack details\n    Throw e\n#Enable Warning CA2200 ' Rethrow to preserve stack details\nEnd Try
```

Use the SuppressMessageAttribute

You can use a [SuppressMessageAttribute](#) to suppress a warning either in the source file or in a global suppressions file for the project (*GlobalSuppressions.cs* or *GlobalSuppressions.vb*). This attribute provides a way to suppress a warning in only certain parts of your project or file.

The two required, positional parameters for the `SuppressMessageAttribute` attribute are the *category* of the rule and the *rule ID*. The following code snippet passes "Usage" and "CA2200:Rethrow to preserve stack details" for these parameters.

```
[System.Diagnostics.CodeAnalysis.SuppressMessage("Usage", "CA2200:Rethrow to preserve stack details",
Justification = "Not production code.")]
private static void IgnorableCharacters()
{
    try
    {
        ...
    }
    catch (Exception e)
    {
        throw e;
    }
}
```

If you add the attribute to the global suppressions file, you *scope* the suppression to the desired level, for example "member". You specify the API where the warning should be suppressed using the `Target` property.

```
[assembly: SuppressMessage("Usage", "CA2200:Rethrow to preserve stack details", Justification = "Not
production code.", Scope = "member", Target = "~M:MyApp.Program.IgnorableCharacters")]
```

Use the *documentation ID* for the API you want to reference in the `Target` attribute. For information about documentation IDs, see [Documentation ID format](#).

To suppress warnings for compiler-generated code that doesn't map to explicitly provided user source, you must put the suppression attribute in a global suppressions file. For example, the following code suppresses a violation against a compiler-emitted constructor:

```
[module: SuppressMessage("Design", "CA1055:AbstractTypesDoNotHavePublicConstructors", Scope="member",
Target="MyTools.Type..ctor")]
```

See also

- [Suppress violations \(Visual Studio\)](#)

Code quality rule configuration options

9/20/2022 • 5 minutes to read • [Edit Online](#)

The *code quality* rules have additional configuration options, besides just [configuring their severity](#). For example, each code quality analyzer can be configured to only apply to specific parts of your codebase. You specify these options by adding key-value pairs to the same [EditorConfig](#) file where you specify rule severities and general editor preferences.

NOTE

This article does not detail how to configure a rule's severity. The `.editorconfig` option to set a rule's severity has a different prefix (`dotnet_diagnostic`) to the options described here (`dotnet_code_quality`). In addition, the options described here pertain to code quality rules only, whereas the severity option applies to code style rules as well. As a quick reference, you can configure a rule's severity using the following option syntax:

```
dotnet_diagnostic.<rule ID>.severity = <severity value>
```

However, for detailed information about configuring rule severity, see [Severity level](#).

Option scopes

Each refining option can be configured for all rules, for a category of rules (for example, Security or Design), or for a specific rule.

All rules

The syntax for configuring an option for *all*/rules is as follows:

SYNTAX	EXAMPLE
<code>dotnet_code_quality.<OptionName> = <OptionValue></code>	<code>dotnet_code_quality.api_surface = public</code>

The values for `<OptionName>` are listed under [Options](#).

Category of rules

The syntax for configuring an option for a *category of rules* is as follows:

SYNTAX	EXAMPLE
<code>dotnet_code_quality.<RuleCategory>.<OptionName> = OptionValue</code>	<code>dotnet_code_quality.Naming.api_surface = public</code>

The following table lists the available values for `<RuleCategory>`.

Design

Documentation

Globalization

Interoperability

Maintainability

Naming

Performance

SingleFile

Reliability

Security

Usage

Specific rule

The syntax for configuring an option for a *specific* rule is as follows:

SYNTAX	EXAMPLE
<code>dotnet_code_quality.<RuleId>.<OptionName> = <OptionValue></code>	<code>dotnet_code_quality.CA1040.api_surface = public</code>

Options

This section lists some of the available options. To see the full list of available options, see [Analyzer configuration](#).

- [api_surface](#)
- [exclude_async_void_methods](#)
- [exclude_single_letter_type_parameters](#)
- [output_kind](#)
- [required_modifiers](#)
- [exclude_extension_method_this_parameter](#)
- [null_check_validation_methods](#)
- [additional_string_formatting_methods](#)
- [excluded_type_names_with_derived_types](#)
- [excluded_symbol_names](#)
- [disallowed_symbol_names](#)

api_surface

DESCRIPTION	ALLOWABLE VALUES	DEFAULT VALUE	CONFIGURABLE RULES

DESCRIPTION	ALLOWABLE VALUES	DEFAULT VALUE	CONFIGURABLE RULES
Which part of the API surface to analyze	<input type="checkbox"/> public <input type="checkbox"/> internal or <input type="checkbox"/> friend <input type="checkbox"/> private <input type="checkbox"/> all	<input type="checkbox"/> public	CA1000 CA1002 CA1003 CA1005 CA1008 CA1010 CA1012 CA1021 CA1024 CA1027 CA1028 CA1030 CA1036 CA1040 CA1041 CA1043 CA1044 CA1045 CA1046 CA1047 CA1051 CA1052 CA1054 CA1055 CA1056 CA1058 CA1062 CA1063 CA1068 CA1070 CA1700 CA1707 CA1708 CA1710 CA1711 CA1714 CA1715 CA1716 CA1717 CA1720 CA1721 CA1725 CA1801 CA1802 CA1815 CA1819 CA1822 CA2208 CA2217 CA2225 CA2226 CA2231 CA2234

exclude_async_void_methods

DESCRIPTION	ALLOWABLE VALUES	DEFAULT VALUE	CONFIGURABLE RULES
Whether to ignore asynchronous methods that don't return a value	<input type="checkbox"/> true <input type="checkbox"/> false	<input type="checkbox"/> false	CA2007

NOTE

This option was named `skip_async_void_methods` in an earlier version.

exclude_single_letter_type_parameters

DESCRIPTION	ALLOWABLE VALUES	DEFAULT VALUE	CONFIGURABLE RULES
Whether to exclude single-character type parameters from the rule, for example, <code>s</code> in <code>Collection<s></code>	<input type="checkbox"/> true <input type="checkbox"/> false	<input type="checkbox"/> false	CA1715

NOTE

This option was named `allow_single_letter_type_parameters` in an earlier version.

output_kind

DESCRIPTION	ALLOWABLE VALUES	DEFAULT VALUE	CONFIGURABLE RULES
Specifies that code in a project that generates this type of assembly should be analyzed	One or more fields of the OutputKind enumeration Separate multiple values with a comma (,)	All output kinds	CA2007

required_modifiers

DESCRIPTION	ALLOWABLE VALUES	DEFAULT VALUE	CONFIGURABLE RULES
Specifies the required modifiers for APIs that should be analyzed	One or more values from the below allowed modifiers table Separate multiple values with a comma (,)	Depends on each rule	CA1802

ALLOWED MODIFIER	SUMMARY
<code>none</code>	No modifier requirement
<code>static</code> or <code>Shared</code>	Must be declared as <code>static</code> (<code>Shared</code> in Visual Basic)
<code>const</code>	Must be declared as <code>const</code>
<code>readonly</code>	Must be declared as <code>readonly</code>
<code>abstract</code>	Must be declared as <code>abstract</code>
<code>virtual</code>	Must be declared as <code>virtual</code>
<code>override</code>	Must be declared as <code>override</code>
<code>sealed</code>	Must be declared as <code>sealed</code>
<code>extern</code>	Must be declared as <code>extern</code>
<code>async</code>	Must be declared as <code>async</code>

exclude_extension_method_this_parameter

DESCRIPTION	ALLOWABLE VALUES	DEFAULT VALUE	CONFIGURABLE RULES
Whether to skip analysis for the <code>this</code> parameter of extension methods	<code>true</code> <code>false</code>	<code>false</code>	CA1062

null_check_validation_methods

DESCRIPTION	ALLOWABLE VALUES	DEFAULT VALUE	CONFIGURABLE RULES
Names of null-check validation methods that validate that arguments passed to the method are non-null	Allowed method name formats (separated by): - Method name only (includes all methods with the name, regardless of the containing type or namespace) - Fully qualified names in the symbol's documentation ID format , with an optional <code>M:</code> prefix	None	CA1062

additional_string_formatting_methods

DESCRIPTION	ALLOWABLE VALUES	DEFAULT VALUE	CONFIGURABLE RULES
Names of additional string formatting methods	Allowed method name formats (separated by): - Method name only (includes all methods with the name, regardless of the containing type or namespace) - Fully qualified names in the symbol's documentation ID format , with an optional <code>M:</code> prefix	None	CA2241

excluded_type_names_with_derived_types

DESCRIPTION	ALLOWABLE VALUES	DEFAULT VALUE	CONFIGURABLE RULES
Names of types, such that the type and all its derived types are excluded for analysis	Allowed symbol name formats (separated by): - Type name only (includes all types with the name, regardless of the containing type or namespace) - Fully qualified names in the symbol's documentation ID format , with an optional <code>T:</code> prefix	None	CA1001 CA1054 CA1055] (quality-rules/ca1056.md) CA1062 CA1068 CA1303 CA1304 CA1508 CA2000 CA2100 CA2301 CA2302 CA2311 CA2312 CA2321 CA2322 CA2327 CA2328 CA2329 CA2330 CA3001 CA3002 CA3003 CA3004 CA3005 CA3006 CA3007 CA3008 CA3009 CA3010 CA3011 CA3012 CA5361 CA5376 CA5377 CA5378 CA5380 CA5381 CA5382 CA5383 CA5384 CA5387 CA5388 CA5389 CA5390

excluded_symbol_names

DESCRIPTION	ALLOWABLE VALUES	DEFAULT VALUE	CONFIGURABLE RULES
Names of symbols that are excluded for analysis	<p>Allowed symbol name formats (separated by):</p> <ul style="list-style-type: none"> - Symbol name only (includes all symbols with the name, regardless of the containing type or namespace) - Fully qualified names in the symbol's documentation ID format. <p>Each symbol name requires a symbol kind prefix, such as <code>M:</code> prefix for methods, <code>T:</code> prefix for types, and <code>N:</code> prefix for namespaces.</p> <ul style="list-style-type: none"> - <code>.ctor</code> for constructors and <code>.cctor</code> for static constructors 	None	CA1001 CA1054 CA1055] (quality-rules/ca1056.md) CA1062 CA1068 CA1303 CA1304 CA1508 CA2000 CA2100 CA2301 CA2302 CA2311 CA2312 CA2321 CA2322 CA2327 CA2328 CA2329 CA2330 CA3001 CA3002 CA3003 CA3004 CA3005 CA3006 CA3007 CA3008 CA3009 CA3010 CA3011 CA3012 CA5361 CA5376 CA5377 CA5378 CA5380 CA5381 CA5382 CA5383 CA5384 CA5387 CA5388 CA5389 CA5390

disallowed_symbol_names

DESCRIPTION	ALLOWABLE VALUES	DEFAULT VALUE	CONFIGURABLE RULES
Names of symbols that are disallowed in the context of the analysis	<p>Allowed symbol name formats (separated by):</p> <ul style="list-style-type: none"> - Symbol name only (includes all symbols with the name, regardless of the containing type or namespace) - Fully qualified names in the symbol's documentation ID format. <p>Each symbol name requires a symbol kind prefix, such as <code>M:</code> prefix for methods, <code>T:</code> prefix for types, and <code>N:</code> prefix for namespaces.</p> <ul style="list-style-type: none"> - <code>.ctor</code> for constructors and <code>.cctor</code> for static constructors 	None	CA1031

Predefined configuration files

9/20/2022 • 2 minutes to read • [Edit Online](#)

Predefined EditorConfig and rule set files are available that make it quick and easy to enable a category of code quality rules, such as security or design rules. By enabling a specific category of rules, you can identify targeted issues and specific conditions. To access these predefined files, install the [Microsoft.CodeAnalysis.NetAnalyzers](#) NuGet analyzer package.

[Microsoft.CodeAnalysis.NetAnalyzers](#) includes predefined EditorConfig files and rule sets for the following rule categories:

- All rules
- Dataflow
- Design
- Documentation
- Globalization
- Interoperability
- Maintainability
- Naming
- Performance
- Ported from FxCop
- Reliability
- Security
- Usage

Each of those categories of rules has an EditorConfig or rule set file to:

- Enable all the rules in the category (and disable all other rules)
- Use each rule's default severity and enabled by default setting (and disable all other rules)

TIP

The "all rules" category has an additional EditorConfig or rule set file to disable all rules. Use this file to quickly get rid of any analyzer warnings or errors in a project.

Predefined EditorConfig files

The predefined EditorConfig files for the [Microsoft.CodeAnalysis.NetAnalyzers](#) analyzer package are located in the *editorconfig* subdirectory of where the NuGet package was installed. For example, the EditorConfig file to enable all security rules is located at *editorconfig/SecurityRulesEnabled/.editorconfig*.

Copy the chosen *.editorconfig* file to your project's root directory.

Predefined rule sets

The predefined rule set files for the [Microsoft.CodeAnalysis.NetAnalyzers](#) analyzer package are located in the *rulesets* subdirectory of where the NuGet package was installed. For example, the rule set file to enable all security rules is located at *rulesets/SecurityRulesEnabled.ruleset*. Copy one or more of the rule sets and paste them in the directory that contains your project.

See also

- [Analyzer configuration](#)
- [.NET code style rule options for EditorConfig](#)

Code-style rule options

9/20/2022 • 4 minutes to read • [Edit Online](#)

You can define and maintain consistent *code style* in your codebase by defining .NET code-style rules and their associated options in a [configuration file](#). These rules are surfaced by various development IDEs, such as Visual Studio, as you edit your code. For .NET projects, these rules can also be [enforced at build time](#). You can enable or disable individual rules and configure the degree to which you want each rule enforced, via a severity level.

TIP

- When you define code style options in an EditorConfig file, you're configuring how you want the [code style analyzers](#) to analyze your code. The EditorConfig file is the configuration file for these analyzers.
- Code style options can also be set in Visual Studio in the [Text editor options](#) dialog. These are per-user options that are only respected while editing in Visual Studio. These options are not respected at build time or by other IDEs. Additionally, if the project or solution opened inside Visual Studio has an EditorConfig file, then options from the EditorConfig file take precedence. In Visual Studio on Windows, you can also generate an EditorConfig file from your text-editor options. Select **Tools > Options > Text Editor > [C# or Basic] > Code Style > General**, and then click **Generate .editorconfig file from settings**. For more information, see [Code style preferences](#).

Code style rules are divided into following subcategories:

- [Language rules](#)
- [Unnecessary code rules](#)
- [Formatting rules](#)
- [Naming rules](#)

Each of these subcategories defines its own syntax for specifying options. For more information about these rules and the corresponding options, see [Code-style rule reference](#).

Example EditorConfig file

To help you get started, here's an example `.editorconfig` file with the default options.

NOTE

The concise syntax seen in this file, where the severity is specified after the option value for language rules, for example, `dotnet_style_READONLY_FIELD = true:suggestion`, is only fully understood by Visual Studio. Compilers don't understand the `:<severity-level>` suffix and it is ignored if you compile your code from the command line. To set severity in a way that's understood by both Visual Studio and the C# and Visual Basic compilers, set the severity on the rule that contains the option by using the following syntax:

```
dotnet_diagnostic.<rule-ID>.severity = <severity-level>
```

For more information, see [Option format](#).

TIP

In Visual Studio, you can add the following default .NET .editorconfig file to your project from the **Add New Item** dialog box.

```
#####
# Core EditorConfig Options  #
#####

root = true
# All files
[*]
indent_size = space

# XML project files
[*.csproj,vbproj,vcxproj,vcxproj.filters,proj,projitems,shproj]
indent_size = 2

# XML config files
[*.props,targets,ruleset,config,nuspec,resx,vsixmanifest,vsct]
indent_size = 2

# Code files
[*.cs,csx,vb,vbx]
indent_size = 4
insert_final_newline = true
charset = utf-8-bom
#####

# .NET Coding Conventions    #
#####

[*.cs,vb]
# Organize usings
dotnet_sort_system_directives_first = true
# this. preferences
dotnet_style_qualification_for_field = false:silent
dotnet_style_qualification_for_property = false:silent
dotnet_style_qualification_for_method = false:silent
dotnet_style_qualification_for_event = false:silent
# Language keywords vs BCL types preferences
dotnet_style_predefined_type_for_locals_parameters_members = true:silent
dotnet_style_predefined_type_for_member_access = true:silent
# Parentheses preferences
dotnet_style_parentheses_in_arithmetic_binary_operators = always_for_clarity:silent
dotnet_style_parentheses_in_relational_binary_operators = always_for_clarity:silent
dotnet_style_parentheses_in_other_binary_operators = always_for_clarity:silent
dotnet_style_parentheses_in_other_operators = never_if_unnecessary:silent
# Modifier preferences
dotnet_style_require_accessibility_modifiers = for_non_interface_members:silent
dotnet_style_READONLY_field = true:suggestion
# Expression-level preferences
dotnet_style_object_initializer = true:suggestion
dotnet_style_collection_initializer = true:suggestion
dotnet_style_explicit_tuple_names = true:suggestion
dotnet_style_null_propagation = true:suggestion
dotnet_style_coalesce_expression = true:suggestion
dotnet_style_prefer_is_null_check_over_reference_equality_method = true:silent
dotnet_style_prefer_inferred_tuple_names = true:suggestion
dotnet_style_prefer_inferred_anonymous_type_member_names = true:suggestion
dotnet_style_prefer_auto_properties = true:silent
dotnet_style_prefer_conditional_expression_over_assignment = true:silent
dotnet_style_prefer_conditional_expression_over_return = true:silent
#####

# Naming Conventions      #
#####

# Style Definitions
dotnet_naming_style.pascal_case_style.capitalization      = pascal_case
# Use PascalCase for constant fields
```

```
# use fastalast for constant rules

dotnet_naming_rule.constant_fields_should_be_pascal_case.severity = suggestion
dotnet_naming_rule.constant_fields_should_be_pascal_case.symbols = constant_fields
dotnet_naming_rule.constant_fields_should_be_pascal_case.style = pascal_case_style
dotnet_naming_symbols.constant_fields.applicable_kinds = field
dotnet_naming_symbols.constant_fields.applicable_accessibilities = *
dotnet_naming_symbols.constant_fields.required_modifiers = const
#####
# C# Coding Conventions #
#####

[*.cs]
# var preferences
csharp_style_var_for_built_in_types = true:silent
csharp_style_var_when_type_is_apparent = true:silent
csharp_style_var_elsewhere = true:silent
# Expression-bodied members
csharp_style_expression_bodied_methods = false:silent
csharp_style_expression_bodied_constructors = false:silent
csharp_style_expression_bodied_operators = false:silent
csharp_style_expression_bodied_properties = true:silent
csharp_style_expression_bodied_indexers = true:silent
csharp_style_expression_bodied_accessors = true:silent
# Pattern matching preferences
csharp_style_pattern_matching_over_is_with_cast_check = true:suggestion
csharp_style_pattern_matching_over_as_with_null_check = true:suggestion
# Null-checking preferences
csharp_style_throw_expression = true:suggestion
csharp_style_conditional_delegate_call = true:suggestion
# Modifier preferences
csharp_preferred_modifier_order =
public,private,protected,internal,static,extern,new,virtual,abstract,sealed,override,readonly,unsafe,volatile
e,async:suggestion
# Expression-level preferences
csharp_prefer_braces = true:silent
csharp_style_deconstructed_variable_declaration = true:suggestion
csharp_prefer_simple_default_expression = true:suggestion
csharp_style_prefer_local_over_anonymous_function = true:suggestion
csharp_style_inlined_variable_declaration = true:suggestion
#####
# C# Formatting Rules #
#####

# New line preferences
csharp_new_line_before_open_brace = all
csharp_new_line_before_else = true
csharp_new_line_before_catch = true
csharp_new_line_before_finally = true
csharp_new_line_before_members_in_object_initializers = true
csharp_new_line_before_members_in_anonymous_types = true
csharp_new_line_between_query_expression_clauses = true
# Indentation preferences
csharp_indent_case_contents = true
csharp_indent_switch_labels = true
csharp_indent_labels = flush_left
# Space preferences
csharp_space_after_cast = false
csharp_space_after_keywords_in_control_flow_statements = true
csharp_space_between_method_call_parameter_list_parentheses = false
csharp_space_between_method_declaration_parameter_list_parentheses = false
csharp_space_between_parentheses = false
csharp_space_before_colon_in_inheritance_clause = true
csharp_space_after_colon_in_inheritance_clause = true
csharp_space_around_binary_operators = before_and_after
csharp_space_between_method_declaration_empty_parameter_list_parentheses = false
csharp_space_between_method_call_name_and_opening_parenthesis = false
csharp_space_between_method_call_empty_parameter_list_parentheses = false
# Wrapping preferences
csharp_preserve_single_line_statements = true
csharp_preserve_single_line_blocks = true
#####
# VB Coding Conventions #
#####
```

```
# VB Coding Conventions      #
#####
[*.vb]
# Modifier preferences
visual_basic_preferred_modifier_order =
Partial,Default,Private,Protected,Public,Friend,NotOverridable,Overridable,MustOverride,Overloads,Overrides,
MustInherit,NotInheritable,Static,Shared,Shadows,ReadOnly,WriteOnly,Dim,Const,WithEvents,Widening,Narrowing,
Custom,Async:suggestion
```

See also

- [Code style analysis rule reference](#)
- [Enforce code style on build](#)
- [Quick Actions in Visual Studio](#)
- [Create portable custom editor options in Visual Studio](#)
- [.NET Compiler Platform "Roslyn" .editorconfig file](#)
- [.NET runtime .editorconfig file](#)

Rule categories

9/20/2022 • 2 minutes to read • [Edit Online](#)

Each code analysis rule belongs to a category of rules. For example, design rules support adherence to the .NET design guidelines, and security rules help prevent security flaws. You can [configure the severity level](#) for an entire category of rules. You can also [configure additional options](#) on a per-category basis.

The following table shows the different code analysis rule categories and provides a link to the rules in each category. It also lists the configuration value to use in an EditorConfig file to [bulk-configure rule severity](#) on a per-category basis. For example, to set the severity of security rule violations to be errors, the EditorConfig entry is `dotnet_analyzer_diagnostic.category-Security.severity = error`.

TIP

Setting the severity for a category of rules using the `dotnet_analyzer_diagnostic.category-<category>.severity` syntax doesn't apply to rules that are disabled by default. However, starting in .NET 6, you can use the [AnalysisMode<Category>](#) project property to enable all the rules in a category.

CATEGORY	DESCRIPTION	EDITORCONFIG VALUE
Design rules	Design rules support adherence to the .NET Framework Design Guidelines .	<code>dotnet_analyzer_diagnostic.category-Design.severity</code>
Documentation rules	Documentation rules support writing well-documented libraries through the correct use of XML documentation comments for externally visible APIs.	<code>dotnet_analyzer_diagnostic.category-Documentation.severity</code>
Globalization rules	Globalization rules support world-ready libraries and applications.	<code>dotnet_analyzer_diagnostic.category-Globalization.severity</code>
Portability and interoperability rules	Portability rules support portability across different platforms. Interoperability rules support interaction with COM clients.	<code>dotnet_analyzer_diagnostic.category-Interoperability.severity</code>
Maintainability rules	Maintainability rules support library and application maintenance.	<code>dotnet_analyzer_diagnostic.category-Maintainability.severity</code>
Naming rules	Naming rules support adherence to the naming conventions of the .NET design guidelines.	<code>dotnet_analyzer_diagnostic.category-Naming.severity</code>
Performance rules	Performance rules support high-performance libraries and applications.	<code>dotnet_analyzer_diagnostic.category-Performance.severity</code>
SingleFile rules	Single-file rules support single-file applications.	<code>dotnet_analyzer_diagnostic.category-SingleFile.severity</code>

CATEGORY	DESCRIPTION	EDITORCONFIG VALUE
Reliability rules	Reliability rules support library and application reliability, such as correct memory and thread usage.	<code>dotnet_analyzer_diagnostic.category-Reliability.severity</code>
Security rules	Security rules support safer libraries and applications. These rules help prevent security flaws in your program.	<code>dotnet_analyzer_diagnostic.category-Security.severity</code>
Style rules	Style rules support consistent code style in your codebase. These rules start with the "IDE" prefix.	<code>dotnet_analyzer_diagnostic.category-Style.severity</code>
Usage rules	Usage rules support proper usage of .NET.	<code>dotnet_analyzer_diagnostic.category-Usage.severity</code>
N/A	You can use this EditorConfig value to enable the following rules: IDE0051 , IDE0064 , IDE0076 . While these rules start with "IDE", they are not technically part of the <code>Style</code> category.	<code>dotnet_analyzer_diagnostic.category-CodeQuality.severity</code>

Code quality rules

9/20/2022 • 53 minutes to read • [Edit Online](#)

.NET code analysis provides rules that aim to improve code quality. The rules are organized into areas such as design, globalization, performance, and security. Certain rules are specific to .NET API usage, while others are about generic code quality.

Index of rules

The following table lists code quality analysis rules.

RULE ID AND WARNING	DESCRIPTION
CA1000: Do not declare static members on generic types	When a static member of a generic type is called, the type argument must be specified for the type. When a generic instance member that does not support inference is called, the type argument must be specified for the member. In these two cases, the syntax for specifying the type argument is different and easily confused.
CA1001: Types that own disposable fields should be disposable	A class declares and implements an instance field that is a System.IDisposable type, and the class does not implement IDisposable. A class that declares an IDisposable field indirectly owns an unmanaged resource and should implement the IDisposable interface.
CA1002: Do not expose generic lists	System.Collections.Generic.List<(Of <(T>)>) is a generic collection that is designed for performance, not inheritance. Therefore, List does not contain any virtual members. The generic collections that are designed for inheritance should be exposed instead.
CA1003: Use generic event handler instances	A type contains a delegate that returns void, whose signature contains two parameters (the first an object and the second a type that is assignable to EventArgs), and the containing assembly targets Microsoft .NET Framework 2.0.
CA1005: Avoid excessive parameters on generic types	The more type parameters a generic type contains, the more difficult it is to know and remember what each type parameter represents. It is usually obvious with one type parameter, as in List<T>, and in certain cases that have two type parameters, as in Dictionary< TKey, TValue >. However, if more than two type parameters exist, the difficulty becomes too great for most users.
CA1008: Enums should have zero value	The default value of an uninitialized enumeration, just as other value types, is zero. A nonflags-attributed enumeration should define a member by using the value of zero so that the default value is a valid value of the enumeration. If an enumeration that has the FlagsAttribute attribute applied defines a zero-valued member, its name should be "None" to indicate that no values have been set in the enumeration.
CA1010: Collections should implement generic interface	To broaden the usability of a collection, implement one of the generic collection interfaces. Then the collection can be used to populate generic collection types.

Rule ID and Warning	Description
CA1012: Abstract types should not have constructors	Constructors on abstract types can be called only by derived types. Because public constructors create instances of a type, and you cannot create instances of an abstract type, an abstract type that has a public constructor is incorrectly designed.
CA1014: Mark assemblies with CLSCompliantAttribute	The Common Language Specification (CLS) defines naming restrictions, data types, and rules to which assemblies must conform if they will be used across programming languages. Good design dictates that all assemblies explicitly indicate CLS compliance by using CLSCompliantAttribute . If this attribute is not present on an assembly, the assembly is not compliant.
CA1016: Mark assemblies with AssemblyVersionAttribute	.NET uses the version number to uniquely identify an assembly and to bind to types in strongly named assemblies. The version number is used together with version and publisher policy. By default, applications run only with the assembly version with which they were built.
CA1017: Mark assemblies with ComVisibleAttribute	ComVisibleAttribute determines how COM clients access managed code. Good design dictates that assemblies explicitly indicate COM visibility. COM visibility can be set for the whole assembly and then overridden for individual types and type members. If this attribute is not present, the contents of the assembly are visible to COM clients.
CA1018: Mark attributes with AttributeUsageAttribute	When you define a custom attribute, mark it by using AttributeUsageAttribute to indicate where in the source code the custom attribute can be applied. The meaning and intended usage of an attribute will determine its valid locations in code.
CA1019: Define accessors for attribute arguments	Attributes can define mandatory arguments that must be specified when you apply the attribute to a target. These are also known as positional arguments because they are supplied to attribute constructors as positional parameters. For every mandatory argument, the attribute should also provide a corresponding read-only property so that the value of the argument can be retrieved at execution time. Attributes can also define optional arguments, which are also known as named arguments. These arguments are supplied to attribute constructors by name and should have a corresponding read/write property.
CA1021: Avoid out parameters	Passing types by reference (using out or ref) requires experience with pointers, understanding how value types and reference types differ, and handling methods with multiple return values. Also, the difference between out and ref parameters is not widely understood.
CA1024: Use properties where appropriate	A public or protected method has a name that starts with "Get", takes no parameters, and returns a value that is not an array. The method might be a good candidate to become a property.
CA1027: Mark enums with FlagsAttribute	An enumeration is a value type that defines a set of related named constants. Apply FlagsAttribute to an enumeration when its named constants can be meaningfully combined.
CA1028: Enum storage should be Int32	An enumeration is a value type that defines a set of related named constants. By default, the System.Int32 data type is used to store the constant value. Although you can change this underlying type, it is not required or recommended for most scenarios.

Rule ID and Warning	Description
CA1030: Use events where appropriate	This rule detects methods that have names that ordinarily would be used for events. If a method is called in response to a clearly defined state change, the method should be invoked by an event handler. Objects that call the method should raise events instead of calling the method directly.
CA1031: Do not catch general exception types	General exceptions should not be caught. Catch a more specific exception, or rethrow the general exception as the last statement in the catch block.
CA1032: Implement standard exception constructors	Failure to provide the full set of constructors can make it difficult to correctly handle exceptions.
CA1033: Interface methods should be callable by child types	An unsealed externally visible type provides an explicit method implementation of a public interface and does not provide an alternative externally visible method that has the same name.
CA1034: Nested types should not be visible	A nested type is a type that is declared in the scope of another type. Nested types are useful to encapsulate private implementation details of the containing type. Used for this purpose, nested types should not be externally visible.
CA1036: Override methods on comparable types	A public or protected type implements the System.IComparable interface. It does not override Object.Equals nor does it overload the language-specific operator for equality, inequality, less than, or greater than.
CA1040: Avoid empty interfaces	Interfaces define members that provide a behavior or usage contract. The functionality that is described by the interface can be adopted by any type, regardless of where the type appears in the inheritance hierarchy. A type implements an interface by providing implementations for the members of the interface. An empty interface does not define any members; therefore, it does not define a contract that can be implemented.
CA1041: Provide ObsoleteAttribute message	A type or member is marked by using a System.ObsoleteAttribute attribute that does not have its ObsoleteAttribute.Message property specified. When a type or member that is marked by using ObsoleteAttribute is compiled, the Message property of the attribute is displayed. This gives the user information about the obsolete type or member.
CA1043: Use integral or string argument for indexers	Indexers (that is, indexed properties) should use integral or string types for the index. These types are typically used for indexing data structures and they increase the usability of the library. Use of the Object type should be restricted to those cases where the specific integral or string type cannot be specified at design time.
CA1044: Properties should not be write only	Although it is acceptable and often necessary to have a read-only property, the design guidelines prohibit the use of write-only properties. This is because letting a user set a value, and then preventing the user from viewing that value, does not provide any security. Also, without read access, the state of shared objects cannot be viewed, which limits their usefulness.

Rule ID and Warning	Description
CA1045: Do not pass types by reference	Passing types by reference (using <code>out</code> or <code>ref</code>) requires experience with pointers, understanding how value types and reference types differ, and handling methods that have multiple return values. Library architects who design for a general audience should not expect users to become proficient in working with <code>out</code> or <code>ref</code> parameters.
CA1046: Do not overload operator equals on reference types	For reference types, the default implementation of the equality operator is almost always correct. By default, two references are equal only if they point to the same object.
CA1047: Do not declare protected members in sealed types	Types declare protected members so that inheriting types can access or override the member. By definition, sealed types cannot be inherited, which means that protected methods on sealed types cannot be called.
CA1050: Declare types in namespaces	Types are declared in namespaces to prevent name collisions and as a way to organize related types in an object hierarchy.
CA1051: Do not declare visible instance fields	The primary use of a field should be as an implementation detail. Fields should be private or internal and should be exposed by using properties.
CA1052: Static holder types should be sealed	A public or protected type contains only static members and is not declared by using the <code>sealed</code> (C# Reference) (<code>NotInheritable</code>) modifier. A type that is not meant to be inherited should be marked by using the <code>sealed</code> modifier to prevent its use as a base type.
CA1053: Static holder types should not have constructors	A public or nested public type declares only static members and has a public or protected default constructor. The constructor is unnecessary because calling static members does not require an instance of the type. The string overload should call the uniform resource identifier (URI) overload by using the string argument for safety and security.
CA1054: URI parameters should not be strings	If a method takes a string representation of a URI, a corresponding overload should be provided that takes an instance of the <code>Uri</code> class, which provides these services in a safe and secure manner.
CA1055: URI return values should not be strings	This rule assumes that the method returns a URI. A string representation of a URI is prone to parsing and encoding errors, and can lead to security vulnerabilities. The <code>System.Uri</code> class provides these services in a safe and secure manner.
CA1056: URI properties should not be strings	This rule assumes that the property represents a Uniform Resource Identifier (URI). A string representation of a URI is prone to parsing and encoding errors, and can lead to security vulnerabilities. The <code>System.Uri</code> class provides these services in a safe and secure manner.
CA1058: Types should not extend certain base types	An externally visible type extends certain base types. Use one of the alternatives.
CA1060: Move P/Invokes to NativeMethods class	Platform Invocation methods, such as those that are marked by using the <code>System.Runtime.InteropServices.DllImportAttribute</code> attribute, or methods that are defined by using the <code>Declare</code> keyword in Visual Basic, access unmanaged code. These methods should be of the <code>NativeMethods</code> , <code>SafeNativeMethods</code> , or <code>UnsafeNativeMethods</code> class.

Rule ID and Warning	Description
CA1061: Do not hide base class methods	A method in a base type is hidden by an identically named method in a derived type, when the parameter signature of the derived method differs only by types that are more weakly derived than the corresponding types in the parameter signature of the base method.
CA1062: Validate arguments of public methods	All reference arguments that are passed to externally visible methods should be checked against null.
CA1063: Implement IDisposable correctly	All IDisposable types should implement the Dispose pattern correctly.
CA1064: Exceptions should be public	An internal exception is visible only inside its own internal scope. After the exception falls outside the internal scope, only the base exception can be used to catch the exception. If the internal exception is inherited from Exception , SystemException , or ApplicationException , the external code will not have sufficient information to know what to do with the exception.
CA1065: Do not raise exceptions in unexpected locations	A method that is not expected to throw exceptions throws an exception.
CA1066: Implement IEquatable when overriding Equals	A value type overrides Equals method, but does not implement IEquatable<T> .
CA1067: Override Equals when implementing IEquatable	A type implements IEquatable<T> , but does not override Equals method.
CA1068: CancellationToken parameters must come last	A method has a CancellationToken parameter that is not the last parameter.
CA1069: Enums should not have duplicate values	An enumeration has multiple members which are explicitly assigned the same constant value.
CA1070: Do not declare event fields as virtual	A field-like event was declared as virtual.
CA1200: Avoid using cref tags with a prefix	The <code>cref</code> attribute in an XML documentation tag means "code reference". It specifies that the inner text of the tag is a code element, such as a type, method, or property. Avoid using <code>cref</code> tags with prefixes, because it prevents the compiler from verifying references. It also prevents the Visual Studio integrated development environment (IDE) from finding and updating these symbol references during refactorings.
CA1303: Do not pass literals as localized parameters	An externally visible method passes a string literal as a parameter to a .NET constructor or method, and that string should be localizable.
CA1304: Specify CultureInfo	A method or constructor calls a member that has an overload that accepts a <code>System.Globalization.CultureInfo</code> parameter, and the method or constructor does not call the overload that takes the <code>CultureInfo</code> parameter. When a <code>CultureInfo</code> or <code>System.IFormatProvider</code> object is not supplied, the default value that is supplied by the overloaded member might not have the effect that you want in all locales.

Rule ID and Warning	Description
CA1305: Specify IFormatProvider	A method or constructor calls one or more members that have overloads that accept a System.IFormatProvider parameter, and the method or constructor does not call the overload that takes the IFormatProvider parameter. When a System.Globalization.CultureInfo or IFormatProvider object is not supplied, the default value that is supplied by the overloaded member might not have the effect that you want in all locales.
CA1307: Specify StringComparison for clarity	A string comparison operation uses a method overload that does not set a StringComparison parameter.
CA1308: Normalize strings to uppercase	Strings should be normalized to uppercase. A small group of characters cannot make a round trip when they are converted to lowercase.
CA1309: Use ordinal StringComparison	A string comparison operation that is nonlinguistic does not set the StringComparison parameter to either Ordinal or OrdinalIgnoreCase. By explicitly setting the parameter to either StringComparison.Ordinal or StringComparison.OrdinalIgnoreCase, your code often gains speed, becomes more correct, and becomes more reliable.
CA1310: Specify StringComparison for correctness	A string comparison operation uses a method overload that does not set a StringComparison parameter and uses culture-specific string comparison by default.
CA1401: P/Invokes should not be visible	A public or protected method in a public type has the System.Runtime.InteropServices.DllImportAttribute attribute (also implemented by the Declare keyword in Visual Basic). Such methods should not be exposed.
CA1416: Validate platform compatibility	Using platform-dependent APIs on a component makes the code no longer work across all platforms.
CA1417: Do not use <code>outAttribute</code> on string parameters for P/Invokes	String parameters passed by value with the <code>outAttribute</code> can destabilize the runtime if the string is an interned string.
CA1418: Use valid platform string	Platform compatibility analyzer requires a valid platform name and version.
CA1419: Provide a parameterless constructor that is as visible as the containing type for concrete types derived from 'System.Runtime.InteropServices.SafeHandle'	Providing a parameterless constructor that is as visible as the containing type for a type derived from <code>System.Runtime.InteropServices.SafeHandle</code> enables better performance and usage with source-generated interop solutions.
CA1501: Avoid excessive inheritance	A type is more than four levels deep in its inheritance hierarchy. Deeply nested type hierarchies can be difficult to follow, understand, and maintain.
CA1502: Avoid excessive complexity	This rule measures the number of linearly independent paths through the method, which is determined by the number and complexity of conditional branches.
CA1505: Avoid unmaintainable code	A type or method has a low maintainability index value. A low maintainability index indicates that a type or method is probably difficult to maintain and would be a good candidate for redesign.
CA1506: Avoid excessive class coupling	This rule measures class coupling by counting the number of unique type references that a type or method contains.

RULE ID AND WARNING	DESCRIPTION
CA1507: Use nameof in place of string	A string literal is used as an argument where a <code>nameof</code> expression could be used.
CA1508: Avoid dead conditional code	A method has conditional code that always evaluates to <code>true</code> or <code>false</code> at run time. This leads to dead code in the <code>false</code> branch of the condition.
CA1509: Invalid entry in code metrics configuration file	Code metrics rules, such as CA1501 , CA1502 , CA1505 and CA1506 , supplied a configuration file named <code>CodeMetricsConfig.txt</code> that has an invalid entry.
CA1700: Do not name enum values 'Reserved'	This rule assumes that an enumeration member that has a name that contains "reserved" is not currently used but is a placeholder to be renamed or removed in a future version. Renaming or removing a member is a breaking change.
CA1707: Identifiers should not contain underscores	By convention, identifier names do not contain the underscore (<code>_</code>) character. This rule checks namespaces, types, members, and parameters.
CA1708: Identifiers should differ by more than case	Identifiers for namespaces, types, members, and parameters cannot differ only by case because languages that target the common language runtime are not required to be case-sensitive.
CA1710: Identifiers should have correct suffix	By convention, the names of types that extend certain base types or that implement certain interfaces, or types that are derived from these types, have a suffix that is associated with the base type or interface.
CA1711: Identifiers should not have incorrect suffix	By convention, only the names of types that extend certain base types or that implement certain interfaces, or types that are derived from these types, should end with specific reserved suffixes. Other type names should not use these reserved suffixes.
CA1712: Do not prefix enum values with type name	Names of enumeration members are not prefixed by using the type name because development tools are expected to provide type information.
CA1713: Events should not have before or after prefix	The name of an event starts with "Before" or "After". To name related events that are raised in a specific sequence, use the present or past tense to indicate the relative position in the sequence of actions.
CA1714: Flags enums should have plural names	A public enumeration has the <code>System.FlagsAttribute</code> attribute, and its name does not end in "s". Types that are marked by using <code>FlagsAttribute</code> have names that are plural because the attribute indicates that more than one value can be specified.
CA1715: Identifiers should have correct prefix	The name of an externally visible interface does not start with an uppercase "I". The name of a generic type parameter on an externally visible type or method does not start with an uppercase "T".
CA1716: Identifiers should not match keywords	A namespace name or a type name matches a reserved keyword in a programming language. Identifiers for namespaces and types should not match keywords that are defined by languages that target the common language runtime.

Rule ID and Warning	Description
CA1717: Only FlagsAttribute enums should have plural names	Naming conventions dictate that a plural name for an enumeration indicates that more than one value of the enumeration can be specified at the same time.
CA1720: Identifiers should not contain type names	The name of a parameter in an externally visible member contains a data type name, or the name of an externally visible member contains a language-specific data type name.
CA1721: Property names should not match get methods	The name of a public or protected member starts with "Get" and otherwise matches the name of a public or protected property. "Get" methods and properties should have names that clearly distinguish their function.
CA1724: Type Names Should Not Match Namespaces	Type names should not match the names of .NET namespaces. Violating this rule can reduce the usability of the library.
CA1725: Parameter names should match base declaration	Consistent naming of parameters in an override hierarchy increases the usability of the method overrides. A parameter name in a derived method that differs from the name in the base declaration can cause confusion about whether the method is an override of the base method or a new overload of the method.
CA1727: Use PascalCase for named placeholders	Use PascalCase for named placeholders in the logging message template.
CA1801: Review unused parameters	A method signature includes a parameter that is not used in the method body.
CA1802: Use Literals Where Appropriate	A field is declared static and read-only (Shared and ReadOnly in Visual Basic), and is initialized by using a value that is computable at compile time. Because the value that is assigned to the targeted field is computable at compile time, change the declaration to a const (Const in Visual Basic) field so that the value is computed at compile time instead of at run time.
CA1805: Do not initialize unnecessarily	The .NET runtime initializes all fields of reference types to their default values before running the constructor. In most cases, explicitly initializing a field to its default value is redundant, which adds to maintenance costs and may degrade performance (such as with increased assembly size).
CA1806: Do not ignore method results	A new object is created but never used; or a method that creates and returns a new string is called and the new string is never used; or a COM or P/Invoke method returns an HRESULT or error code that is never used.
CA1810: Initialize reference type static fields inline	When a type declares an explicit static constructor, the just-in-time (JIT) compiler adds a check to each static method and instance constructor of the type to make sure that the static constructor was previously called. Static constructor checks can decrease performance.
CA1812: Avoid uninstantiated internal classes	An instance of an assembly-level type is not created by code in the assembly.
CA1813: Avoid unsealed attributes	.NET provides methods for retrieving custom attributes. By default, these methods search the attribute inheritance hierarchy. Sealing the attribute eliminates the search through the inheritance hierarchy and can improve performance.

Rule ID and Warning	Description
CA1814: Prefer jagged arrays over multidimensional	A jagged array is an array whose elements are arrays. The arrays that make up the elements can be of different sizes, leading to less wasted space for some sets of data.
CA1815: Override equals and operator equals on value types	For value types, the inherited implementation of Equals uses the Reflection library and compares the contents of all fields. Reflection is computationally expensive, and comparing every field for equality might be unnecessary. If you expect users to compare or sort instances, or to use instances as hash table keys, your value type should implement Equals.
CA1816: Call GC.SuppressFinalize correctly	A method that is an implementation of Dispose does not call GC.SuppressFinalize; or a method that is not an implementation of Dispose calls GC.SuppressFinalize; or a method calls GC.SuppressFinalize and passes something other than this (Me in Visual Basic).
CA1819: Properties should not return arrays	Arrays that are returned by properties are not write-protected, even when the property is read-only. To keep the array tamper-proof, the property must return a copy of the array. Typically, users will not understand the adverse performance implications of calling such a property.
CA1820: Test for empty strings using string length	Comparing strings by using the String.Length property or the String.IsNullOrEmpty method is significantly faster than using Equals.
CA1821: Remove empty finalizers	Whenever you can, avoid finalizers because of the additional performance overhead that is involved in tracking object lifetime. An empty finalizer incurs added overhead and delivers no benefit.
CA1822: Mark members as static	Members that do not access instance data or call instance methods can be marked as static (Shared in Visual Basic). After you mark the methods as static, the compiler will emit nonvirtual call sites to these members. This can give you a measurable performance gain for performance-sensitive code.
CA1823: Avoid unused private fields	Private fields were detected that do not appear to be accessed in the assembly.
CA1824: Mark assemblies with NeutralResourcesLanguageAttribute	The NeutralResourcesLanguage attribute informs the resource manager of the language that was used to display the resources of a neutral culture for an assembly. This improves lookup performance for the first resource that you load and can reduce your working set.
CA1825: Avoid zero-length array allocations	Initializing a zero-length array leads to unnecessary memory allocation. Instead, use the statically allocated empty array instance by calling Array.Empty . The memory allocation is shared across all invocations of this method.
CA1826: Use property instead of Linq Enumerable method	Enumerable LINQ method was used on a type that supports an equivalent, more efficient property.
CA1827: Do not use Count/LongCount when Any can be used	Count or LongCount method was used where Any method would be more efficient.
CA1828: Do not use CountAsync/LongCountAsync when AnyAsync can be used	CountAsync or LongCountAsync method was used where AnyAsync method would be more efficient.

RULE ID AND WARNING	DESCRIPTION
CA1829: Use Length/Count property instead of Enumerable.Count method	Count LINQ method was used on a type that supports an equivalent, more efficient <code>Length</code> or <code>Count</code> property.
CA1830: Prefer strongly-typed Append and Insert method overloads on StringBuilder	Append and Insert provide overloads for multiple types beyond <code>String</code> . When possible, prefer the strongly-typed overloads over using <code>ToString()</code> and the string-based overload.
CA1831: Use AsSpan instead of Range-based indexers for string when appropriate	When using a range-indexer on a string and implicitly assigning the value to <code>ReadOnlySpan<char></code> type, the method Substring will be used instead of Slice, which produces a copy of requested portion of the string.
CA1832: Use AsSpan or AsMemory instead of Range-based indexers for getting ReadOnlySpan or ReadOnlyMemory portion of an array	When using a range-indexer on an array and implicitly assigning the value to a <code>ReadOnlySpan<T></code> or <code>ReadOnlyMemory<T></code> type, the method <code>GetSubArray</code> will be used instead of <code>Slice</code> , which produces a copy of requested portion of the array.
CA1833: Use AsSpan or AsMemory instead of Range-based indexers for getting Span or Memory portion of an array	When using a range-indexer on an array and implicitly assigning the value to a <code>Span<T></code> or <code>Memory<T></code> type, the method <code>GetSubArray</code> will be used instead of <code>Slice</code> , which produces a copy of requested portion of the array.
CA1834: Use <code>StringBuilder.Append(char)</code> for single character strings	<code>StringBuilder</code> has an <code>Append</code> overload that takes a <code>char</code> as its argument. Prefer calling the <code>char</code> overload for performance reasons.
CA1835: Prefer the 'Memory'-based overloads for 'ReadAsync' and 'WriteAsync'	'Stream' has a 'ReadAsync' overload that takes a 'Memory<Byte>' as the first argument, and a 'WriteAsync' overload that takes a 'ReadOnlyMemory<Byte>' as the first argument. Prefer calling the memory based overloads, which are more efficient.
CA1836: Prefer <code>IsEmpty</code> over <code>Count</code> when available	Prefer <code>IsEmpty</code> property that is more efficient than <code>Count</code> , <code>Length</code> , <code>Count<TSource>(IEnumerable<TSource>)</code> or <code>LongCount<TSource>(IEnumerable<TSource>)</code> to determine whether the object contains or not any items.
CA1837: Use <code>Environment.ProcessId</code> instead of <code>Process.GetCurrentProcess().Id</code>	<code>Environment.ProcessId</code> is simpler and faster than <code>Process.GetCurrentProcess().Id</code> .
CA1838: Avoid <code>StringBuilder</code> parameters for P/Invokes	Marshalling of 'StringBuilder' always creates a native buffer copy, resulting in multiple allocations for one marshalling operation.
CA1839: Use <code>Environment.ProcessPath</code> instead of <code>Process.GetCurrentProcess().MainModule.FileName</code>	<code>Environment.ProcessPath</code> is simpler and faster than <code>Process.GetCurrentProcess().MainModule.FileName</code> .
CA1840: Use <code>Environment.CurrentManagedThreadId</code> instead of <code>Thread.CurrentThread.ManagedThreadId</code>	<code>Environment.CurrentManagedThreadId</code> is more compact and efficient than <code>Thread.CurrentThread.ManagedThreadId</code> .
CA1841: Prefer Dictionary Contains methods	Calling <code>Contains</code> on the <code>Keys</code> or <code>Values</code> collection may often be more expensive than calling <code>ContainsKey</code> or <code>ContainsValue</code> on the dictionary itself.
CA1842: Do not use 'WhenAll' with a single task	Using <code>WhenAll</code> with a single task may result in performance loss. Await or return the task instead.

RULE ID AND WARNING	DESCRIPTION
CA1843: Do not use 'WaitAll' with a single task	Using <code>WaitAll</code> with a single task may result in performance loss. Await or return the task instead.
CA1844: Provide memory-based overrides of async methods when subclassing 'Stream'	To improve performance, override the memory-based async methods when subclassing 'Stream'. Then implement the array-based methods in terms of the memory-based methods.
CA1845: Use span-based 'string.Concat'	It is more efficient to use <code>AsSpan</code> and <code>string.Concat</code> , instead of <code>Substring</code> and a concatenation operator.
CA1846: Prefer <code>AsSpan</code> over <code>Substring</code>	<code>AsSpan</code> is more efficient than <code>Substring</code> . <code>Substring</code> performs an O(n) string copy, while <code>AsSpan</code> does not and has a constant cost. <code>AsSpan</code> also does not perform any heap allocations.
CA1847: Use char literal for a single character lookup	Use <code>string.Contains(char)</code> instead of <code>string.Contains(string)</code> when searching for a single character.
CA1848: Use the <code>LoggerMessage</code> delegates	For improved performance, use the <code>LoggerMessage</code> delegates.
CA1849: Call async methods when in an async method	In a method which is already asynchronous, calls to other methods should be to their async versions, where they exist.
CA1850: Prefer static <code>HashData</code> method over <code>ComputeHash</code>	It's more efficient to use the static <code>HashData</code> method over creating and managing a <code>HashAlgorithm</code> instance to call <code>ComputeHash</code> .
CA1851: Possible multiple enumerations of <code>IEnumerable</code> collection	Possible multiple enumerations of <code>IEnumerable</code> collection. Consider using an implementation that avoids multiple enumerations.
CA1854: Prefer the 'IDictionary.TryGetValue(TKey, out TValue)' method	Prefer 'TryGetValue' over a Dictionary indexer access guarded by a 'ContainsKey' check. 'ContainsKey' and the indexer both look up the key, so using 'TryGetValue' avoids the extra lookup.
CA2000: Dispose objects before losing scope	Because an exceptional event might occur that will prevent the finalizer of an object from running, the object should be explicitly disposed before all references to it are out of scope.
CA2002: Do not lock on objects with weak identity	An object is said to have a weak identity when it can be directly accessed across application domain boundaries. A thread that tries to acquire a lock on an object that has a weak identity can be blocked by a second thread in a different application domain that has a lock on the same object.
CA2007: Do not directly await a Task	An asynchronous method <code>awaits</code> a <code>Task</code> directly. When an asynchronous method awaits a <code>Task</code> directly, continuation occurs in the same thread that created the task. This behavior can be costly in terms of performance and can result in a deadlock on the UI thread. Consider calling <code>Task.ConfigureAwait(Boolean)</code> to signal your intention for continuation.
CA2008: Do not create tasks without passing a <code>TaskScheduler</code>	A task creation or continuation operation uses a method overload that does not specify a <code>TaskScheduler</code> parameter.

RULE ID AND WARNING	DESCRIPTION
CA2009: Do not call <code>ToImmutableCollection</code> on an <code>ImmutableCollection</code> value	<code>ToImmutable</code> method was unnecessarily called on an immutable collection from <code>System.Collections.Immutable</code> namespace.
CA2011: Do not assign property within its setter	A property was accidentally assigned a value within its own <code>set accessor</code> .
CA2012: Use <code>ValueTasks</code> correctly	ValueTasks returned from member invocations are intended to be directly awaited. Attempts to consume a <code>ValueTask</code> multiple times or to directly access one's result before it's known to be completed may result in an exception or corruption. Ignoring such a <code>ValueTask</code> is likely an indication of a functional bug and may degrade performance.
CA2013: Do not use <code>ReferenceEquals</code> with value types	When comparing values using <code>System.Object.ReferenceEquals</code> , if <code>objA</code> and <code>objB</code> are value types, they are boxed before they are passed to the <code>ReferenceEquals</code> method. This means that even if both <code>objA</code> and <code>objB</code> represent the same instance of a value type, the <code>ReferenceEquals</code> method nevertheless returns false.
CA2014: Do not use <code>stackalloc</code> in loops.	Stack space allocated by a <code>stackalloc</code> is only released at the end of the current method's invocation. Using it in a loop can result in unbounded stack growth and eventual stack overflow conditions.
CA2015: Do not define finalizers for types derived from <code>MemoryManager<T></code>	Adding a finalizer to a type derived from <code>MemoryManager<T></code> may permit memory to be freed while it is still in use by a <code>Span<T></code> .
CA2016: Forward the <code>CancellationToken</code> parameter to methods that take one	Forward the <code>CancellationToken</code> parameter to methods that take one to ensure the operation cancellation notifications gets properly propagated, or pass in <code>CancellationToken.None</code> explicitly to indicate intentionally not propagating the token.
CA2017: Parameter count mismatch	Number of parameters supplied in the logging message template do not match the number of named placeholders.
CA2018: The <code>count</code> argument to <code>Buffer.BlockCopy</code> should specify the number of bytes to copy	When using <code>Buffer.BlockCopy</code> , the <code>count</code> argument specifies the number of bytes to copy. You should only use <code>Array.Length</code> for the <code>count</code> argument on arrays whose elements are exactly one byte in size. <code>byte</code> , <code>sbyte</code> , and <code>bool</code> arrays have elements that are one byte in size.
CA2100: Review SQL queries for security vulnerabilities	A method sets the <code>System.Data.IDbCommand.CommandText</code> property by using a string that is built from a string argument to the method. This rule assumes that the string argument contains user input. A SQL command string that is built from user input is vulnerable to SQL injection attacks.
CA2101: Specify marshalling for P/Invoke string arguments	A platform invoke member allows partially trusted callers, has a string parameter, and does not explicitly marshal the string. This can cause a potential security vulnerability.
CA2109: Review visible event handlers	A public or protected event-handling method was detected. Event-handling methods should not be exposed unless absolutely necessary.
CA2119: Seal methods that satisfy private interfaces	An inheritable public type provides an overridable method implementation of an internal (Friend in Visual Basic) interface. To fix a violation of this rule, prevent the method from being overridden outside the assembly.

RULE ID AND WARNING	DESCRIPTION
CA2153: Avoid handling Corrupted State Exceptions	Corrupted State Exceptions (CSEs) indicate that memory corruption exists in your process. Catching these rather than allowing the process to crash can lead to security vulnerabilities if an attacker can place an exploit into the corrupted memory region.
CA2200: Rethrow to preserve stack details	An exception is rethrown and the exception is explicitly specified in the throw statement. If an exception is rethrown by specifying the exception in the throw statement, the list of method calls between the original method that threw the exception and the current method is lost.
CA2201: Do not raise reserved exception types	This makes the original error difficult to detect and debug.
CA2207: Initialize value type static fields inline	A value type declares an explicit static constructor. To fix a violation of this rule, initialize all static data when it is declared and remove the static constructor.
CA2208: Instantiate argument exceptions correctly	A call is made to the default (parameterless) constructor of an exception type that is or derives from ArgumentException, or an incorrect string argument is passed to a parameterized constructor of an exception type that is or derives from ArgumentException.
CA2211: Non-constant fields should not be visible	Static fields that are neither constants nor read-only are not thread-safe. Access to such a field must be carefully controlled and requires advanced programming techniques to synchronize access to the class object.
CA2213: Disposable fields should be disposed	A type that implements System.IDisposable declares fields that are of types that also implement IDisposable. The Dispose method of the field is not called by the Dispose method of the declaring type.
CA2214: Do not call overridable methods in constructors	When a constructor calls a virtual method, the constructor for the instance that invokes the method may not have executed.
CA2215: Dispose methods should call base class dispose	If a type inherits from a disposable type, it must call the Dispose method of the base type from its own Dispose method.
CA2216: Disposable types should declare finalizer	A type that implements System.IDisposable and has fields that suggest the use of unmanaged resources does not implement a finalizer, as described by Object.Finalize.
CA2217: Do not mark enums with FlagsAttribute	An externally visible enumeration is marked by using FlagsAttribute, and it has one or more values that are not powers of two or a combination of the other defined values on the enumeration.
CA2218: Override GetHashCode on overriding Equals	A public type overrides System.Object.Equals but does not override System.Object.GetHashCode .
CA2219: Do not raise exceptions in exception clauses	When an exception is raised in a finally or fault clause, the new exception hides the active exception. When an exception is raised in a filter clause, the run time silently catches the exception. This makes the original error difficult to detect and debug.
CA2224: Override equals on overloading operator equals	A public type implements the equality operator but doesn't override System.Object.Equals .

Rule ID and Warning	Description
CA2225: Operator overloads have named alternates	An operator overload was detected, and the expected named alternative method was not found. The named alternative member provides access to the same functionality as the operator and is provided for developers who program in languages that do not support overloaded operators.
CA2226: Operators should have symmetrical overloads	A type implements the equality or inequality operator and does not implement the opposite operator.
CA2227: Collection properties should be read only	A writable collection property allows a user to replace the collection with a different collection. A read-only property stops the collection from being replaced but still allows the individual members to be set.
CA2229: Implement serialization constructors	To fix a violation of this rule, implement the serialization constructor. For a sealed class, make the constructor private; otherwise, make it protected.
CA2231: Overload operator equals on overriding ValueType.Equals	A value type overrides Object.Equals but does not implement the equality operator.
CA2234: Pass System.Uri objects instead of strings	A call is made to a method that has a string parameter whose name contains "uri", "URI", "urn", "URN", "url", or "URL". The declaring type of the method contains a corresponding method overload that has a System.Uri parameter.
CA2235: Mark all non-serializable fields	An instance field of a type that is not serializable is declared in a type that is serializable.
CA2237: Mark ISerializable types with SerializableAttribute	To be recognized by the common language runtime as serializable, types must be marked by using the SerializableAttribute attribute even when the type uses a custom serialization routine through implementation of the ISerializable interface.
CA2241: Provide correct arguments to formatting methods	The format argument that is passed to System.String.Format does not contain a format item that corresponds to each object argument, or vice versa.
CA2242: Test for NaN correctly	This expression tests a value against Single.NaN or Double.NaN. Use Single.IsNaN(Single) or Double.IsNaN(Double) to test the value.
CA2243: Attribute string literals should parse correctly	The string literal parameter of an attribute does not parse correctly for a URL, a GUID, or a version.
CA2244: Do not duplicate indexed element initializations	An object initializer has more than one indexed element initializer with the same constant index. All but the last initializer are redundant.
CA2245: Do not assign a property to itself	A property was accidentally assigned to itself.
CA2246: Do not assign a symbol and its member in the same statement	Assigning a symbol and its member, that is, a field or a property, in the same statement is not recommended. It is not clear if the member access was intended to use the symbol's old value prior to the assignment or the new value from the assignment in this statement.

RULE ID AND WARNING	DESCRIPTION
CA2247: Argument passed to TaskCompletionSource constructor should be TaskCreationOptions enum instead of TaskContinuationOptions enum.	TaskCompletionSource has constructors that take TaskCreationOptions that control the underlying Task, and constructors that take object state that's stored in the task. Accidentally passing a TaskContinuationOptions instead of a TaskCreationOptions will result in the call treating the options as state.
CA2248: Provide correct enum argument to Enum.HasFlag	The enum type passed as an argument to the <code>HasFlag</code> method call is different from the calling enum type.
CA2249: Consider using String.Contains instead of String.IndexOf	Calls to <code>string.IndexOf</code> where the result is used to check for the presence/absence of a substring can be replaced by <code>string.Contains</code> .
CA2250: Use <code>ThrowIfCancellationRequested</code>	<code>ThrowIfCancellationRequested</code> automatically checks whether the token has been canceled, and throws an <code>OperationCanceledException</code> if it has.
CA2251: Use <code>String.Equals</code> over <code>String.Compare</code>	It is both clearer and likely faster to use <code>String.Equals</code> instead of comparing the result of <code>String.Compare</code> to zero.
CA2252: Opt in to preview features	Opt in to preview features before using preview APIs.
CA2253: Named placeholders should not be numeric values	Named placeholders in the logging message template should not be comprised of only numeric characters.
CA2254: Template should be a static expression	The logging message template should not vary between calls.
CA2255: The <code>ModuleInitializer</code> attribute should not be used in libraries	Module initializers are intended to be used by application code to ensure an application's components are initialized before the application code begins executing.
CA2256: All members declared in parent interfaces must have an implementation in a DynamicInterfaceCastableImplementation-attributed interface	Types attributed with <code>DynamicInterfaceCastableImplementationAttribute</code> act as an interface implementation for a type that implements the <code>IDynamicInterfaceCastable</code> type. As a result, it must provide an implementation of all of the members defined in the inherited interfaces, because the type that implements <code>IDynamicInterfaceCastable</code> will not provide them otherwise.
CA2257: Members defined on an interface with 'DynamicInterfaceCastableImplementationAttribute' should be 'static'	Since a type that implements <code>IDynamicInterfaceCastable</code> may not implement a dynamic interface in metadata, calls to an instance interface member that is not an explicit implementation defined on this type are likely to fail at run time. Mark new interface members <code>static</code> to avoid run-time errors.
CA2258: Providing a 'DynamicInterfaceCastableImplementation' interface in Visual Basic is unsupported	Providing a functional <code>DynamicInterfaceCastableImplementationAttribute</code> -attributed interface requires the Default Interface Members feature, which is unsupported in Visual Basic.
CA2300: Do not use insecure deserializer BinaryFormatter	Insecure deserializers are vulnerable when deserializing untrusted data. An attacker could modify the serialized data to include unexpected types to inject objects with malicious side effects.

Rule ID and Warning	Description
CA2301: Do not call <code>BinaryFormatter.Deserialize</code> without first setting <code>BinaryFormatterBinder</code>	Insecure deserializers are vulnerable when deserializing untrusted data. An attacker could modify the serialized data to include unexpected types to inject objects with malicious side effects.
CA2302: Ensure <code>BinaryFormatterBinder</code> is set before calling <code>BinaryFormatter.Deserialize</code>	Insecure deserializers are vulnerable when deserializing untrusted data. An attacker could modify the serialized data to include unexpected types to inject objects with malicious side effects.
CA2305: Do not use insecure deserializer <code>LoSFormatter</code>	Insecure deserializers are vulnerable when deserializing untrusted data. An attacker could modify the serialized data to include unexpected types to inject objects with malicious side effects.
CA2310: Do not use insecure deserializer <code>NetDataContractSerializer</code>	Insecure deserializers are vulnerable when deserializing untrusted data. An attacker could modify the serialized data to include unexpected types to inject objects with malicious side effects.
CA2311: Do not deserialize without first setting <code>NetDataContractSerializer.Binder</code>	Insecure deserializers are vulnerable when deserializing untrusted data. An attacker could modify the serialized data to include unexpected types to inject objects with malicious side effects.
CA2312: Ensure <code>NetDataContractSerializer.Binder</code> is set before deserializing	Insecure deserializers are vulnerable when deserializing untrusted data. An attacker could modify the serialized data to include unexpected types to inject objects with malicious side effects.
CA2315: Do not use insecure deserializer <code>ObjectStateFormatter</code>	Insecure deserializers are vulnerable when deserializing untrusted data. An attacker could modify the serialized data to include unexpected types to inject objects with malicious side effects.
CA2321: Do not deserialize with <code>JavaScriptSerializer</code> using a <code>SimpleTypeResolver</code>	Insecure deserializers are vulnerable when deserializing untrusted data. An attacker could modify the serialized data to include unexpected types to inject objects with malicious side effects.
CA2322: Ensure <code>JavaScriptSerializer</code> is not initialized with <code>SimpleTypeResolver</code> before deserializing	Insecure deserializers are vulnerable when deserializing untrusted data. An attacker could modify the serialized data to include unexpected types to inject objects with malicious side effects.
CA2326: Do not use <code>TypeNameHandling</code> values other than <code>None</code>	Insecure deserializers are vulnerable when deserializing untrusted data. An attacker could modify the serialized data to include unexpected types to inject objects with malicious side effects.
CA2327: Do not use insecure <code>JsonSerializerSettings</code>	Insecure deserializers are vulnerable when deserializing untrusted data. An attacker could modify the serialized data to include unexpected types to inject objects with malicious side effects.
CA2328: Ensure that <code>JsonSerializerSettings</code> are secure	Insecure deserializers are vulnerable when deserializing untrusted data. An attacker could modify the serialized data to include unexpected types to inject objects with malicious side effects.

Rule ID and Warning	Description
CA2329: Do not deserialize with JsonSerializer using an insecure configuration	Insecure deserializers are vulnerable when deserializing untrusted data. An attacker could modify the serialized data to include unexpected types to inject objects with malicious side effects.
CA2330: Ensure that JsonSerializer has a secure configuration when deserializing	Insecure deserializers are vulnerable when deserializing untrusted data. An attacker could modify the serialized data to include unexpected types to inject objects with malicious side effects.
CA2350: Ensure DataTable.ReadXml()'s input is trusted	When deserializing a DataTable with untrusted input, an attacker can craft malicious input to perform a denial of service attack. There may be unknown remote code execution vulnerabilities.
CA2351: Ensure DataSet.ReadXml()'s input is trusted	When deserializing a DataSet with untrusted input, an attacker can craft malicious input to perform a denial of service attack. There may be unknown remote code execution vulnerabilities.
CA2352: Unsafe DataSet or DataTable in serializable type can be vulnerable to remote code execution attacks	A class or struct marked with SerializableAttribute contains a DataSet or DataTable field or property, and doesn't have a GeneratedCodeAttribute .
CA2353: Unsafe DataSet or DataTable in serializable type	A class or struct marked with an XML serialization attribute or a data contract attribute contains a DataSet or DataTable field or property.
CA2354: Unsafe DataSet or DataTable in deserialized object graph can be vulnerable to remote code execution attack	Deserializing with an System.Runtime.Serialization.IFormatter serialized, and the casted type's object graph can include a DataSet or DataTable .
CA2355: Unsafe DataSet or DataTable in serialized object graph	Deserializing when the casted or specified type's object graph can include a DataSet or DataTable .
CA2356: Unsafe DataSet or DataTable in web deserialized object graph	A method with a System.Web.Services.WebMethodAttribute or System.ServiceModel.OperationContractAttribute has a parameter that may reference a DataSet or DataTable .
CA2361: Ensure autogenerated class containing DataSet.ReadXml() is not used with untrusted data	When deserializing a DataSet with untrusted input, an attacker can craft malicious input to perform a denial of service attack. There may be unknown remote code execution vulnerabilities.
CA2362: Unsafe DataSet or DataTable in autogenerated serializable type can be vulnerable to remote code execution attacks	When deserializing untrusted input with BinaryFormatter and the deserialized object graph contains a DataSet or DataTable , an attacker can craft a malicious payload to perform a remote code execution attack.
CA3001: Review code for SQL injection vulnerabilities	When working with untrusted input and SQL commands, be mindful of SQL injection attacks. An SQL injection attack can execute malicious SQL commands, compromising the security and integrity of your application.
CA3002: Review code for XSS vulnerabilities	When working with untrusted input from web requests, be mindful of cross-site scripting (XSS) attacks. An XSS attack injects untrusted input into raw HTML output, allowing the attacker to execute malicious scripts or maliciously modify content in your web page.
CA3003: Review code for file path injection vulnerabilities	When working with untrusted input from web requests, be mindful of using user-controlled input when specifying paths to files.

Rule ID and Warning	Description
CA3004: Review code for information disclosure vulnerabilities	Disclosing exception information gives attackers insight into the internals of your application, which can help attackers find other vulnerabilities to exploit.
CA3006: Review code for process command injection vulnerabilities	When working with untrusted input, be mindful of command injection attacks. A command injection attack can execute malicious commands on the underlying operating system, compromising the security and integrity of your server.
CA3007: Review code for open redirect vulnerabilities	When working with untrusted input, be mindful of open redirect vulnerabilities. An attacker can exploit an open redirect vulnerability to use your website to give the appearance of a legitimate URL, but redirect an unsuspecting visitor to a phishing or other malicious webpage.
CA3008: Review code for XPath injection vulnerabilities	When working with untrusted input, be mindful of XPath injection attacks. Constructing XPath queries using untrusted input may allow an attacker to maliciously manipulate the query to return an unintended result, and possibly disclose the contents of the queried XML.
CA3009: Review code for XML injection vulnerabilities	When working with untrusted input, be mindful of XML injection attacks.
CA3010: Review code for XAML injection vulnerabilities	When working with untrusted input, be mindful of XAML injection attacks. XAML is a markup language that directly represents object instantiation and execution. That means elements created in XAML can interact with system resources (for example, network access and file system IO).
CA3011: Review code for DLL injection vulnerabilities	When working with untrusted input, be mindful of loading untrusted code. If your web application loads untrusted code, an attacker may be able to inject malicious DLLs into your process and execute malicious code.
CA3012: Review code for regex injection vulnerabilities	When working with untrusted input, be mindful of regex injection attacks. An attacker can use regex injection to maliciously modify a regular expression, to make the regex match unintended results, or to make the regex consume excessive CPU resulting in a Denial of Service attack.
CA3061: Do not add schema by URL	Do not use the unsafe overload of the Add method because it may cause dangerous external references.
CA3075: Insecure DTD Processing	If you use insecure DTDProcessing instances or reference external entity sources, the parser may accept untrusted input and disclose sensitive information to attackers.
CA3076: Insecure XSLT Script Execution	If you execute Extensible Stylesheet Language Transformations (XSLT) in .NET applications insecurely, the processor may resolve untrusted URI references that could disclose sensitive information to attackers, leading to Denial of Service and Cross-Site attacks.
CA3077: Insecure Processing in API Design, XML Document and XML Text Reader	When designing an API derived from XmlDocument and XmlTextReader, be mindful of DtdProcessing. Using insecure DTDProcessing instances when referencing or resolving external entity sources or setting insecure values in the XML may lead to information disclosure.

Rule ID and Warning	Description
CA3147: Mark verb handlers with ValidateAntiForgeryToken	When designing an ASP.NET MVC controller, be mindful of cross-site request forgery attacks. A cross-site request forgery attack can send malicious requests from an authenticated user to your ASP.NET MVC controller.
CA5350: Do Not Use Weak Cryptographic Algorithms	Weak encryption algorithms and hashing functions are used today for a number of reasons, but they should not be used to guarantee the confidentiality or integrity of the data they protect. This rule triggers when it finds TripleDES, SHA1, or RIPEMD160 algorithms in the code.
CA5351 Do Not Use Broken Cryptographic Algorithms	Broken cryptographic algorithms are not considered secure and their use should be strongly discouraged. This rule triggers when it finds the MD5 hash algorithm or either the DES or RC2 encryption algorithms in code.
CA5358: Do Not Use Unsafe Cipher Modes	Do Not Use Unsafe Cipher Modes
CA5359 Do not disable certificate validation	A certificate can help authenticate the identity of the server. Clients should validate the server certificate to ensure requests are sent to the intended server. If the <code>ServerCertificateValidationCallback</code> always returns <code>true</code> , any certificate will pass validation.
CA5360 Do not call dangerous methods in deserialization	Insecure deserialization is a vulnerability which occurs when untrusted data is used to abuse the logic of an application, inflict a Denial-of-Service (DoS) attack, or even execute arbitrary code upon it being deserialized. It's frequently possible for malicious users to abuse these deserialization features when the application is deserializing untrusted data which is under their control. Specifically, invoke dangerous methods in the process of deserialization. Successful insecure deserialization attacks could allow an attacker to carry out attacks such as DoS attacks, authentication bypasses, and remote code execution.
CA5361: Do not disable Schannel use of strong crypto	Setting <code>Switch.System.Net.DontEnableSchUseStrongCrypto</code> to <code>true</code> weakens the cryptography used in outgoing Transport Layer Security (TLS) connections. Weaker cryptography can compromise the confidentiality of communication between your application and the server, making it easier for attackers to eavesdrop sensitive data.
CA5362 Potential reference cycle in deserialized object graph	If deserializing untrusted data, then any code processing the deserialized object graph needs to handle reference cycles without going into infinite loops. This includes both code that's part of a deserialization callback and code that processes the object graph after deserialization completed. Otherwise, an attacker could perform a Denial-of-Service attack with malicious data containing a reference cycle.
CA5363: Do not disable request validation	Request validation is a feature in ASP.NET that examines HTTP requests and determines whether they contain potentially dangerous content that can lead to injection attacks, including cross-site-scripting.
CA5364: Do not use deprecated security protocols	Transport Layer Security (TLS) secures communication between computers, most commonly with Hypertext Transfer Protocol Secure (HTTPS). Older protocol versions of TLS are less secure than TLS 1.2 and TLS 1.3 and are more likely to have new vulnerabilities. Avoid older protocol versions to minimize risk.

Rule ID and Warning	Description
CA5365 Do Not Disable HTTP Header Checking	HTTP header checking enables encoding of the carriage return and newline characters, \r and \n, that are found in response headers. This encoding can help to avoid injection attacks that exploit an application that echoes untrusted data contained by the header.
CA5366 Use XmlReader For DataSet Read XML	Using a DataSet to read XML with untrusted data may load dangerous external references, which should be restricted by using an XmlReader with a secure resolver or with DTD processing disabled.
CA5367 Do Not Serialize Types With Pointer Fields	This rule checks whether there's a serializable class with a pointer field or property. Members that can't be serialized can be a pointer, such as static members or fields marked with NonSerializedAttribute .
CA5368 Set ViewStateUserKey For Classes Derived From Page	Setting the ViewStateUserKey property can help you prevent attacks on your application by allowing you to assign an identifier to the view-state variable for individual users so that attackers cannot use the variable to generate an attack. Otherwise, there will be vulnerabilities to cross-site request forgery.
CA5369: Use XmlReader for Deserialize	Processing untrusted DTD and XML schemas may enable loading dangerous external references, which should be restricted by using an XmlReader with a secure resolver or with DTD and XML inline schema processing disabled.
CA5370: Use XmlReader for validating reader	Processing untrusted DTD and XML schemas may enable loading dangerous external references. This dangerous loading can be restricted by using an XmlReader with a secure resolver or with DTD and XML inline schema processing disabled.
CA5371: Use XmlReader for schema read	Processing untrusted DTD and XML schemas may enable loading dangerous external references. Using an XmlReader with a secure resolver or with DTD and XML inline schema processing disabled restricts this.
CA5372: Use XmlReader for XPathDocument	Processing XML from untrusted data may load dangerous external references, which can be restricted by using an XmlReader with a secure resolver or with DTD processing disabled.
CA5373: Do not use obsolete key derivation function	This rule detects the invocation of weak key derivation methods System.Security.Cryptography.PasswordDeriveBytes and Rfc2898DeriveBytes.CryptDeriveKey . System.Security.Cryptography.PasswordDeriveBytes used a weak algorithm PBKDF1.
CA5374 Do Not Use XslTransform	This rule checks if System.Xml.Xsl.XslTransform is instantiated in the code. System.Xml.Xsl.XslTransform is now obsolete and shouldn't be used.
CA5375 Do not use account shared access signature	An account SAS can delegate access to read, write, and delete operations on blob containers, tables, queues, and file shares that are not permitted with a service SAS. However, it doesn't support container-level policies and has less flexibility and control over the permissions that are granted. Once malicious users get it, your storage account will be compromised easily.
CA5376 Use SharedAccessProtocolHttpsOnly	SAS is sensitive data that can't be transported in plain text on HTTP.

Rule ID and Warning	Description
CA5377 Use container level access policy	A container-level access policy can be modified or revoked at any time. It provides greater flexibility and control over the permissions that are granted.
CA5378: Do not disable ServicePointManagerSecurityProtocols	<p>Setting</p> <pre>Switch.System.ServiceModel.DisableUsingServicePointManagerSecurityProtocol to true limits Windows Communication Framework's (WCF) Transport Layer Security (TLS) connections to using TLS 1.0. That version of TLS will be deprecated.</pre>
CA5379 Do not use weak key derivation function algorithm	The Rfc2898DeriveBytes class defaults to using the SHA1 algorithm. You should specify the hash algorithm to use in some overloads of the constructor with SHA256 or higher. Note, HashAlgorithm property only has a <code>get</code> accessor and doesn't have a <code>overridden</code> modifier.
CA5380: Do not add certificates to root store	This rule detects code that adds a certificate into the Trusted Root Certification Authorities certificate store. By default, the Trusted Root Certification Authorities certificate store is configured with a set of public CAs that has met the requirements of the Microsoft Root Certificate Program.
CA5381: Ensure certificates are not added to root store	This rule detects code that potentially adds a certificate into the Trusted Root Certification Authorities certificate store. By default, the Trusted Root Certification Authorities certificate store is configured with a set of public certification authorities (CAs) that has met the requirements of the Microsoft Root Certificate Program.
CA5382 Use secure cookies in ASP.NET Core	Applications available over HTTPS must use secure cookies, which indicate to the browser that the cookie should only be transmitted using Secure Sockets Layer (SSL).
CA5383 Ensure use secure cookies in ASP.NET Core	Applications available over HTTPS must use secure cookies, which indicate to the browser that the cookie should only be transmitted using Secure Sockets Layer (SSL).
CA5384 Do not use digital signature algorithm (DSA)	DSA is a weak asymmetric encryption algorithm.
CA5385 Use Rivest–Shamir–Adleman (RSA) algorithm with sufficient key size	An RSA key smaller than 2048 bits is more vulnerable to brute force attacks.
CA5386: Avoid hardcoding SecurityProtocolType value	Transport Layer Security (TLS) secures communication between computers, most commonly with Hypertext Transfer Protocol Secure (HTTPS). Protocol versions TLS 1.0 and TLS 1.1 are deprecated, while TLS 1.2 and TLS 1.3 are current. In the future, TLS 1.2 and TLS 1.3 may be deprecated. To ensure that your application remains secure, avoid hardcoding a protocol version and target at least .NET Framework v4.7.1.
CA5387 Do not use weak key derivation function with insufficient iteration count	This rule checks if a cryptographic key was generated by Rfc2898DeriveBytes with an iteration count of less than 100,000. A higher iteration count can help mitigate against dictionary attacks that try to guess the generated cryptographic key.
CA5388 Ensure sufficient iteration count when using weak key derivation function	This rule checks if a cryptographic key was generated by Rfc2898DeriveBytes with an iteration count that may be less than 100,000. A higher iteration count can help mitigate against dictionary attacks that try to guess the generated cryptographic key.

Rule ID and Warning	Description
CA5389: Do not add archive item's path to the target file system path	File path can be relative and can lead to file system access outside of the expected file system target path, leading to malicious config changes and remote code execution via lay-and-wait technique.
CA5390 Do not hard-code encryption key	For a symmetric algorithm to be successful, the secret key must be known only to the sender and the receiver. When a key is hard-coded, it is easily discovered. Even with compiled binaries, it is easy for malicious users to extract it. Once the private key is compromised, the cipher text can be decrypted directly and is not protected anymore.
CA5391 Use antiforgery tokens in ASP.NET Core MVC controllers	Handling a <code>POST</code> , <code>PUT</code> , <code>PATCH</code> , or <code>DELETE</code> request without validating an antiforgery token may be vulnerable to cross-site request forgery attacks. A cross-site request forgery attack can send malicious requests from an authenticated user to your ASP.NET Core MVC controller.
CA5392 Use DefaultDllImportSearchPaths attribute for P/Invokes	By default, P/Invoke functions using <code>DllImportAttribute</code> probe a number of directories, including the current working directory for the library to load. This can be a security issue for certain applications, leading to DLL hijacking.
CA5393 Do not use unsafe DllImportSearchPath value	There could be a malicious DLL in the default DLL search directories and assembly directories. Or, depending on where your application is run from, there could be a malicious DLL in the application's directory.
CA5394 Do not use insecure randomness	Using a cryptographically weak pseudo-random number generator may allow an attacker to predict what security-sensitive value will be generated.
CA5395 Miss <code>HttpVerb</code> attribute for action methods	All the action methods that create, edit, delete, or otherwise modify data needs to be protected with the antiforgery attribute from cross-site request forgery attacks. Performing a GET operation should be a safe operation that has no side effects and doesn't modify your persisted data.
CA5396 Set <code>HttpOnly</code> to true for <code>HttpCookie</code>	As a defense in depth measure, ensure security sensitive HTTP cookies are marked as <code>HttpOnly</code> . This indicates web browsers should disallow scripts from accessing the cookies. Injected malicious scripts are a common way of stealing cookies.
CA5397: Do not use deprecated <code>SslProtocols</code> values	Transport Layer Security (TLS) secures communication between computers, most commonly with Hypertext Transfer Protocol Secure (HTTPS). Older protocol versions of TLS are less secure than TLS 1.2 and TLS 1.3 and are more likely to have new vulnerabilities. Avoid older protocol versions to minimize risk.
CA5398: Avoid hardcoded <code>SslProtocols</code> values	Transport Layer Security (TLS) secures communication between computers, most commonly with Hypertext Transfer Protocol Secure (HTTPS). Protocol versions TLS 1.0 and TLS 1.1 are deprecated, while TLS 1.2 and TLS 1.3 are current. In the future, TLS 1.2 and TLS 1.3 may be deprecated. To ensure that your application remains secure, avoid hardcoding a protocol version.
CA5399 Definitely disable <code>HttpClient</code> certificate revocation list check	A revoked certificate isn't trusted anymore. It could be used by attackers passing some malicious data or stealing sensitive data in HTTPS communication.

RULE ID AND WARNING	DESCRIPTION
CA5400 Ensure HttpClient certificate revocation list check is not disabled	A revoked certificate isn't trusted anymore. It could be used by attackers passing some malicious data or stealing sensitive data in HTTPS communication.
CA5401 Do not use CreateEncryptor with non-default IV	Symmetric encryption should always use a non-repeatable initialization vector to prevent dictionary attacks.
CA5402 Use CreateEncryptor with the default IV	Symmetric encryption should always use a non-repeatable initialization vector to prevent dictionary attacks.
CA5403: Do not hard-code certificate	The <code>data</code> or <code>rawData</code> parameter of a <code>X509Certificate</code> or <code>X509Certificate2</code> constructor is hard-coded.
CA5404: Do not disable token validation checks	<code>TokenValidationParameters</code> properties that control token validation should not be set to <code>false</code> .
CA5405: Do not always skip token validation in delegates	The callback assigned to <code>AudienceValidator</code> or <code>LifetimeValidator</code> always returns <code>true</code> .
IL3000 Avoid accessing Assembly file path when publishing as a single file	Avoid accessing Assembly file path when publishing as a single file.
IL3001 Avoid accessing Assembly file path when publishing as a single-file	Avoid accessing Assembly file path when publishing as a single file.
IL3002 Avoid calling members annotated with 'RequiresAssemblyFilesAttribute' when publishing as a single file	Avoid calling members annotated with 'RequiresAssemblyFilesAttribute' when publishing as a single file
IL3003 'RequiresAssemblyFilesAttribute' annotations must match across all interface implementations or overrides.	'RequiresAssemblyFilesAttribute' annotations must match across all interface implementations or overrides.

Legend

The following table shows the type of information that is provided for each rule in the reference documentation.

ITEM	DESCRIPTION
Type	The TypeName for the rule.
Rule ID	The unique identifier for the rule. RuleId and Category are used for in-source suppression of a warning.
Category	The category of the rule, for example, security.
Fix is breaking or non-breaking	Whether the fix for a violation of the rule is a breaking change. Breaking change means that an assembly that has a dependency on the target that caused the violation will not recompile with the new fixed version or might fail at run time because of the change. When multiple fixes are available and at least one fix is a breaking change and one fix is not, both 'Breaking' and 'Non-breaking' are specified.
Cause	The specific managed code that causes the rule to generate a warning.
Description	Discusses the issues that are behind the warning.
How to fix violations	Explains how to change the source code to satisfy the rule and prevent it from generating a warning.

ITEM	DESCRIPTION
When to suppress warnings	Describes when it is safe to suppress a warning from the rule.
Example code	Examples that violate the rule and corrected examples that satisfy the rule.
Related rules	Related rules.

Design rules

9/20/2022 • 10 minutes to read • [Edit Online](#)

Design rules support adherence to the [.NET Framework design guidelines](#).

In this section

RULE	DESCRIPTION
CA1000: Do not declare static members on generic types	When a static member of a generic type is called, the type argument must be specified for the type. When a generic instance member that does not support inference is called, the type argument must be specified for the member. In these two cases, the syntax for specifying the type argument is different and easily confused.
CA1001: Types that own disposable fields should be disposable	A class declares and implements an instance field that is a System.IDisposable type and the class does not implement IDisposable. A class that declares an IDisposable field indirectly owns an unmanaged resource and should implement the IDisposable interface.
CA1002: Do not expose generic lists	System.Collections.Generic.List<(Of <(T>)>) is a generic collection that is designed for performance, not inheritance. Therefore, List does not contain any virtual members. The generic collections that are designed for inheritance should be exposed instead.
CA1003: Use generic event handler instances	A type contains a delegate that returns void, whose signature contains two parameters (the first an object and the second a type that is assignable to EventArgs), and the containing assembly targets .NET Framework 2.0.
CA1005: Avoid excessive parameters on generic types	The more type parameters a generic type contains, the more difficult it is to know and remember what each type parameter represents. It is usually obvious with one type parameter, as in List<T>, and in certain cases with two type parameters, as in Dictionary< TKey, TValue >. However, if more than two type parameters exist, the difficulty becomes too great for most users.
CA1008: Enums should have zero value	The default value of an uninitialized enumeration, just as other value types, is zero. A nonflags attributed enumeration should define a member by using the value of zero so that the default value is a valid value of the enumeration. If an enumeration that has the FlagsAttribute attribute applied defines a zero-valued member, its name should be "None" to indicate that no values have been set in the enumeration.
CA1010: Collections should implement generic interface	To broaden the usability of a collection, implement one of the generic collection interfaces. Then the collection can be used to populate generic collection types.

Rule	Description
CA1012: Abstract types should not have constructors	Constructors on abstract types can be called only by derived types. Because public constructors create instances of a type, and you cannot create instances of an abstract type, an abstract type that has a public constructor is incorrectly designed.
CA1014: Mark assemblies with CLSCompliantAttribute	The Common Language Specification (CLS) defines naming restrictions, data types, and rules to which assemblies must conform if they will be used across programming languages. Good design dictates that all assemblies explicitly indicate CLS compliance by using CLSCompliantAttribute. If this attribute is not present on an assembly, the assembly is not compliant.
CA1016: Mark assemblies with AssemblyVersionAttribute	.NET uses the version number to uniquely identify an assembly, and to bind to types in strongly named assemblies. The version number is used together with version and publisher policy. By default, applications run only with the assembly version with which they were built.
CA1017: Mark assemblies with ComVisibleAttribute	ComVisibleAttribute determines how COM clients access managed code. Good design dictates that assemblies explicitly indicate COM visibility. COM visibility can be set for the whole assembly and then overridden for individual types and type members. If this attribute is not present, the contents of the assembly are visible to COM clients.
CA1018: Mark attributes with AttributeUsageAttribute	When you define a custom attribute, mark it by using AttributeUsageAttribute to indicate where in the source code the custom attribute can be applied. The meaning and intended usage of an attribute will determine its valid locations in code.
CA1019: Define accessors for attribute arguments	Attributes can define mandatory arguments that must be specified when you apply the attribute to a target. These are also known as positional arguments because they are supplied to attribute constructors as positional parameters. For every mandatory argument, the attribute should also provide a corresponding read-only property so that the value of the argument can be retrieved at execution time. Attributes can also define optional arguments, which are also known as named arguments. These arguments are supplied to attribute constructors by name and should have a corresponding read/write property.
CA1021: Avoid out parameters	Passing types by reference (using out or ref) requires experience with pointers, understanding how value types and reference types differ, and handling methods with multiple return values. Also, the difference between out and ref parameters is not widely understood.
CA1024: Use properties where appropriate	A public or protected method has a name that starts with "Get", takes no parameters, and returns a value that is not an array. The method might be a good candidate to become a property.

Rule	Description
CA1027: Mark enums with FlagsAttribute	An enumeration is a value type that defines a set of related named constants. Apply FlagsAttribute to an enumeration when its named constants can be meaningfully combined.
CA1028: Enum storage should be Int32	An enumeration is a value type that defines a set of related named constants. By default, the System.Int32 data type is used to store the constant value. Even though you can change this underlying type, it is not required or recommended for most scenarios.
CA1030: Use events where appropriate	This rule detects methods that have names that ordinarily would be used for events. If a method is called in response to a clearly defined state change, the method should be invoked by an event handler. Objects that call the method should raise events instead of calling the method directly.
CA1031: Do not catch general exception types	General exceptions should not be caught. Catch a more-specific exception, or rethrow the general exception as the last statement in the catch block.
CA1032: Implement standard exception constructors	Failure to provide the full set of constructors can make it difficult to correctly handle exceptions.
CA1033: Interface methods should be callable by child types	An unsealed externally visible type provides an explicit method implementation of a public interface and does not provide an alternative externally visible method that has the same name.
CA1034: Nested types should not be visible	A nested type is a type that is declared in the scope of another type. Nested types are useful to encapsulate private implementation details of the containing type. Used for this purpose, nested types should not be externally visible.
CA1036: Override methods on comparable types	A public or protected type implements the System.IComparable interface. It does not override Object.Equals nor does it overload the language-specific operator for equality, inequality, less than, or greater than.
CA1040: Avoid empty interfaces	Interfaces define members that provide a behavior or usage contract. The functionality that is described by the interface can be adopted by any type, regardless of where the type appears in the inheritance hierarchy. A type implements an interface by providing implementations for the members of the interface. An empty interface does not define any members; therefore, it does not define a contract that can be implemented.
CA1041: Provide ObsoleteAttribute message	A type or member is marked by using a System.ObsoleteAttribute attribute that does not have its ObsoleteAttribute.Message property specified. When a type or member that is marked by using ObsoleteAttribute is compiled, the Message property of the attribute is displayed, which gives the user information about the obsolete type or member.

Rule	Description
CA1043: Use integral or string argument for indexers	Indexers (that is, indexed properties) should use integral or string types for the index. These types are typically used for indexing data structures and they increase the usability of the library. Use of the Object type should be restricted to those cases where the specific integral or string type cannot be specified at design time.
CA1044: Properties should not be write only	Although it is acceptable and often necessary to have a read-only property, the design guidelines prohibit the use of write-only properties. This is because letting a user set a value, and then preventing the user from viewing that value, does not provide any security. Also, without read access, the state of shared objects cannot be viewed, which limits their usefulness.
CA1045: Do not pass types by reference	Passing types by reference (using out or ref) requires experience with pointers, understanding how value types and reference types differ, and handling methods with multiple return values. Library architects who design for a general audience should not expect users to become proficient in working with out or ref parameters.
CA1046: Do not overload operator equals on reference types	For reference types, the default implementation of the equality operator is almost always correct. By default, two references are equal only if they point to the same object.
CA1047: Do not declare protected members in sealed types	Types declare protected members so that inheriting types can access or override the member. By definition, sealed types cannot be inherited, which means that protected methods on sealed types cannot be called.
CA1050: Declare types in namespaces	Types are declared in namespaces to prevent name collisions and as a way to organize related types in an object hierarchy.
CA1051: Do not declare visible instance fields	The primary use of a field should be as an implementation detail. Fields should be private or internal and should be exposed by using properties.
CA1052: Static holder types should be sealed	A public or protected type contains only static members and is not declared by using the sealed (C#) or NotInheritable (Visual Basic) modifier. A type that is not meant to be inherited should be marked by using the sealed modifier to prevent its use as a base type.
CA1053: Static holder types should not have constructors	A public or nested public type declares only static members and has a public or protected default constructor. The constructor is unnecessary because calling static members does not require an instance of the type. The string overload should call the uniform resource identifier (URI) overload by using the string argument for safety and security.
CA1054: URI parameters should not be strings	If a method takes a string representation of a URI, a corresponding overload should be provided that takes an instance of the Uri class, which provides these services in a safe and secure manner.

Rule	Description
CA1055: URI return values should not be strings	This rule assumes that the method returns a URI. A string representation of a URI is prone to parsing and encoding errors, and can lead to security vulnerabilities. The System.Uri class provides these services in a safe and secure manner.
CA1056: URI properties should not be strings	This rule assumes that the property represents a URI. A string representation of a URI is prone to parsing and encoding errors, and can lead to security vulnerabilities. The System.Uri class provides these services in a safe and secure manner.
CA1058: Types should not extend certain base types	An externally visible type extends certain base types. Use one of the alternatives.
CA1060: Move P/Invokes to NativeMethods class	Platform Invocation methods, such as those marked with the System.Runtime.InteropServices.DllImportAttribute or methods defined by using the Declare keyword in Visual Basic, access unmanaged code. These methods should be of the NativeMethods, SafeNativeMethods, or UnsafeNativeMethods class.
CA1061: Do not hide base class methods	A method in a base type is hidden by an identically named method in a derived type, when the parameter signature of the derived method differs only by types that are more weakly derived than the corresponding types in the parameter signature of the base method.
CA1062: Validate arguments of public methods	All reference arguments that are passed to externally visible methods should be checked against null.
CA1063: Implement IDisposable correctly	All IDisposable types should implement the Dispose pattern correctly.
CA1064: Exceptions should be public	An internal exception is visible only inside its own internal scope. After the exception falls outside the internal scope, only the base exception can be used to catch the exception. If the internal exception is inherited from System.Exception , System.SystemException , or System.ApplicationException , the external code will not have sufficient information to know what to do with the exception.
CA1065: Do not raise exceptions in unexpected locations	A method that is not expected to throw exceptions throws an exception.
CA1066: Implement IEquatable when overriding Equals	A value type overrides Equals method, but does not implement IEquatable<T> .
CA1067: Override Equals when implementing IEquatable	A type implements IEquatable<T> , but does not override Equals method.
CA1068: CancellationToken parameters must come last	A method has a CancellationToken parameter that is not the last parameter.
CA1069: Enums should not have duplicate values	An enumeration has multiple members which are explicitly assigned the same constant value.

RULE	DESCRIPTION
CA1070: Do not declare event fields as virtual	A field-like event was declared as virtual.

CA1000: Do not declare static members on generic types

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA1000
Category	Design
Fix is breaking or non-breaking	Breaking

Cause

A generic type contains a `static` (`Shared` in Visual Basic) member.

By default, this rule only looks at externally visible types, but this is [configurable](#).

Rule description

When a `static` member of a generic type is called, the type argument must be specified for the type. When a generic instance member that does not support inference is called, the type argument must be specified for the member. The syntax for specifying the type argument in these two cases is different and easily confused, as the following calls demonstrate:

```
' Shared method in a generic type.  
GenericType(Of Integer).SharedMethod()  
  
' Generic instance method that does not support inference.  
someObject.GenericMethod(Of Integer)()
```

```
// Static method in a generic type.  
GenericType<int>.StaticMethod();  
  
// Generic instance method that does not support inference.  
someObject.GenericMethod<int>();
```

Generally, both of the prior declarations should be avoided so that the type argument does not have to be specified when the member is called. This results in a syntax for calling members in generics that is no different from the syntax for non-generics.

How to fix violations

To fix a violation of this rule, remove the static member or change it to an instance member.

When to suppress warnings

Do not suppress a warning from this rule. Providing generics in a syntax that is easy to understand and use reduces the time that is required to learn and increases the adoption rate of new libraries.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1000
// The code that's violating the rule is on this line.
#pragma warning restore CA1000
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.cs,vb]
dotnet_diagnostic.CA1000.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.cs,vb]
dotnet_analyzer_diagnostic.category-Design.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

Use the following option to configure which parts of your codebase to run this rule on.

- [Include specific API surfaces](#)

You can configure this option for just this rule, for all rules, or for all rules in this category ([Design](#)). For more information, see [Code quality rule configuration options](#).

Include specific API surfaces

You can configure which parts of your codebase to run this rule on, based on their accessibility. For example, to specify that the rule should run only against the non-public API surface, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXX.api_surface = private, internal
```

Related rules

- [CA1005: Avoid excessive parameters on generic types](#)
- [CA1010: Collections should implement generic interface](#)
- [CA1002: Do not expose generic lists](#)
- [CA1003: Use generic event handler instances](#)

See also

- [Generics](#)

CA1001: Types that own disposable fields should be disposable

9/20/2022 • 4 minutes to read • [Edit Online](#)

	Value
Rule ID	CA1001
Category	Design
Fix is breaking or non-breaking	Non-breaking - If the type is not visible outside the assembly. Breaking - If the type is visible outside the assembly.

Cause

A class declares and implements an instance field that is an [System.IDisposable](#) type, and the class does not implement [IDisposable](#).

By default, this rule analyzes the entire codebase, but this is [configurable](#).

Rule description

A class that declares an [IDisposable](#) field indirectly owns an unmanaged resource. The class should implement the [IDisposable](#) interface to dispose of the unmanaged resource that it owns once the resource is no longer in use. If the class does not *directly* own any unmanaged resources, it should not implement a finalizer.

This rule respects types implementing [System.IAsyncDisposable](#) as disposable types.

How to fix violations

To fix a violation of this rule, implement the [IDisposable](#) interface. In the [IDisposable.Dispose](#) method, call the [Dispose](#) method of the field's type.

When to suppress warnings

In general, do not suppress a warning from this rule. It's okay to suppress the warning when the dispose ownership of the field is not held by the containing type.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1001
// The code that's violating the rule is on this line.
#pragma warning restore CA1001
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1001.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Design.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

Use the following options to configure which parts of your codebase to run this rule on.

- [Exclude specific symbols](#)
- [Exclude specific types and their derived types](#)

These options can be configured for just this rule, for all rules, or for all rules in this category ([Design](#)). For more information, see [Code quality rule configuration options](#).

Exclude specific symbols

You can exclude specific symbols, such as types and methods, from analysis. For example, to specify that the rule should not run on any code within types named `MyType`, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Symbol name only (includes all symbols with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#). Each symbol name requires a symbol-kind prefix, such as `M:` for methods, `T:` for types, and `N:` for namespaces.
- `.ctor` for constructors and `.cctor` for static constructors.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType</code>	Matches all symbols named <code>MyType</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType1 MyType2</code>	Matches all symbols named either <code>MyType1</code> or <code>MyType2</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS.MyType.MyMethod(ParamType)</code>	Matches specific method <code>MyMethod</code> with the specified fully qualified signature.
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS1.MyType1.MyMethod1(ParamType) M:NS2.MyType2.MyMethod2(ParamType)</code>	Matches specific methods <code>MyMethod1</code> and <code>MyMethod2</code> with the respective fully qualified signatures.

Exclude specific types and their derived types

You can exclude specific types and their derived types from analysis. For example, to specify that the rule should

not run on any methods within types named `MyType` and their derived types, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Type name only (includes all types with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#), with an optional `T:` prefix.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType</code>	Matches all types named <code>MyType</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType1 MyType2</code>	Matches all types named either <code>MyType1</code> or <code>MyType2</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS.MyType</code>	Matches specific type <code>MyType</code> with given fully qualified name and all of its derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS1.MyType1 M:NS2.MyType2</code>	Matches specific types <code>MyType1</code> and <code>MyType2</code> with the respective fully qualified names, and all of their derived types.

Example

The following example shows a class that violates the rule and a class that satisfies the rule by implementing [IDisposable](#). The class does not implement a finalizer because the class does not directly own any unmanaged resources.

```
Imports System
Imports System.IO

Namespace ca1001

    ' This class violates the rule.
    Public Class NoDisposeMethod

        Dim newFile As FileStream

        Sub New()
            newFile = New FileStream("c:\temp.txt", FileMode.Open)
        End Sub

    End Class

    ' This class satisfies the rule.
    Public Class HasDisposeMethod
        Implements IDisposable

        Dim newFile As FileStream

        Sub New()
            newFile = New FileStream("c:\temp.txt", FileMode.Open)
        End Sub

        Protected Overridable Overloads Sub Dispose(disposing As Boolean)

            If disposing Then
                ' dispose managed resources
                newFile.Close()
            End If

            ' free native resources

        End Sub 'Dispose

        Public Overloads Sub Dispose() Implements IDisposable.Dispose

            Dispose(True)
            GC.SuppressFinalize(Me)

        End Sub 'Dispose

    End Class

End Namespace
```

```
// This class violates the rule.
public class NoDisposeMethod
{
    FileStream _newFile;

    public NoDisposeMethod()
    {
        _newFile = new FileStream(@"c:\temp.txt", FileMode.Open);
    }
}

// This class satisfies the rule.
public class HasDisposeMethod : IDisposable
{
    FileStream _newFile;

    public HasDisposeMethod()
    {
        _newFile = new FileStream(@"c:\temp.txt", FileMode.Open);
    }

    protected virtual void Dispose(bool disposing)
    {
        if (disposing)
        {
            // Dispose managed resources.
            _newFile.Close();
        }
        // Free native resources.
    }

    public void Dispose()
    {
        Dispose(true);
        GC.SuppressFinalize(this);
    }
}
```

Related rules

- [CA2213: Disposable fields should be disposed](#)
- [CA2216: Disposable types should declare finalizer](#)
- [CA2215: Dispose methods should call base class dispose](#)

See also

- [Implement a Dispose method](#)

CA1002: Do not expose generic lists

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA1002
Category	Design
Fix is breaking or non-breaking	Breaking

Cause

A type contains an externally visible member that is a `System.Collections.Generic.List<T>` type, returns a `List<T>` type, or whose signature includes a `List<T>` parameter.

By default, this rule only looks at externally visible types, but this is [configurable](#).

Rule description

`System.Collections.Generic.List<T>` is a generic collection that's designed for performance and not inheritance. `List<T>` does not contain virtual members that make it easier to change the behavior of an inherited class. The following generic collections are designed for inheritance and should be exposed instead of `List<T>`.

- `System.Collections.ObjectModel.Collection<T>`
- `System.Collections.ObjectModel.ReadOnlyCollection<T>`
- `System.Collections.ObjectModel.KeyedCollection<TKey,TItem>`
- `System.Collections.Generic.IList<T>`
- `System.Collections.Generic.ICollection<T>`

How to fix violations

To fix a violation of this rule, change the `System.Collections.Generic.List<T>` type to one of the generic collections that's designed for inheritance.

When to suppress warnings

Do not suppress a warning from this rule unless the assembly that raises this warning is not meant to be a reusable library. For example, it would be safe to suppress this warning in a performance-tuned application where a performance benefit was gained from the use of generic lists.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1002
// The code that's violating the rule is on this line.
#pragma warning restore CA1002
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1002.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Design.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

Use the following option to configure which parts of your codebase to run this rule on.

- [Include specific API surfaces](#)

You can configure this option for just this rule, for all rules, or for all rules in this category ([Design](#)). For more information, see [Code quality rule configuration options](#).

Include specific API surfaces

You can configure which parts of your codebase to run this rule on, based on their accessibility. For example, to specify that the rule should run only against the non-public API surface, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.api_surface = private, internal
```

Related rules

[CA1005: Avoid excessive parameters on generic types](#)

[CA1010: Collections should implement generic interface](#)

[CA1000: Do not declare static members on generic types](#)

[CA1003: Use generic event handler instances](#)

See also

[Generics](#)

CA1003: Use generic event handler instances

9/20/2022 • 3 minutes to read • [Edit Online](#)

ITEM	VALUE
RuleId	CA1003
Category	Design
Fix is breaking or non-breaking	Breaking

Cause

A type contains a delegate that returns void and whose signature contains two parameters (the first an object and the second a type that is assignable to EventArgs), and the containing assembly targets .NET.

By default, this rule only looks at externally visible types, but this is [configurable](#).

Rule description

Before .NET Framework 2.0, in order to pass custom information to the event handler, a new delegate had to be declared that specified a class that was derived from the [System.EventArgs](#) class. In .NET Framework 2.0 and later versions, the generic [System.EventHandler<TEventArgs>](#) delegate allows any class that's derived from [EventArgs](#) to be used together with the event handler.

How to fix violations

To fix a violation of this rule, remove the delegate and replace its use by using the [System.EventHandler<TEventArgs>](#) delegate.

If the delegate is autogenerated by the Visual Basic compiler, change the syntax of the event declaration to use the [System.EventHandler<TEventArgs>](#) delegate.

When to suppress warnings

Do not suppress a warning from this rule.

Configure code to analyze

Use the following option to configure which parts of your codebase to run this rule on.

- [Include specific API surfaces](#)

You can configure this option for just this rule, for all rules, or for all rules in this category ([Design](#)). For more information, see [Code quality rule configuration options](#).

Include specific API surfaces

You can configure which parts of your codebase to run this rule on, based on their accessibility. For example, to specify that the rule should run only against the non-public API surface, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXX.api_surface = private, internal
```

Example

The following example shows a delegate that violates the rule. In the Visual Basic example, comments describe how to modify the example to satisfy the rule. For the C# example, an example follows that shows the modified code.

```

Imports System

Namespace ca1003

    Public Class CustomEventArgs
        Inherits EventArgs

        Public info As String = "data"

    End Class

    Public Class ClassThatRaisesEvent

        ' This statement creates a new delegate, which violates the rule.
        Event SomeEvent(sender As Object, e As CustomEventArgs)

        ' To satisfy the rule, comment out the previous line
        ' and uncomment the following line.
        'Event SomeEvent As EventHandler(Of CustomEventArgs)

        Protected Overridable Sub OnSomeEvent(e As CustomEventArgs)
            RaiseEvent SomeEvent(Me, e)
        End Sub

        Sub SimulateEvent()
            OnSomeEvent(New CustomEventArgs())
        End Sub

    End Class

    Public Class ClassThatHandlesEvent

        Sub New(eventRaiser As ClassThatRaisesEvent)
            AddHandler eventRaiser.SomeEvent, AddressOf HandleEvent
        End Sub

        Private Sub HandleEvent(sender As Object, e As CustomEventArgs)
            Console.WriteLine("Event handled: {0}", e.info)
        End Sub

    End Class

    Class Test

        Shared Sub Main()

            Dim eventRaiser As New ClassThatRaisesEvent()
            Dim eventHandler As New ClassThatHandlesEvent(eventRaiser)

            eventRaiser.SimulateEvent()

        End Sub

    End Class

End Namespace

```

```

// This delegate violates the rule.
public delegate void CustomEventHandler(object sender, CustomEventArgs e);

public class CustomEventArgs : EventArgs
{
    public string info = "data";
}

public class ClassThatRaisesEvent
{
    public event CustomEventHandler SomeEvent;

    protected virtual void OnSomeEvent(CustomEventArgs e)
    {
        SomeEvent?.Invoke(this, e);
    }

    public void SimulateEvent()
    {
        OnSomeEvent(new CustomEventArgs());
    }
}

public class ClassThatHandlesEvent
{
    public ClassThatHandlesEvent(ClassThatRaisesEvent eventRaiser)
    {
        eventRaiser.SomeEvent += new CustomEventHandler(HandleEvent);
    }

    private void HandleEvent(object sender, CustomEventArgs e)
    {
        Console.WriteLine("Event handled: {0}", e.info);
    }
}

class Test
{
    static void MainEvent()
    {
        var eventRaiser = new ClassThatRaisesEvent();
        var eventHandler = new ClassThatHandlesEvent(eventRaiser);

        eventRaiser.SimulateEvent();
    }
}

```

The following code snippet removes the delegate declaration from the previous example, which satisfies the rule. It replaces its use in the `ClassThatRaisesEvent` and `ClassThatHandlesEvent` methods by using the `System.EventHandler<TEventArgs>` delegate.

```

public class CustomEventArgs : EventArgs
{
    public string info = "data";
}

public class ClassThatRaisesEvent
{
    public event EventHandler<CustomEventArgs> SomeEvent;

    protected virtual void OnSomeEvent(CustomEventArgs e)
    {
        SomeEvent?.Invoke(this, e);
    }

    public void SimulateEvent()
    {
        OnSomeEvent(new CustomEventArgs());
    }
}

public class ClassThatHandlesEvent
{
    public ClassThatHandlesEvent(ClassThatRaisesEvent eventRaiser)
    {
        eventRaiser.SomeEvent += new EventHandler<CustomEventArgs>(HandleEvent);
    }

    private void HandleEvent(object sender, CustomEventArgs e)
    {
        Console.WriteLine("Event handled: {0}", e.info);
    }
}

class Test
{
    static void MainEvent()
    {
        var eventRaiser = new ClassThatRaisesEvent();
        var eventHandler = new ClassThatHandlesEvent(eventRaiser);

        eventRaiser.SimulateEvent();
    }
}

```

Related rules

- [CA1005: Avoid excessive parameters on generic types](#)
- [CA1010: Collections should implement generic interface](#)
- [CA1000: Do not declare static members on generic types](#)
- [CA1002: Do not expose generic lists](#)

See also

- [Generics](#)

CA1005: Avoid excessive parameters on generic types

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA1005
Category	Design
Fix is breaking or non-breaking	Breaking

Cause

An externally visible generic type has more than two type parameters.

By default, this rule only looks at externally visible types, but this is [configurable](#).

Rule description

The more type parameters a generic type contains, the more difficult it is to know and remember what each type parameter represents. It is usually obvious with one type parameter, as in `List<T>`, and in certain cases with two type parameters, as in `Dictionary< TKey, TValue >`. If more than two type parameters exist, the difficulty becomes too great for most users (for example, `TooManyTypeParameters<T, K, v>` in C# or `TooManyTypeParameters(Of T, K, V)` in Visual Basic).

How to fix violations

To fix a violation of this rule, change the design to use no more than two type parameters.

When to suppress warnings

Do not suppress a warning from this rule unless the design absolutely requires more than two type parameters. Providing generics in a syntax that is easy to understand and use reduces the time that is required to learn and increases the adoption rate of new libraries.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1005
// The code that's violating the rule is on this line.
#pragma warning restore CA1005
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1005.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Design.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

Use the following option to configure which parts of your codebase to run this rule on.

- [Include specific API surfaces](#)

You can configure this option for just this rule, for all rules, or for all rules in this category ([Design](#)). For more information, see [Code quality rule configuration options](#).

Include specific API surfaces

You can configure which parts of your codebase to run this rule on, based on their accessibility. For example, to specify that the rule should run only against the non-public API surface, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXX.api_surface = private, internal
```

Related rules

[CA1010: Collections should implement generic interface](#)

[CA1000: Do not declare static members on generic types](#)

[CA1002: Do not expose generic lists](#)

[CA1003: Use generic event handler instances](#)

See also

- [Generics](#)

CA1008: Enums should have zero value

9/20/2022 • 4 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA1008
Category	Design
Fix is breaking or non-breaking	Non-breaking - When you're prompted to add a <code>None</code> value to a non-flag enumeration. Breaking - When you're prompted to rename or remove any enumeration values.

Cause

An enumeration without an applied `System.FlagsAttribute` does not define a member that has a value of zero. Or, an enumeration that has an applied `FlagsAttribute` defines a member that has a value of zero but its name is not 'None'. Or, the enumeration defines multiple, zero-valued members.

By default, this rule only looks at externally visible enumerations, but this is [configurable](#).

Rule description

The default value of an uninitialized enumeration, just like other value types, is zero. A non-flags-attributed enumeration should define a member that has the value of zero so that the default value is a valid value of the enumeration. If appropriate, name the member 'None' (or one of the [additional permitted names](#)). Otherwise, assign zero to the most frequently used member. By default, if the value of the first enumeration member is not set in the declaration, its value is zero.

If an enumeration that has the `FlagsAttribute` applied defines a zero-valued member, its name should be 'None' (or one of the [additional permitted names](#)) to indicate that no values have been set in the enumeration. Using a zero-valued member for any other purpose is contrary to the use of the `FlagsAttribute` in that the `AND` and `OR` bitwise operators are useless with the member. This implies that only one member should be assigned the value zero. If multiple members that have the value zero occur in a flags-attributed enumeration, `Enum.ToString()` returns incorrect results for members that are not zero.

How to fix violations

To fix a violation of this rule for non-flags-attributed enumerations, define a member that has the value of zero; this is a non-breaking change. For flags-attributed enumerations that define a zero-valued member, name this member 'None' and delete any other members that have a value of zero; this is a breaking change.

When to suppress warnings

Do not suppress a warning from this rule except for flags-attributed enumerations that have previously shipped.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1008
// The code that's violating the rule is on this line.
#pragma warning restore CA1008
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1008.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Design.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

Use the following option to configure which parts of your codebase to run this rule on.

- [Include specific API surfaces](#)
- [Additional zero-value field names](#)

You can configure this option for just this rule, for all rules, or for all rules in this category (Design). For more information, see [Code quality rule configuration options](#).

Include specific API surfaces

You can configure which parts of your codebase to run this rule on, based on their accessibility. For example, to specify that the rule should run only against the non-public API surface, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXX.api_surface = private, internal
```

Additional zero-value field names

In .NET 7 and later versions, you can configure other allowable names for a zero-value enumeration field, besides `None`. Separate multiple names by the `|` character. The following table shows some examples.

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CA1008.additional_enum_none_names = Never</code>	Allows both <code>None</code> and <code>Never</code>
<code>dotnet_code_quality.CA1008.additional_enum_none_names = Never Nothing</code>	Allows <code>None</code> , <code>Never</code> , and <code>Nothing</code>

Example

The following example shows two enumerations that satisfy the rule and an enumeration, `BadTraceOptions`, that violates the rule.

```
using System;

namespace ca1008
{
    public enum TraceLevel
    {
        Off = 0,
        Error = 1,
        Warning = 2,
        Info = 3,
        Verbose = 4
    }

    [Flags]
    public enum TraceOptions
    {
        None = 0,
        CallStack = 0x01,
        LogicalStack = 0x02,
        DateTime = 0x04,
        Timestamp = 0x08,
    }

    [Flags]
    public enum BadTraceOptions
    {
        CallStack = 0,
        LogicalStack = 0x01,
        DateTime = 0x02,
        Timestamp = 0x04,
    }

    class UseBadTraceOptions
    {
        static void MainTrace()
        {
            // Set the flags.
            BadTraceOptions badOptions =
                BadTraceOptions.LogicalStack | BadTraceOptions.Timestamp;

            // Check whether CallStack is set.
            if ((badOptions & BadTraceOptions.CallStack) ==
                BadTraceOptions.CallStack)
            {
                // This 'if' statement is always true.
            }
        }
    }
}
```

```

Imports System

Namespace ca1008

    Public Enum TraceLevel
        Off = 0
        AnError = 1
        Warning = 2
        Info = 3
        Verbose = 4
    End Enum

    <Flags>
    Public Enum TraceOptions
        None = 0
        CallStack = &H1
        LogicalStack = &H2
        DateTime = &H4
        Timestamp = &H8
    End Enum

    <Flags>
    Public Enum BadTraceOptions
        CallStack = 0
        LogicalStack = &H1
        DateTime = &H2
        Timestamp = &H4
    End Enum

    Class UseBadTraceOptions

        Shared Sub Main1008()

            ' Set the flags.
            Dim badOptions As BadTraceOptions =
                BadTraceOptions.LogicalStack Or BadTraceOptions.Timestamp

            ' Check whether CallStack is set.
            If ((badOptions And BadTraceOptions.CallStack) =
                BadTraceOptions.CallStack) Then
                ' This 'If' statement is always true.
            End If

        End Sub

    End Class

End Namespace

```

Related rules

- [CA2217: Do not mark enums with FlagsAttribute](#)
- [CA1700: Do not name enum values 'Reserved'](#)
- [CA1712: Do not prefix enum values with type name](#)
- [CA1028: Enum storage should be Int32](#)
- [CA1027: Mark enums with FlagsAttribute](#)

See also

- [System.Enum](#)

CA1010: Collections should implement generic interface

9/20/2022 • 4 minutes to read • [Edit Online](#)

ITEM	VALUE
RuleId	CA1010
Category	Design
Fix is breaking or non-breaking	Non-breaking

Cause

A type implements the [System.Collections.IEnumerable](#) interface but does not implement the [System.Collections.Generic.IEnumerable<T>](#) interface, and the containing assembly targets .NET. This rule ignores types that implement [System.Collections.IDictionary](#).

By default, this rule only looks at externally visible types, but this is [configurable](#). You can also configure additional interfaces to require that a generic interface be implemented.

Rule description

To broaden the usability of a collection, implement one of the generic collection interfaces. Then the collection can be used to populate generic collection types such as the following:

- [System.Collections.Generic.List<T>](#)
- [System.Collections.Generic.Queue<T>](#)
- [System.Collections.Generic.Stack<T>](#)

How to fix violations

To fix a violation of this rule, implement one of the following generic collection interfaces:

- [System.Collections.Generic.IEnumerable<T>](#)
- [System.Collections.Generic.ICollection<T>](#)
- [System.Collections.Generic.IList<T>](#)

When to suppress warnings

It is safe to suppress a warning from this rule; however, the use of the collection will be more limited.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1010
// The code that's violating the rule is on this line.
#pragma warning restore CA1010
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.cs,vb]
dotnet_diagnostic.CA1010.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.cs,vb]
dotnet_analyzer_diagnostic.category-Design.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

Use the following options to configure which parts of your codebase to run this rule on.

- [Include specific API surfaces](#)
- [Additional required generic interfaces](#)

You can configure these options for just this rule, for all rules, or for all rules in this category ([Design](#)). For more information, see [Code quality rule configuration options](#).

Include specific API surfaces

You can configure which parts of your codebase to run this rule on, based on their accessibility. For example, to specify that the rule should run only against the non-public API surface, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.api_surface = private, internal
```

Additional required generic interfaces

You can configure the list of interface names (separated by `|`) with their required generic fully qualified interface (separated by `->`).

Allowed interface formats:

- Interface name only (includes all interfaces with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#) with an optional `T:` prefix.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CA1010.additional_required_generic_interfaces = ISomething->System.Collections.Generic.IEnumerable`1</code>	All types that implement <code>ISomething</code> regardless of its namespace are expected to also implement <code>System.Collections.Generic.IEnumerable<T></code> .
<code>dotnet_code_quality.CA1010.additional_required_generic_interfaces = T:System.Collections.IDictionary->T:System.Collections.Generic.IDictionary`2</code>	All types that implement <code>System.Collections.IDictionary</code> are expected to also implement <code>System.Collections.Generic.IDictionary<TKey, TValue></code> .

Example

The following example shows a class that derives from the non-generic `CollectionBase` class and violates this rule.

```
public class Book
{
    public Book()
    {
    }
}

public class BookCollection : CollectionBase
{
    public BookCollection()
    {
    }

    public void Add(Book value)
    {
        InnerList.Add(value);
    }

    public void Remove(Book value)
    {
        InnerList.Remove(value);
    }

    public void Insert(int index, Book value)
    {
        InnerList.Insert(index, value);
    }

    public Book this[int index]
    {
        get { return (Book)InnerList[index]; }
        set { InnerList[index] = value; }
    }

    public bool Contains(Book value)
    {
        return InnerList.Contains(value);
    }

    public int IndexOf(Book value)
    {
        return InnerList.IndexOf(value);
    }

    public void CopyTo(Book[] array, int arrayIndex)
    {
        InnerList.CopyTo(array, arrayIndex);
    }
}
```

To fix a violation of this rule, do one of the following:

- [Implement the generic interface](#).
- [Change the base class](#) to a type that already implements both the generic and non-generic interfaces, such as the `Collection<T>` class.

Fix by interface implementation

The following example fixes the violation by implementing these generic interfaces: `IEnumerable<T>`, `ICollection<T>`, and `IList<T>`.

```
public class Book
{
    public Book()
    {
    }
}

public class BookCollection : CollectionBase, IList<Book>
{
    public BookCollection()
    {
    }

    int IList<Book>.IndexOf(Book item)
    {
        return this.List.IndexOf(item);
    }

    void IList<Book>.Insert(int location, Book item)
    {
    }

    Book IList<Book>.this[int index]
    {
        get { return (Book)this.List[index]; }
        set { }
    }

    void ICollection<Book>.Add(Book item)
    {
    }

    bool ICollection<Book>.Contains(Book item)
    {
        return true;
    }

    void ICollection<Book>.CopyTo(Book[] array, int arrayIndex)
    {
    }

    bool ICollection<Book>.IsReadOnly
    {
        get { return false; }
    }

    bool ICollection<Book>.Remove(Book item)
    {
        if (InnerList.Contains(item))
        {
            InnerList.Remove(item);
            return true;
        }
        return false;
    }

    IEnumerator<Book> IEnumerable<Book>.GetEnumerator()
    {
        return new BookCollectionEnumerator(InnerList.GetEnumerator());
    }

    private class BookCollectionEnumerator : IEnumerator<Book>
    {
        private IEnumerator _Enumerator;

        public BookCollectionEnumerator(IEnumerable enumerator)
        {
            _Enumerator = enumerator;
        }
    }
}
```

```

    }

    public Book Current
    {
        get { return (Book)_Enumerator.Current; }
    }

    object IEnumerator.Current
    {
        get { return _Enumerator.Current; }
    }

    public bool MoveNext()
    {
        return _Enumerator.MoveNext();
    }

    public void Reset()
    {
        _Enumerator.Reset();
    }

    public void Dispose()
    {
    }
}
}

```

Fix by base class change

The following example fixes the violation by changing the base class of the collection from the non-generic `CollectionBase` class to the generic `Collection<T>` (`Collection(Of T)` in Visual Basic) class.

```

public class Book
{
    public Book()
    {
    }
}

public class BookCollection : Collection<Book>
{
    public BookCollection()
    {
    }
}

```

Changing the base class of an already released class is considered a breaking change to existing consumers.

Related rules

- [CA1005: Avoid excessive parameters on generic types](#)
- [CA1000: Do not declare static members on generic types](#)
- [CA1002: Do not expose generic lists](#)
- [CA1003: Use generic event handler instances](#)

See also

- [Generics](#)

CA1012: Abstract types should not have public constructors

9/20/2022 • 2 minutes to read • [Edit Online](#)

ITEM	VALUE
RuleId	CA1012
Category	Design
Fix is breaking or non-breaking	Non-breaking

Cause

A type is abstract and has a public constructor.

By default, this rule only looks at externally visible types, but this is [configurable](#).

Rule description

Constructors on abstract types can be called only by derived types. Because public constructors create instances of a type and you cannot create instances of an abstract type, an abstract type that has a public constructor is incorrectly designed.

How to fix violations

To fix a violation of this rule, either make the constructor protected or don't declare the type as abstract.

When to suppress warnings

Do not suppress a warning from this rule. The abstract type has a public constructor.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1012
// The code that's violating the rule is on this line.
#pragma warning restore CA1012
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1012.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Design.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

Use the following option to configure which parts of your codebase to run this rule on.

- [Include specific API surfaces](#)

You can configure this option for just this rule, for all rules, or for all rules in this category ([Design](#)). For more information, see [Code quality rule configuration options](#).

Include specific API surfaces

You can configure which parts of your codebase to run this rule on, based on their accessibility. For example, to specify that the rule should run only against the non-public API surface, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.api_surface = private, internal
```

Example

The following code snippet contains an abstract type that violates this rule.

```
' Violates this rule
Public MustInherit Class Book

    Public Sub New()
    End Sub

End Class
```

```
// Violates this rule
public abstract class Book
{
    public Book()
    {
    }
}
```

The following code snippet fixes the previous violation by changing the accessibility of the constructor from `public` to `protected`.

```
// Does not violate this rule
public abstract class Book
{
    protected Book()
    {
    }
}
```

```
' Violates this rule
Public MustInherit Class Book

Protected Sub New()
End Sub

End Class
```

CA1014: Mark assemblies with CLSCompliantAttribute

9/20/2022 • 2 minutes to read • [Edit Online](#)

ITEM	VALUE
RuleId	CA1014
Category	Design
Fix is breaking or non-breaking	Non-breaking

Cause

An assembly does not have the [System.CLSCompliantAttribute](#) attribute applied to it.

Rule description

The Common Language Specification (CLS) defines naming restrictions, data types, and rules to which assemblies must conform if they will be used across programming languages. Good design dictates that all assemblies explicitly indicate CLS compliance with [CLSCompliantAttribute](#). If the attribute is not present on an assembly, the assembly is not compliant.

It is possible for a CLS-compliant assembly to contain types or type members that are not compliant.

How to fix violations

To fix a violation of this rule, add the attribute to the assembly. Instead of marking the whole assembly as noncompliant, you should determine which type or type members are not compliant and mark these elements as such. If possible, you should provide a CLS-compliant alternative for noncompliant members so that the widest possible audience can access all the functionality of your assembly.

When to suppress warnings

Do not suppress a warning from this rule. If you do not want the assembly to be compliant, apply the attribute and set its value to `false`.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1014
// The code that's violating the rule is on this line.
#pragma warning restore CA1014
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1014.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Design.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Example

The following example shows an assembly that has the [System.CLSCompliantAttribute](#) attribute applied that declares it CLS-compliant.

```
[assembly:CLSCompliant(true)]
namespace DesignLibrary {}
```

```
<assembly:CLSCompliant(true)>
Namespace DesignLibrary
End Namespace
```

See also

- [System.CLSCompliantAttribute](#)
- [Language Independence and Language-Independent Components](#)

CA1016: Mark assemblies with AssemblyVersionAttribute

9/20/2022 • 2 minutes to read • [Edit Online](#)

ITEM	VALUE
RuleId	CA1016
Category	Design
Fix is breaking or non-breaking	Non-breaking

Cause

The assembly does not have a version number.

Rule description

The identity of an assembly is composed of the following information:

- Assembly name
- Version number
- Culture
- Public key (for strongly named assemblies).

.NET uses the version number to uniquely identify an assembly and to bind to types in strongly named assemblies. The version number is used together with version and publisher policy. By default, applications run only with the assembly version with which they were built.

How to fix violations

To fix a violation of this rule, add a version number to the assembly by using the [System.Reflection.AssemblyVersionAttribute](#) attribute.

When to suppress warnings

Do not suppress a warning from this rule for assemblies that are used by third parties or in a production environment.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1016
// The code that's violating the rule is on this line.
#pragma warning restore CA1016
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1016.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Design.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Example

The following example shows an assembly that has the [AssemblyVersionAttribute](#) attribute applied.

```
using System;
using System.Reflection;

[assembly: AssemblyVersionAttribute("4.3.2.1")]
namespace DesignLibrary {}
```

```
<Assembly: AssemblyVersionAttribute("4.3.2.1")>
Namespace DesignLibrary
End Namespace
```

See also

- [Assembly versioning](#)
- [How to: Create a publisher policy](#)

CA1017: Mark assemblies with ComVisibleAttribute

9/20/2022 • 2 minutes to read • [Edit Online](#)

ITEM	VALUE
RuleId	CA1017
Category	Design
Fix is breaking or non-breaking	Non-breaking

Cause

An assembly does not have the [System.Runtime.InteropServices.ComVisibleAttribute](#) attribute applied to it.

Rule description

The [ComVisibleAttribute](#) attribute determines how COM clients access managed code. Good design dictates that assemblies explicitly indicate COM visibility. COM visibility can be set for a whole assembly and then overridden for individual types and type members. If the attribute is not present, the contents of the assembly are visible to COM clients.

How to fix violations

To fix a violation of this rule, add the attribute to the assembly. If you do not want the assembly to be visible to COM clients, apply the attribute and set its value to `false`.

When to suppress warnings

Do not suppress a warning from this rule. If you want the assembly to be visible, apply the attribute and set its value to `true`.

Example

The following example shows an assembly that has the [ComVisibleAttribute](#) attribute applied to prevent it from being visible to COM clients.

```
<Assembly: System.Runtime.InteropServices.ComVisible(False)>
Namespace DesignLibrary
End Namespace
```

```
[assembly: System.Runtime.InteropServices.ComVisible(false)]
namespace DesignLibrary {}
```

See also

- [Interoperating with Unmanaged Code](#)
- [Qualifying .NET Types for Interoperation](#)

CA1018: Mark attributes with AttributeUsageAttribute

9/20/2022 • 2 minutes to read • [Edit Online](#)

ITEM	VALUE
RuleId	CA1018
Category	Design
Fix is breaking or non-breaking	Breaking

Cause

The [System.AttributeUsageAttribute](#) attribute is not present on the custom attribute.

Rule description

When you define a custom attribute, mark it by using [AttributeUsageAttribute](#) to indicate where in the source code the custom attribute can be applied. The meaning and intended usage of an attribute will determine its valid locations in code. For example, you might define an attribute that identifies the person who is responsible for maintaining and enhancing each type in a library, and that responsibility is always assigned at the type level. In this case, compilers should enable the attribute on classes, enumerations, and interfaces, but should not enable it on methods, events, or properties. Organizational policies and procedures would dictate whether the attribute should be enabled on assemblies.

The [System.AttributeTargets](#) enumeration defines the targets that you can specify for a custom attribute. If you omit [AttributeUsageAttribute](#), your custom attribute will be valid for all targets, as defined by the `All` value of [AttributeTargets](#) enumeration.

How to fix violations

To fix a violation of this rule, specify targets for the attribute by using [AttributeUsageAttribute](#). See the following example.

When to suppress warnings

You should fix a violation of this rule instead of excluding the message. Even if the attribute inherits [AttributeUsageAttribute](#), the attribute should be present to simplify code maintenance.

Example

The following example defines two attributes. `BadCodeMaintainerAttribute` incorrectly omits the [AttributeUsageAttribute](#) statement, and `GoodCodeMaintainerAttribute` correctly implements the attribute that is described earlier in this section. (The property `DeveloperName` is required by the design rule [CA1019: Define accessors for attribute arguments](#) and is included for completeness.)

```

using System;

namespace ca1018
{
    // Violates rule: MarkAttributesWithAttributeUsage.
    public sealed class BadCodeMaintainerAttribute : Attribute
    {
        public BadCodeMaintainerAttribute(string developerName)
        {
            DeveloperName = developerName;
        }
        public string DeveloperName { get; }
    }

    // Satisfies rule: Attributes specify AttributeUsage.
    // This attribute is valid for type-level targets.
    [AttributeUsage(AttributeTargets.Class | AttributeTargets.Enum | AttributeTargets.Interface | AttributeTargets.Delegate)]
    public sealed class GoodCodeMaintainerAttribute : Attribute
    {
        public GoodCodeMaintainerAttribute(string developerName)
        {
            DeveloperName = developerName;
        }
        public string DeveloperName { get; }
    }
}

```

```

Imports System

Namespace ca1018

    ' Violates rule: MarkAttributesWithAttributeUsage.
    Public NotInheritable Class BadCodeMaintainerAttribute
        Inherits Attribute

        Public Sub New(developerName As String)
            Me.DeveloperName = developerName
        End Sub 'New

        Public ReadOnly Property DeveloperName() As String
    End Class

    ' Satisfies rule: Attributes specify AttributeUsage.
    ' The attribute is valid for type-level targets.
    <AttributeUsage(AttributeTargets.Class Or AttributeTargets.Enum Or AttributeTargets.Interface Or AttributeTargets.Delegate)>
    Public NotInheritable Class GoodCodeMaintainerAttribute
        Inherits Attribute

        Public Sub New(developerName As String)
            Me.DeveloperName = developerName
        End Sub 'New

        Public ReadOnly Property DeveloperName() As String
    End Class

End Namespace

```

Related rules

- [CA1019: Define accessors for attribute arguments](#)
- [CA1813: Avoid unsealed attributes](#)

See also

- [Attributes](#)

CA1019: Define accessors for attribute arguments

9/20/2022 • 3 minutes to read • [Edit Online](#)

ITEM	VALUE
RuleId	CA1019
Category	Design
Fix is breaking or non-breaking	Non-breaking

Cause

In its constructor, an attribute defines arguments that do not have corresponding properties.

Rule description

Attributes can define mandatory arguments that must be specified when you apply the attribute to a target. These are also known as positional arguments because they are supplied to attribute constructors as positional parameters. For every mandatory argument, the attribute should also provide a corresponding read-only property so that the value of the argument can be retrieved at execution time. This rule checks that for each constructor parameter, you have defined the corresponding property.

Attributes can also define optional arguments, which are also known as named arguments. These arguments are supplied to attribute constructors by name and should have a corresponding read/write property.

For mandatory and optional arguments, the corresponding properties and constructor parameters should use the same name but different casing. Properties use Pascal casing, and parameters use camel casing.

How to fix violations

To fix a violation of this rule, add a read-only property for each constructor parameter that does not have one.

When to suppress warnings

Suppress a warning from this rule if you do not want the value of the mandatory argument to be retrievable.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1019
// The code that's violating the rule is on this line.
#pragma warning restore CA1019
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1019.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Design.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Examples

Custom attributes

The following example shows two attributes that define a mandatory (positional) parameter. The first implementation of the attribute is incorrectly defined. The second implementation is correct.

```
// Violates rule: DefineAccessorsForAttributeArguments.
[AttributeUsage(AttributeTargets.All)]
public sealed class BadCustomAttribute : Attribute
{
    string _data;

    // Missing the property that corresponds to
    // the someStringData constructor parameter.

    public BadCustomAttribute(string someStringData)
    {
        _data = someStringData;
    }
}

// Satisfies rule: Attributes should have accessors for all arguments.
[AttributeUsage(AttributeTargets.All)]
public sealed class GoodCustomAttribute : Attribute
{
    public GoodCustomAttribute(string someStringData)
    {
        SomeStringData = someStringData;
    }

    //The constructor parameter and property
    //name are the same except for case.

    public string SomeStringData { get; }
}
```

```

Imports System

Namespace ca1019

    ' Violates rule: DefineAccessorsForAttributeArguments.
    <AttributeUsage(AttributeTargets.All)>
    Public NotInheritable Class BadCustomAttribute
        Inherits Attribute
        Private data As String

        ' Missing the property that corresponds to
        ' the someStringData parameter.
        Public Sub New(someStringData As String)
            data = someStringData
        End Sub 'New
    End Class 'BadCustomAttribute

    ' Satisfies rule: Attributes should have accessors for all arguments.
    <AttributeUsage(AttributeTargets.All)>
    Public NotInheritable Class GoodCustomAttribute
        Inherits Attribute

        Public Sub New(someStringData As String)
            Me.SomeStringData = someStringData
        End Sub 'New

        'The constructor parameter and property
        'name are the same except for case.

        Public ReadOnly Property SomeStringData() As String
    End Class

End Namespace

```

Positional and named arguments

Positional and named arguments make it clear to consumers of your library which arguments are mandatory for the attribute and which arguments are optional.

The following example shows an implementation of an attribute that has both positional and named arguments:

```

[AttributeUsage(AttributeTargets.All)]
public sealed class GoodCustomAttribute : Attribute
{
    public GoodCustomAttribute(string mandatoryData)
    {
        MandatoryData = mandatoryData;
    }

    public string MandatoryData { get; }

    public string OptionalData { get; set; }
}

```

The following example shows how to apply the custom attribute to two properties:

```

[GoodCustomAttribute("ThisIsSomeMandatoryData", OptionalData = "ThisIsSomeOptionalData")]
public string MyProperty { get; set; }

[GoodCustomAttribute("ThisIsSomeMoreMandatoryData")]
public string MyOtherProperty { get; set; }

```

Related rules

[CA1813: Avoid unsealed attributes](#)

See also

- [Attributes](#)

CA1021: Avoid out parameters

9/20/2022 • 7 minutes to read • [Edit Online](#)

ITEM	VALUE
RuleId	CA1021
Category	Design
Fix is breaking or non-breaking	Breaking

Cause

A public or protected method in a public type has an `out` parameter.

By default, this rule only looks at externally visible types, but this is [configurable](#).

Rule description

Passing types by reference (using `out` or `ref`) requires experience with pointers, understanding how value types and reference types differ, and handling methods with multiple return values. Also, the difference between `out` and `ref` parameters is not widely understood.

When a reference type is passed "by reference," the method intends to use the parameter to return a different instance of the object. Passing a reference type by reference is also known as using a double pointer, pointer to a pointer, or double indirection. By using the default calling convention, which is pass "by value," a parameter that takes a reference type already receives a pointer to the object. The pointer, not the object to which it points, is passed by value. Pass by value means that the method cannot change the pointer to have it point to a new instance of the reference type. However, it can change the contents of the object to which it points. For most applications this is sufficient and yields the desired behavior.

If a method must return a different instance, use the return value of the method to accomplish this. See the [System.String](#) class for a variety of methods that operate on strings and return a new instance of a string. When this model is used, the caller must decide whether the original object is preserved.

Although return values are commonplace and heavily used, the correct application of `out` and `ref` parameters requires intermediate design and coding skills. Library architects who design for a general audience should not expect users to become proficient in working with `out` or `ref` parameters.

How to fix violations

To fix a violation of this rule that is caused by a value type, have the method return the object as its return value. If the method must return multiple values, redesign it to return a single instance of an object that holds the values.

To fix a violation of this rule that is caused by a reference type, make sure that the desired behavior is to return a new instance of the reference. If it is, the method should use its return value to do this.

When to suppress warnings

It is safe to suppress a warning from this rule. However, this design could cause usability issues.

SUPPRESS A WARNING

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1021
// The code that's violating the rule is on this line.
#pragma warning restore CA1021
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1021.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Design.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

CONFIGURE CODE TO ANALYZE

Use the following option to configure which parts of your codebase to run this rule on.

- [Include specific API surfaces](#)

You can configure this option for just this rule, for all rules, or for all rules in this category ([Design](#)). For more information, see [Code quality rule configuration options](#).

Include specific API surfaces

You can configure which parts of your codebase to run this rule on, based on their accessibility. For example, to specify that the rule should run only against the non-public API surface, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXX.api_surface = private, internal
```

EXAMPLE 1

The following library shows two implementations of a class that generates responses to user feedback. The first implementation (`BadRefAndOut`) forces the library user to manage three return values. The second implementation (`RedesignedRefAndOut`) simplifies the user experience by returning an instance of a container class (`ReplyData`) that manages the data as a single unit.

```
public enum Actions
{
    Unknown,
    Discard,
    ForwardToManagement,
    ForwardToDeveloper
}

public enum TypeOfFeedback
{
```

```

    Complaint,
    Praise,
    Suggestion,
    Incomprehensible
}

public class BadRefAndOut
{
    // Violates rule: DoNotPassTypesByReference.

    public static bool ReplyInformation(TypeOfFeedback input,
        out string reply, ref Actions action)
    {
        bool returnReply = false;
        string replyText = "Your feedback has been forwarded " +
            "to the product manager.";

        reply = String.Empty;
        switch (input)
        {
            case TypeOfFeedback.Complaint:
            case TypeOfFeedback.Praise:
                action = Actions.ForwardToManagement;
                reply = "Thank you. " + replyText;
                returnReply = true;
                break;
            case TypeOfFeedback.Suggestion:
                action = Actions.ForwardToDeveloper;
                reply = replyText;
                returnReply = true;
                break;
            case TypeOfFeedback.Incomprehensible:
            default:
                action = Actions.Discard;
                returnReply = false;
                break;
        }
        return returnReply;
    }
}

// Redesigned version does not use out or ref parameters.
// Instead, it returns this container type.

public class ReplyData
{
    bool _returnReply;

    // Constructors.
    public ReplyData()
    {
        this.Reply = String.Empty;
        this.Action = Actions.Discard;
        this._returnReply = false;
    }

    public ReplyData(Actions action, string reply, bool returnReply)
    {
        this.Reply = reply;
        this.Action = action;
        this._returnReply = returnReply;
    }

    // Properties.
    public string Reply { get; }
    public Actions Action { get; }

    public override string ToString()

```

```

    {
        return String.Format("Reply: {0} Action: {1} return? {2}",
            Reply, Action.ToString(), _returnReply.ToString());
    }
}

public class RedesignedRefAndOut
{
    public static ReplyData ReplyInformation(TypeOfFeedback input)
    {
        ReplyData answer;
        string replyText = "Your feedback has been forwarded " +
            "to the product manager.";

        switch (input)
        {
            case TypeOfFeedback.Complaint:
            case TypeOfFeedback.Praise:
                answer = new ReplyData(
                    Actions.ForwardToManagement,
                    "Thank you. " + replyText,
                    true);
                break;
            case TypeOfFeedback.Suggestion:
                answer = new ReplyData(
                    Actions.ForwardToDeveloper,
                    replyText,
                    true);
                break;
            case TypeOfFeedback.Incomprehensible:
            default:
                answer = new ReplyData();
                break;
        }
        return answer;
    }
}

```

Example 2

The following application illustrates the experience of the user. The call to the redesigned library (`UseTheSimplifiedClass` method) is more straightforward, and the information returned by the method is easily managed. The output from the two methods is identical.

```

public class UseComplexMethod
{
    static void UseTheComplicatedClass()
    {
        // Using the version with the ref and out parameters.
        // You do not have to initialize an out parameter.

        string[] reply = new string[5];

        // You must initialize a ref parameter.
        Actions[] action = {Actions.Unknown,Actions.Unknown,
                            Actions.Unknown,Actions.Unknown,
                            Actions.Unknown,Actions.Unknown};

        bool[] disposition = new bool[5];
        int i = 0;

        foreach (TypeOfFeedback t in Enum.GetValues(typeof(TypeOfFeedback)))
        {
            // The call to the library.
            disposition[i] = BadRefAndOut.ReplyInformation(
                t, out reply[i], ref action[i]);
            Console.WriteLine("Reply: {0} Action: {1} return? {2} ",
                reply[i], action[i], disposition[i]);
            i++;
        }
    }

    static void UseTheSimplifiedClass()
    {
        ReplyData[] answer = new ReplyData[5];
        int i = 0;
        foreach (TypeOfFeedback t in Enum.GetValues(typeof(TypeOfFeedback)))
        {
            // The call to the library.
            answer[i] = RedesignedRefAndOut.ReplyInformation(t);
            Console.WriteLine(answer[i++]);
        }
    }

    public static void UseClasses()
    {
        UseTheComplicatedClass();

        // Print a blank line in output.
        Console.WriteLine("");

        UseTheSimplifiedClass();
    }
}

```

Example 3

The following example library illustrates how `ref` parameters for reference types are used and shows a better way to implement this functionality.

```

public class ReferenceTypesAndParameters
{
    // The following syntax will not work. You cannot make a
    // reference type that is passed by value point to a new
    // instance. This needs the ref keyword.

    public static void BadPassTheObject(string argument)
    {
        argument += " ABCDE";
    }

    // The following syntax works, but is considered bad design.
    // It reassigned the argument to point to a new instance of string.
    // Violates rule DoNotPassTypesByReference.

    public static void PassTheReference(ref string argument)
    {
        argument += " ABCDE";
    }

    // The following syntax works and is a better design.
    // It returns the altered argument as a new instance of string.

    public static string BetterThanPassTheReference(string argument)
    {
        return argument + " ABCDE";
    }
}

```

Example 4

The following application calls each method in the library to demonstrate the behavior.

```

public class Test
{
    public static void MainTest()
    {
        string s1 = "12345";
        string s2 = "12345";
        string s3 = "12345";

        Console.WriteLine("Changing pointer - passed by value:");
        Console.WriteLine(s1);
        ReferenceTypesAndParameters.BadPassTheObject(s1);
        Console.WriteLine(s1);

        Console.WriteLine("Changing pointer - passed by reference:");
        Console.WriteLine(s2);
        ReferenceTypesAndParameters.PassTheReference(ref s2);
        Console.WriteLine(s2);

        Console.WriteLine("Passing by return value:");
        s3 = ReferenceTypesAndParameters.BetterThanPassTheReference(s3);
        Console.WriteLine(s3);
    }
}

```

This example produces the following output:

```
Changing pointer - passed by value:  
12345  
12345  
Changing pointer - passed by reference:  
12345  
12345 ABCDE  
Passing by return value:  
12345 ABCDE
```

Try pattern methods

Methods that implement the `Try<Something>` pattern, such as [System.Int32.TryParse](#), do not raise this violation. The following example shows a structure (value type) that implements the [System.Int32.TryParse](#) method.

```

public struct Point
{
    public Point(int axisX, int axisY)
    {
        X = axisX;
        Y = axisY;
    }

    public int X { get; }

    public int Y { get; }

    public override int GetHashCode()
    {
        return X ^ Y;
    }

    public override bool Equals(object obj)
    {
        if (!(obj is Point))
            return false;

        return Equals((Point)obj);
    }

    public bool Equals(Point other)
    {
        if (X != other.X)
            return false;

        return Y == other.Y;
    }

    public static bool operator ==(Point point1, Point point2)
    {
        return point1.Equals(point2);
    }

    public static bool operator !=(Point point1, Point point2)
    {
        return !point1.Equals(point2);
    }

    // Does not violate this rule
    public static bool TryParse(string value, out Point result)
    {
        // TryParse Implementation
        result = new Point(0, 0);
        return false;
    }
}

```

Related rules

[CA1045: Do not pass types by reference](#)

CA1024: Use properties where appropriate

9/20/2022 • 6 minutes to read • [Edit Online](#)

ITEM	VALUE
RuleId	CA1024
Category	Design
Fix is breaking or non-breaking	Breaking

Cause

A method has a name that starts with `Get`, takes no parameters, and returns a value that's not an array.

By default, this rule only looks at externally visible methods, but this is [configurable](#).

Rule description

In most cases, properties represent data and methods perform actions. Properties are accessed like fields, which makes them easier to use. A method is a good candidate to become a property if one of these conditions is present:

- The method takes no arguments and returns the state information of an object.
- The method accepts a single argument to set some part of the state of an object.

How to fix violations

To fix a violation of this rule, change the method to a property.

When to suppress warnings

Suppress a warning from this rule if the method meets one of the following criteria. In these situations, a method is preferable to a property.

- The method can't behave as a field.
- The method performs a time-consuming operation. The method is perceptibly slower than the time that is required to set or get the value of a field.
- The method performs a conversion. Accessing a field does not return a converted version of the data that it stores.
- The `Get` method has an observable side effect. Retrieving the value of a field does not produce any side effects.
- The order of execution is important. Setting the value of a field does not rely on the occurrence of other operations.
- Calling the method two times in succession creates different results.
- The method is `static` but returns an object that can be changed by the caller. Retrieving the value of a field does not allow the caller to change the data that's stored by the field.
- The method returns an array.

SUPPRESS A WARNING

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1024
// The code that's violating the rule is on this line.
#pragma warning restore CA1024
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.cs,vb]
dotnet_diagnostic.CA1024.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.cs,vb]
dotnet_analyzer_diagnostic.category-Design.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

CONFIGURE CODE TO ANALYZE

Use the following option to configure which parts of your codebase to run this rule on.

- [Include specific API surfaces](#)

You can configure this option for just this rule, for all rules, or for all rules in this category ([Design](#)). For more information, see [Code quality rule configuration options](#).

Include specific API surfaces

You can configure which parts of your codebase to run this rule on, based on their accessibility. For example, to specify that the rule should run only against the non-public API surface, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXX.api_surface = private, internal
```

EXAMPLE

The following example contains several methods that should be converted to properties and several that should not because they don't behave like fields.

```
public class Appointment
{
    static long nextAppointmentID;
    static double[] discountScale = { 5.0, 10.0, 33.0 };
    string customerName;
    long customerID;
    DateTime when;

    // Static constructor.
    static Appointment()
    {
        // Initializes the static variable for Next appointment ID.
    }
}
```

```
// This method violates the rule, but should not be a property.  
// This method has an observable side effect.  
// Calling the method twice in succession creates different results.  
public static long GetNextAvailableID()  
{  
    nextAppointmentID++;  
    return nextAppointmentID - 1;  
}  
  
// This method violates the rule, but should not be a property.  
// This method performs a time-consuming operation.  
// This method returns an array.  
public Appointment[] GetCustomerHistory()  
{  
    // Connect to a database to get the customer's appointment history.  
    return LoadHistoryFromDB(customerID);  
}  
  
// This method violates the rule, but should not be a property.  
// This method is static but returns a mutable object.  
public static double[] GetDiscountScaleForUpdate()  
{  
    return discountScale;  
}  
  
// This method violates the rule, but should not be a property.  
// This method performs a conversion.  
public string GetWeekDayString()  
{  
    return DateTimeFormatInfo.CurrentInfo.GetDayName(when.DayOfWeek);  
}  
  
// These methods violate the rule and should be properties.  
// They each set or return a piece of the current object's state.  
  
public DayOfWeek GetWeekDay()  
{  
    return when.DayOfWeek;  
}  
  
public void SetCustomerName(string customerName)  
{  
    this.customerName = customerName;  
}  
  
public string GetCustomerName()  
{  
    return customerName;  
}  
  
public void SetCustomerID(long customerID)  
{  
    this.customerID = customerID;  
}  
  
public long GetCustomerID()  
{  
    return customerID;  
}  
  
public void SetScheduleTime(DateTime when)  
{  
    this.when = when;  
}  
  
public DateTime GetScheduleTime()  
{  
    return when;  
}
```

```

}

// Time-consuming method that is called by GetCustomerHistory.
Appointment[] LoadHistoryFromDB(long customerID)
{
    ArrayList records = new ArrayList();
    // Load from database.
    return (Appointment[])records.ToArray();
}
}

```

```

Public Class Appointment
    Shared nextAppointmentID As Long
    Shared discountScale As Double() = {5.0, 10.0, 33.0}
    Private customerName As String
    Private customerID As Long
    Private [when] As Date

    ' Static constructor.
    Shared Sub New()
        ' Initializes the static variable for Next appointment ID.
    End Sub

    ' This method violates the rule, but should not be a property.
    ' This method has an observable side effect.
    ' Calling the method twice in succession creates different results.
    Public Shared Function GetNextAvailableID() As Long
        nextAppointmentID += 1
        Return nextAppointmentID - 1
    End Function

    ' This method violates the rule, but should not be a property.
    ' This method performs a time-consuming operation.
    ' This method returns an array.
    Public Function GetCustomerHistory() As Appointment()
        ' Connect to a database to get the customer's appointment history.
        Return LoadHistoryFromDB(customerID)
    End Function

    ' This method violates the rule, but should not be a property.
    ' This method is static but returns a mutable object.
    Public Shared Function GetDiscountScaleForUpdate() As Double()
        Return discountScale
    End Function

    ' This method violates the rule, but should not be a property.
    ' This method performs a conversion.
    Public Function GetWeekDayString() As String
        Return DateTimeFormatInfo.CurrentInfo.GetDayName([when].DayOfWeek)
    End Function

    ' These methods violate the rule and should be properties.
    ' They each set or return a piece of the current object's state.

    Public Function GetWeekDay() As DayOfWeek
        Return [when].DayOfWeek
    End Function

    Public Sub SetCustomerName(customerName As String)
        Me.customerName = customerName
    End Sub

    Public Function GetCustomerName() As String
        Return customerName
    End Function

    Public Sub SetCustomerID(customerID As Long)
        Me.customerID = customerID
    End Sub

```

```

Me.CustomerID = customerID
End Sub

Public Function GetCustomerID() As Long
    Return customerID
End Function

Public Sub SetScheduleTime([when] As Date)
    Me.[when] = [when]
End Sub

Public Function GetScheduleTime() As Date
    Return [when]
End Function

' Time-consuming method that is called by GetCustomerHistory.
Private Function LoadHistoryFromDB(customerID As Long) As Appointment()
    Dim records As ArrayList = New ArrayList()
    Return CType(records.ToArray(), Appointment())
End Function
End Class

```

Control property expansion in the debugger

One reason programmers avoid using a property is because they do not want the debugger to autoexpand it. For example, the property might involve allocating a large object or calling a P/Invoke, but it might not actually have any observable side effects.

You can prevent the debugger from auto-expanding properties by applying [System.Diagnostics.DebuggerBrowsableAttribute](#). The following example shows this attribute being applied to an instance property.

```

Imports System.Diagnostics

Namespace Microsoft.Samples
    Public Class TestClass
        ' [...]

        <DebuggerBrowsable(DebuggerBrowsableState.Never)> _
        Public ReadOnly Property LargeObject() As LargeObject
            Get
                ' Allocate large object
                ' [...]
            End Get
        End Property
    End Class
End Namespace

```

```
using System.Diagnostics;

namespace Microsoft.Samples
{
    class TestClass
    {
        // [...]

        [DebuggerBrowsable(DebuggerBrowsableState.Never)]
        public LargeObject LargeObject
        {
            get
            {
                // Allocate large object
                // [...]
            }
        }
    }
}
```

CA1027: Mark enums with FlagsAttribute

9/20/2022 • 3 minutes to read • [Edit Online](#)

ITEM	VALUE
RuleId	CA1027
Category	Design
Fix is breaking or non-breaking	Non-breaking

Cause

The values of an enumeration are powers of two or are combinations of other values that are defined in the enumeration, and the [System.FlagsAttribute](#) attribute is not present. To reduce false positives, this rule does not report a violation for enumerations that have contiguous values.

By default, this rule only looks at externally visible enumerations, but this is [configurable](#).

Rule description

An enumeration is a value type that defines a set of related named constants. Apply [FlagsAttribute](#) to an enumeration when its named constants can be meaningfully combined. For example, consider an enumeration of the days of the week in an application that keeps track of which day's resources are available. If the availability of each resource is encoded by using the enumeration that has [FlagsAttribute](#) present, any combination of days can be represented. Without the attribute, only one day of the week can be represented.

For fields that store combinable enumerations, the individual enumeration values are treated as groups of bits in the field. Therefore, such fields are sometimes referred to as *bit fields*. To combine enumeration values for storage in a bit field, use the Boolean conditional operators. To test a bit field to determine whether a specific enumeration value is present, use the Boolean logical operators. For a bit field to store and retrieve combined enumeration values correctly, each value that is defined in the enumeration must be a power of two. Unless this is so, the Boolean logical operators will not be able to extract the individual enumeration values that are stored in the field.

How to fix violations

To fix a violation of this rule, add [FlagsAttribute](#) to the enumeration.

When to suppress warnings

Suppress a warning from this rule if you do not want the enumeration values to be combinable.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1027
// The code that's violating the rule is on this line.
#pragma warning restore CA1027
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1027.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Design.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

Use the following option to configure which parts of your codebase to run this rule on.

- [Include specific API surfaces](#)

You can configure this option for just this rule, for all rules, or for all rules in this category ([Design](#)). For more information, see [Code quality rule configuration options](#).

Include specific API surfaces

You can configure which parts of your codebase to run this rule on, based on their accessibility. For example, to specify that the rule should run only against the non-public API surface, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.api_surface = private, internal
```

Example

In the following example, `DaysEnumNeedsFlags` is an enumeration that meets the requirements for using [FlagsAttribute](#) but doesn't have it. The `ColorEnumShouldNotHaveFlag` enumeration does not have values that are powers of two but incorrectly specifies [FlagsAttribute](#). This violates rule [CA2217: Do not mark enums with FlagsAttribute](#).

```
// Violates rule: MarkEnumsWithFlags.
public enum DaysEnumNeedsFlags
{
    None = 0,
    Monday = 1,
    Tuesday = 2,
    Wednesday = 4,
    Thursday = 8,
    Friday = 16,
    All = Monday | Tuesday | Wednesday | Thursday | Friday
}

// Violates rule: DoNotMarkEnumsWithFlags.
[FlagsAttribute]
public enum ColorEnumShouldNotHaveFlag
{
    None = 0,
    Red = 1,
    Orange = 3,
    Yellow = 4
}
```

Related rules

- [CA2217: Do not mark enums with FlagsAttribute](#)

See also

- [System.FlagsAttribute](#)

CA1028: Enum storage should be Int32

9/20/2022 • 3 minutes to read • [Edit Online](#)

ITEM	VALUE
RuleId	CA1028
Category	Design
Fix is breaking or non-breaking	Breaking

Cause

The underlying type of an enumeration is not [System.Int32](#).

By default, this rule only looks at externally visible enumerations, but this is [configurable](#).

Rule description

An enumeration is a value type that defines a set of related named constants. By default, the [System.Int32](#) data type is used to store the constant value. Even though you can change this underlying type, it is not necessary or recommended for most scenarios. No significant performance gain is achieved by using a data type that is smaller than [Int32](#). If you cannot use the default data type, you should use one of the Common Language System (CLS)-compliant integral types, [Byte](#), [Int16](#), [Int32](#), or [Int64](#) to make sure that all values of the enumeration can be represented in CLS-compliant programming languages.

How to fix violations

To fix a violation of this rule, unless size or compatibility issues exist, use [Int32](#). For situations where [Int32](#) is not large enough to hold the values, use [Int64](#). If backward compatibility requires a smaller data type, use [Byte](#) or [Int16](#).

When to suppress warnings

Suppress a warning from this rule only if backward compatibility issues require it. In applications, failure to comply with this rule usually does not cause problems. In libraries, where language interoperability is required, failure to comply with this rule might adversely affect your users.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1028
// The code that's violating the rule is on this line.
#pragma warning restore CA1028
```

To disable the rule for a file, folder, or project, set its severity to [none](#) in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1028.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Design.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

Use the following option to configure which parts of your codebase to run this rule on.

- [Include specific API surfaces](#)

You can configure this option for just this rule, for all rules, or for all rules in this category ([Design](#)). For more information, see [Code quality rule configuration options](#).

Include specific API surfaces

You can configure which parts of your codebase to run this rule on, based on their accessibility. For example, to specify that the rule should run only against the non-public API surface, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXX.api_surface = private, internal
```

Example

The following example shows two enumerations that don't use the recommended underlying data type.

```
[Flags]
public enum Days : uint
{
    None = 0,
    Monday = 1,
    Tuesday = 2,
    Wednesday = 4,
    Thursday = 8,
    Friday = 16,
    All = Monday | Tuesday | Wednesday | Thursday | Friday
}

public enum Color : sbyte
{
    None = 0,
    Red = 1,
    Orange = 3,
    Yellow = 4
}
```

```

<Flags()
Public Enum Days As UInteger
    None = 0
    Monday = 1
    Tuesday = 2
    Wednesday = 4
    Thursday = 8
    Friday = 16
    All = Monday Or Tuesday Or Wednesday Or Thursday Or Friday
End Enum

Public Enum Color As SByte
    None = 0
    Red = 1
    Orange = 3
    Yellow = 4
End Enum

```

The following example fixes the previous violation by changing the underlying data type to [Int32](#).

```

[Flags]
public enum Days : int
{
    None = 0,
    Monday = 1,
    Tuesday = 2,
    Wednesday = 4,
    Thursday = 8,
    Friday = 16,
    All = Monday | Tuesday | Wednesday | Thursday | Friday
}

public enum Color : int
{
    None = 0,
    Red = 1,
    Orange = 3,
    Yellow = 4
}

```

```

<Flags()
Public Enum Days As Integer
    None = 0
    Monday = 1
    Tuesday = 2
    Wednesday = 4
    Thursday = 8
    Friday = 16
    All = Monday Or Tuesday Or Wednesday Or Thursday Or Friday
End Enum

Public Enum Color As Integer
    None = 0
    Red = 1
    Orange = 3
    Yellow = 4
End Enum

```

Related rules

- [CA1008: Enums should have zero value](#)

- [CA1027: Mark enums with FlagsAttribute](#)
- [CA2217: Do not mark enums with FlagsAttribute](#)
- [CA1700: Do not name enum values 'Reserved'](#)
- [CA1712: Do not prefix enum values with type name](#)

See also

- [System.Byte](#)
- [System.Int16](#)
- [System.Int32](#)
- [System.Int64](#)

CA1030: Use events where appropriate

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA1030
Category	Design
Fix is breaking or non-breaking	Non-breaking

Cause

A method name begins with one of the following:

- AddOn
- RemoveOn
- Fire
- Raise

By default, this rule only looks at externally visible methods, but this is [configurable](#).

Rule description

This rule detects methods that have names that ordinarily would be used for events. Events follow the Observer or Publish-Subscribe design pattern; they are used when a state change in one object must be communicated to other objects. If a method gets called in response to a clearly defined state change, the method should be invoked by an event handler. Objects that call the method should raise events instead of calling the method directly.

Some common examples of events are found in user interface applications where a user action such as clicking a button causes a segment of code to execute. The .NET event model is not limited to user interfaces. It should be used anywhere you must communicate state changes to one or more objects.

How to fix violations

If the method is called when the state of an object changes, consider changing the design to use the .NET event model.

When to suppress warnings

Suppress a warning from this rule if the method does not work with the .NET event model.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1030
// The code that's violating the rule is on this line.
#pragma warning restore CA1030
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1030.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Design.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

Use the following option to configure which parts of your codebase to run this rule on.

- [Include specific API surfaces](#)

You can configure this option for just this rule, for all rules, or for all rules in this category ([Design](#)). For more information, see [Code quality rule configuration options](#).

Include specific API surfaces

You can configure which parts of your codebase to run this rule on, based on their accessibility. For example, to specify that the rule should run only against the non-public API surface, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.api_surface = private, internal
```

CA1031: Do not catch general exception types

9/20/2022 • 3 minutes to read • [Edit Online](#)

ITEM	VALUE
RuleId	CA1031
Category	Design
Fix is breaking or non-breaking	Non-breaking

Cause

A general exception such as `System.Exception` or `System.SystemException` is caught in a `catch` statement, or a general catch clause such as `catch()` is used.

By default, this rule only flags general exception types being caught, but this is [configurable](#).

Rule description

General exceptions should not be caught.

How to fix violations

To fix a violation of this rule, catch a more specific exception, or rethrow the general exception as the last statement in the `catch` block.

When to suppress warnings

Do not suppress a warning from this rule. Catching general exception types can hide run-time problems from the library user and can make debugging more difficult.

NOTE

Starting with .NET Framework 4, the common language runtime (CLR) no longer delivers corrupted state exceptions that occur in the operating system and managed code, such as access violations in Windows, to be handled by managed code. If you want to compile an application in .NET Framework 4 or later versions and maintain handling of corrupted state exceptions, you can apply the `HandleProcessCorruptedStateExceptionsAttribute` attribute to the method that handles the corrupted state exception.

Configure code to analyze

Use the following option to configure which parts of your codebase to run this rule on.

- [Disallowed exception type names](#)

You can configure this option for just this rule, for all rules, or for all rules in this category ([Design](#)). For more information, see [Code quality rule configuration options](#).

Disallowed exception type names

You can configure which exception types are disallowed from being caught. For example, to specify that the rule should flag `catch` handlers with `NullReferenceException`, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CA1031.disallowed_symbol_names = NullReferenceException
```

Allowed type name formats in the option value (separated by `|`):

- Type name only (includes all symbols with the name, regardless of the containing type or namespace)
- Fully qualified names in the symbol's [documentation ID format](#) with a `T:` prefix.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CA1031.disallowed_symbol_names = ExceptionType</code>	Matches all symbols named 'ExceptionType' in the compilation
<code>dotnet_code_quality.CA1031.disallowed_symbol_names = ExceptionType1 ExceptionType2</code>	Matches all symbols named either 'ExceptionType1' or 'ExceptionType2' in the compilation
<code>dotnet_code_quality.CA1031.disallowed_symbol_names = T:NS.ExceptionType</code>	Matches specific types named 'ExceptionType' with given fully qualified name.
<code>dotnet_code_quality.CA1031.disallowed_symbol_names = T:NS1.ExceptionType1 T:NS1.ExceptionType2</code>	Matches types named 'ExceptionType1' and 'ExceptionType2' with respective fully qualified names

You can configure these options for just this rule, for all rules, or for all rules in this category ([Design](#)). For more information, see [Code quality rule configuration options](#).

Example

The following example shows a type that violates this rule and a type that correctly implements the `catch` block.

```

Imports System
Imports System.IO

Namespace ca1031

    ' Creates two violations of the rule.
    Public Class GenericExceptionsCaught

        Dim inStream As FileStream
        Dim outStream As FileStream

        Sub New(inFile As String, outFile As String)

            Try
                inStream = File.Open(inFile, FileMode.Open)
            Catch ex As SystemException
                Console.WriteLine("Unable to open {0}.", inFile)
            End Try

            Try
                outStream = File.Open(outFile, FileMode.Open)
            Catch
                Console.WriteLine("Unable to open {0}.", outFile)
            End Try

        End Sub

    End Class

    Public Class GenericExceptionsCaughtFixed

        Dim inStream As FileStream
        Dim outStream As FileStream

        Sub New(inFile As String, outFile As String)

            Try
                inStream = File.Open(inFile, FileMode.Open)

                ' Fix the first violation by catching a specific exception.
            Catch ex As FileNotFoundException
                Console.WriteLine("Unable to open {0}.", inFile)
                ' For functionally equivalent code, also catch the
                ' remaining exceptions that may be thrown by File.Open
            End Try

            Try
                outStream = File.Open(outFile, FileMode.Open)

                ' Fix the second violation by re-throwing the generic
                ' exception at the end of the catch block.
            Catch
                Console.WriteLine("Unable to open {0}.", outFile)
                Throw
            End Try

        End Sub

    End Class

End Namespace

```

```

// Creates two violations of the rule.
public class GenericExceptionsCaught
{
    FileStream inStream;
    FileStream outStream;

    public GenericExceptionsCaught(string inFile, string outFile)
    {
        try
        {
            inStream = File.Open(inFile, FileMode.Open);
        }
        catch (SystemException)
        {
            Console.WriteLine("Unable to open {0}.". , inFile);
        }

        try
        {
            outStream = File.Open(outFile, FileMode.Open);
        }
        catch
        {
            Console.WriteLine("Unable to open {0}.". , outFile);
        }
    }
}

public class GenericExceptionsCaughtFixed
{
    FileStream inStream;
    FileStream outStream;

    public GenericExceptionsCaughtFixed(string inFile, string outFile)
    {
        try
        {
            inStream = File.Open(inFile, FileMode.Open);
        }

        // Fix the first violation by catching a specific exception.
        catch (FileNotFoundException)
        {
            Console.WriteLine("Unable to open {0}.". , inFile);
        };

        // For functionally equivalent code, also catch
        // remaining exceptions that may be thrown by File.Open

        try
        {
            outStream = File.Open(outFile, FileMode.Open);
        }

        // Fix the second violation by rethrowing the generic
        // exception at the end of the catch block.
        catch
        {
            Console.WriteLine("Unable to open {0}.". , outFile);
            throw;
        }
    }
}

```

Related rules

CA2200: Rethrow to preserve stack details

CA1032: Implement standard exception constructors

9/20/2022 • 2 minutes to read • [Edit Online](#)

ITEM	VALUE
RuleId	CA1032
Category	Design
Fix is breaking or non-breaking	Non-breaking

Cause

A type extends `System.Exception` but doesn't declare all the required constructors.

Rule description

Exception types must implement the following three public constructors:

- `public NewException()`
- `public NewException(string)`
- `public NewException(string, Exception)`

Failure to provide the full set of constructors can make it difficult to correctly handle exceptions. For example, the constructor that has the signature `NewException(string, Exception)` is used to create exceptions that are caused by other exceptions. Without this constructor, you can't create and throw an instance of your custom exception that contains an inner (nested) exception, which is what managed code should do in such a situation.

For more information, see [CA2229: Implement serialization constructors](#).

How to fix violations

To fix a violation of this rule, add the missing constructors to the exception, and make sure that they have the correct accessibility.

When to suppress warnings

It's safe to suppress a warning from this rule when the violation is caused by using a different access level for the public constructors.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1032
// The code that's violating the rule is on this line.
#pragma warning restore CA1032
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1032.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Design.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Example

The following example contains an exception type that violates this rule and an exception type that is correctly implemented.

```
// Violates rule ImplementStandardExceptionConstructors.
public class BadException : Exception
{
    public BadException()
    {
        // Add any type-specific logic, and supply the default message.
    }
}

[Serializable()]
public class GoodException : Exception
{
    public GoodException()
    {
        // Add any type-specific logic, and supply the default message.
    }

    public GoodException(string message) : base(message)
    {
        // Add any type-specific logic.
    }

    public GoodException(string message, Exception innerException) :
        base(message, innerException)
    {
        // Add any type-specific logic for inner exceptions.
    }
}
```

See also

[CA2229: Implement serialization constructors](#)

CA1033: Interface methods should be callable by child types

9/20/2022 • 2 minutes to read • [Edit Online](#)

ITEM	VALUE
RuleId	CA1033
Category	Design
Fix is breaking or non-breaking	Non-breaking

Cause

An unsealed externally visible type provides an explicit method implementation of a public interface and does not provide an alternative externally visible method that has the same name.

Rule description

Consider a base type that explicitly implements a public interface method. A type that derives from the base type can access the inherited interface method only through a reference to the current instance (`this` in C#) that is cast to the interface. If the derived type reimplements (explicitly) the inherited interface method, the base implementation can no longer be accessed. The call through the current instance reference will invoke the derived implementation; this causes recursion and an eventual stack overflow.

This rule does not report a violation for an explicit implementation of `System.IDisposable.Dispose` when an externally visible `Close()` or `System.IDisposable.Dispose(Boolean)` method is provided.

How to fix violations

To fix a violation of this rule, implement a new method that exposes the same functionality and is visible to derived types or change to a nonexplicit implementation. If a breaking change is acceptable, an alternative is to make the type sealed.

When to suppress warnings

It is safe to suppress a warning from this rule if an externally visible method is provided that has the same functionality but a different name than the explicitly implemented method.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1033
// The code that's violating the rule is on this line.
#pragma warning restore CA1033
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1033.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Design.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Example

The following example shows a type, `ViolatingBase`, that violates the rule and a type, `FixedBase`, that shows a fix for the violation.

```
public interface ITest
{
    void SomeMethod();
}

public class ViolatingBase : ITest
{
    void ITest.SomeMethod()
    {
        // ...
    }
}

public class FixedBase : ITest
{
    void ITest.SomeMethod()
    {
        SomeMethod();
    }

    protected void SomeMethod()
    {
        // ...
    }
}

sealed public class Derived : FixedBase, ITest
{
    public void SomeMethod()
    {
        // The following would cause recursion and a stack overflow.
        // ((ITest)this).SomeMethod();

        // The following is unavailable if derived from ViolatingBase.
        base.SomeMethod();
    }
}
```

See also

- [Interfaces](#)

CA1034: Nested types should not be visible

9/20/2022 • 2 minutes to read • [Edit Online](#)

ITEM	VALUE
RuleId	CA1034
Category	Design
Fix is breaking or non-breaking	Breaking

Cause

An externally visible type contains an externally visible type declaration. Nested enumerations, protected types, and builder patterns are exempt from this rule.

Rule description

A nested type is a type declared within the scope of another type. Nested types are useful for encapsulating private implementation details of the containing type. Used for this purpose, nested types should not be externally visible.

Do not use externally visible nested types for logical grouping or to avoid name collisions; instead, use namespaces.

Nested types include the notion of member accessibility, which some programmers do not understand clearly.

Protected types can be used in subclasses and nested types in advance customization scenarios.

How to fix violations

If you do not intend the nested type to be externally visible, change the type's accessibility. Otherwise, remove the nested type from its parent. If the purpose of the nesting is to categorize the nested type, use a namespace to create the hierarchy instead.

When to suppress warnings

Do not suppress a warning from this rule.

Example

The following example shows a type that violates the rule.

```
public class ParentType
{
    public class NestedType
    {
        public NestedType()
        {
        }
    }

    public ParentType()
    {
        NestedType nt = new NestedType();
    }
}
```

```
Imports System

Namespace ca1034

Class ParentType

    Public Class NestedType
        Sub New()
        End Sub
    End Class

    Sub New()
    End Sub

End Class

End Namespace
```

CA1036: Override methods on comparable types

9/20/2022 • 4 minutes to read • [Edit Online](#)

ITEM	VALUE
RuleId	CA1036
Category	Design
Fix is breaking or non-breaking	Non-breaking

Cause

A type implements the [System.IComparable](#) interface and does not override [System.Object.Equals](#) or does not overload the language-specific operator for equality, inequality, less-than, or greater-than. The rule does not report a violation if the type inherits only an implementation of the interface.

By default, this rule only looks at externally visible types, but this is [configurable](#).

Rule description

Types that define a custom sort order implement the [IComparable](#) interface. The [CompareTo](#) method returns an integer value that indicates the correct sort order for two instances of the type. This rule identifies types that set a sort order. Setting a sort order implies that the ordinary meaning of equality, inequality, less-than, and greater-than don't apply. When you provide an implementation of [IComparable](#), you must usually also override [Equals](#) so that it returns values that are consistent with [CompareTo](#). If you override [Equals](#) and are coding in a language that supports operator overloads, you should also provide operators that are consistent with [Equals](#).

How to fix violations

To fix a violation of this rule, override [Equals](#). If your programming language supports operator overloading, supply the following operators:

- `op_Equality`
- `op_Inequality`
- `op_LessThan`
- `op_GreaterThan`

In C#, the tokens that are used to represent these operators are as follows:

```
==  
!=  
<  
>
```

When to suppress warnings

It is safe to suppress a warning from rule CA1036 when the violation is caused by missing operators and your programming language does not support operator overloading, as is the case with Visual Basic. If you

determine that implementing the operators does not make sense in your app context, it's also safe to suppress a warning from this rule when it fires on equality operators other than `op_Equality`. However, you should always override `op_Equality` and the `==` operator if you override [Object.Equals](#).

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1036
// The code that's violating the rule is on this line.
#pragma warning restore CA1036
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1036.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Design.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

Use the following option to configure which parts of your codebase to run this rule on.

- [Include specific API surfaces](#)

You can configure this option for just this rule, for all rules, or for all rules in this category ([Design](#)). For more information, see [Code quality rule configuration options](#).

Include specific API surfaces

You can configure which parts of your codebase to run this rule on, based on their accessibility. For example, to specify that the rule should run only against the non-public API surface, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.api_surface = private, internal
```

Examples

The following code contains a type that correctly implements [IComparable](#). Code comments identify the methods that satisfy various rules that are related to [Equals](#) and the [IComparable](#) interface.

```
// Valid ratings are between A and C.
// A is the highest rating; it is greater than any other valid rating.
// C is the lowest rating; it is less than any other valid rating.

public class RatingInformation : IComparable, IComparable<RatingInformation>
{
    public string Rating
    {
        ...
    }
}
```

```

        get;
        private set;
    }

    public RatingInformation(string rating)
    {
        if (rating == null)
        {
            throw new ArgumentNullException("rating");
        }

        string v = rating.ToUpper(CultureInfo.InvariantCulture);
        if (v.Length != 1 || string.Compare(v, "C", StringComparison.OrdinalIgnoreCase) > 0 || string.Compare(v, "A", StringComparison.OrdinalIgnoreCase) < 0)
        {
            throw new ArgumentException("Invalid rating value was specified.", "rating");
        }

        Rating = v;
    }

    public int CompareTo(object obj)
    {
        if (obj == null)
        {
            return 1;
        }

        RatingInformation other = obj as RatingInformation; // avoid double casting
        if (other == null)
        {
            throw new ArgumentException("A RatingInformation object is required for comparison.", "obj");
        }

        return CompareTo(other);
    }

    public int CompareTo(RatingInformation other)
    {
        if (other is null)
        {
            return 1;
        }

        // Ratings compare opposite to normal string order,
        // so reverse the value returned by String.CompareTo.
        return -string.Compare(this.Rating, other.Rating, StringComparison.OrdinalIgnoreCase);
    }

    public static int Compare(RatingInformation left, RatingInformation right)
    {
        if (object.ReferenceEquals(left, right))
        {
            return 0;
        }
        if (left is null)
        {
            return -1;
        }
        return left.CompareTo(right);
    }

    // Omitting Equals violates rule: OverrideMethodsOnComparableTypes.
    public override bool Equals(object obj)
    {
        RatingInformation other = obj as RatingInformation; //avoid double casting
        if (other is null)
        {
            return false;
        }
    }
}

```

```

        }
        return this.CompareTo(other) == 0;
    }

// Omitting GetHashCode violates rule: OverrideGetHashCodeOnOverridingEquals.
public override int GetHashCode()
{
    char[] c = this.Rating.ToCharArray();
    return (int)c[0];
}

// Omitting any of the following operator overloads
// violates rule: OverrideMethodsOnComparableTypes.
public static bool operator ==(RatingInformation left, RatingInformation right)
{
    if (left is null)
    {
        return right is null;
    }
    return left.Equals(right);
}
public static bool operator !=(RatingInformation left, RatingInformation right)
{
    return !(left == right);
}
public static bool operator <(RatingInformation left, RatingInformation right)
{
    return (Compare(left, right) < 0);
}
public static bool operator >(RatingInformation left, RatingInformation right)
{
    return (Compare(left, right) > 0);
}
}

```

The following application code tests the behavior of the [IComparable](#) implementation that was shown earlier.

```

public class Test
{
    public static void Main1036(string[] args)
    {
        if (args.Length < 2)
        {
            Console.WriteLine("usage - TestRatings string1 string2");
            return;
        }
        RatingInformation r1 = new RatingInformation(args[0]);
        RatingInformation r2 = new RatingInformation(args[1]);
        string answer;

        if (r1.CompareTo(r2) > 0)
            answer = "greater than";
        else if (r1.CompareTo(r2) < 0)
            answer = "less than";
        else
            answer = "equal to";

        Console.WriteLine("{0} is {1} {2}", r1.Rating, answer, r2.Rating);
    }
}

```

See also

- [System.IComparable](#)

- `System.Object.Equals`
- Equality operators

CA1040: Avoid empty interfaces

9/20/2022 • 2 minutes to read • [Edit Online](#)

ITEM	VALUE
RuleId	CA1040
Category	Design
Fix is breaking or non-breaking	Breaking

Cause

The interface does not declare any members or implement two or more other interfaces.

By default, this rule only looks at externally visible interfaces, but this is [configurable](#).

Rule description

Interfaces define members that provide a behavior or usage contract. The functionality that is described by the interface can be adopted by any type, regardless of where the type appears in the inheritance hierarchy. A type implements an interface by providing implementations for the members of the interface. An empty interface does not define any members. Therefore, it does not define a contract that can be implemented.

If your design includes empty interfaces that types are expected to implement, you are probably using an interface as a marker or a way to identify a group of types. If this identification will occur at run time, the correct way to accomplish this is to use a custom attribute. Use the presence or absence of the attribute, or the properties of the attribute, to identify the target types. If the identification must occur at compile time, then it is acceptable to use an empty interface.

How to fix violations

Remove the interface or add members to it. If the empty interface is being used to label a set of types, replace the interface with a custom attribute.

When to suppress warnings

It is safe to suppress a warning from this rule when the interface is used to identify a set of types at compile time.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1040
// The code that's violating the rule is on this line.
#pragma warning restore CA1040
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1040.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Design.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

Use the following option to configure which parts of your codebase to run this rule on.

- [Include specific API surfaces](#)

You can configure this option for just this rule, for all rules, or for all rules in this category ([Design](#)). For more information, see [Code quality rule configuration options](#).

Include specific API surfaces

You can configure which parts of your codebase to run this rule on, based on their accessibility. For example, to specify that the rule should run only against the non-public API surface, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXX.api_surface = private, internal
```

Example

The following example shows an empty interface.

```
// Violates rule
public interface IBadInterface
{}
```

```
' Violates rule
Public Interface IBadInterface
End Interface
```

CA1041: Provide ObsoleteAttribute message

9/20/2022 • 2 minutes to read • [Edit Online](#)

ITEM	VALUE
RuleId	CA1041
Category	Design
Fix is breaking or non-breaking	Non-breaking

Cause

A type or member is marked by using a [System.ObsoleteAttribute](#) attribute that does not have its [System.ObsoleteAttribute.Message](#) property specified.

By default, this rule only looks at externally visible types and members, but this is [configurable](#).

Rule description

[ObsoleteAttribute](#) is used to mark deprecated library types and members. Library consumers should avoid the use of any type or member that is marked obsolete. This is because it might not be supported and will eventually be removed from later versions of the library. When a type or member marked by using [ObsoleteAttribute](#) is compiled, the [Message](#) property of the attribute is displayed. This gives the user information about the obsolete type or member. This information generally includes how long the obsolete type or member will be supported by the library designers and the preferred replacement to use.

How to fix violations

To fix a violation of this rule, add the `message` parameter to the [ObsoleteAttribute](#) constructor.

When to suppress warnings

Do not suppress a warning from this rule because the [Message](#) property provides critical information about the obsolete type or member.

Configure code to analyze

Use the following option to configure which parts of your codebase to run this rule on.

- [Include specific API surfaces](#)

You can configure this option for just this rule, for all rules, or for all rules in this category ([Design](#)). For more information, see [Code quality rule configuration options](#).

Include specific API surfaces

You can configure which parts of your codebase to run this rule on, based on their accessibility. For example, to specify that the rule should run only against the non-public API surface, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXX.api_surface = private, internal
```

Example

The following example shows an obsolete member that has a correctly declared [ObsoleteAttribute](#).

```
[ObsoleteAttribute("This property is obsolete and will be removed in a " +  
    "future version. Use the FullName property instead.", false)]  
public string Name  
{  
    get => "Name";  
}
```

```
Imports System  
  
Namespace ca1041  
  
    Public Class ObsoleteAttributeOnMember  
  
        <ObsoleteAttribute("This property is obsolete and will " &  
            "be removed in a future version. Use the FirstName " &  
            "and LastName properties instead.", False)>  
        ReadOnly Property Name As String  
            Get  
                Return "Name"  
            End Get  
        End Property  
  
        ReadOnly Property FirstName As String  
            Get  
                Return "FirstName"  
            End Get  
        End Property  
  
        ReadOnly Property LastName As String  
            Get  
                Return "LastName"  
            End Get  
        End Property  
  
    End Class  
  
End Namespace
```

See also

- [System.ObsoleteAttribute](#)

CA1043: Use integral or string argument for indexers

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA1043
Category	Design
Fix is breaking or non-breaking	Breaking

Cause

A type contains an indexer that uses an index type other than [System.Int32](#), [System.Int64](#), [System.Object](#), or [System.String](#).

By default, this rule only looks at externally visible types, but this is [configurable](#).

Rule description

Indexers, that is, indexed properties, should use integer or string types for the index. These types are typically used for indexing data structures and increase the usability of the library. Use of the [Object](#) type should be restricted to those cases where the specific integer or string type cannot be specified at design time. If the design requires other types for the index, reconsider whether the type represents a logical data store. If it does not represent a logical data store, use a method.

How to fix violations

To fix a violation of this rule, change the index to an integer or string type or use a method instead of the indexer.

When to suppress warnings

Suppress a warning from this rule only after carefully considering the need for the nonstandard indexer.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1043
// The code that's violating the rule is on this line.
#pragma warning restore CA1043
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1043.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Design.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

Use the following option to configure which parts of your codebase to run this rule on.

- [Include specific API surfaces](#)

You can configure this option for just this rule, for all rules, or for all rules in this category ([Design](#)). For more information, see [Code quality rule configuration options](#).

Include specific API surfaces

You can configure which parts of your codebase to run this rule on, based on their accessibility. For example, to specify that the rule should run only against the non-public API surface, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXX.api_surface = private, internal
```

Example

The following example shows an indexer that uses an [Int32](#) index.

```
string[] Month = new string[] { "Jan", "Feb", "..." };

public string this[int index]
{
    get => Month[index];
}
```

```
Private month() As String = {"Jan", "Feb", "..."}

Default ReadOnly Property Item(index As Integer) As String
    Get
        Return month(index)
    End Get
End Property
```

Related rules

- [CA1024: Use properties where appropriate](#)

CA1044: Properties should not be write only

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA1044
Category	Design
Fix is breaking or non-breaking	Breaking

Cause

A property has a set accessor but not a get accessor.

By default, this rule only looks at externally visible types, but this is [configurable](#).

Rule description

Get accessors provide read access to a property and set accessors provide write access. Although it is acceptable and often necessary to have a read-only property, the design guidelines prohibit the use of write-only properties. This is because letting a user set a value and then preventing the user from viewing the value does not provide any security. Also, without read access, the state of shared objects cannot be viewed, which limits their usefulness.

How to fix violations

To fix a violation of this rule, add a get accessor to the property. Alternatively, if the behavior of a write-only property is necessary, consider converting this property to a method.

When to suppress warnings

It's recommended that you don't suppress warnings from this rule.

Configure code to analyze

Use the following option to configure which parts of your codebase to run this rule on.

- [Include specific API surfaces](#)

You can configure this option for just this rule, for all rules, or for all rules in this category ([Design](#)). For more information, see [Code quality rule configuration options](#).

Include specific API surfaces

You can configure which parts of your codebase to run this rule on, based on their accessibility. For example, to specify that the rule should run only against the non-public API surface, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.api_surface = private, internal
```

Example

In the following example, `BadClassWithWriteOnlyProperty` is a type with a write-only property.

`GoodClassWithReadWriteProperty` contains the corrected code.

```
Imports System

Namespace ca1044

    Public Class BadClassWithWriteOnlyProperty

        Dim someName As String

        ' Violates rule PropertiesShouldNotBeWriteOnly.
        WriteOnly Property Name As String
            Set
                someName = Value
            End Set
        End Property

    End Class

    Public Class GoodClassWithReadWriteProperty

        Property Name As String

    End Class

End Namespace
```

```
public class BadClassWithWriteOnlyProperty
{
    string _someName;

    // Violates rule PropertiesShouldNotBeWriteOnly.
    public string Name
    {
        set
        {
            _someName = value;
        }
    }
}

public class GoodClassWithReadWriteProperty
{
    public string Name { get; set; }
}
```

CA1045: Do not pass types by reference

9/20/2022 • 7 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA1045
Category	Design
Fix is breaking or non-breaking	Breaking

Cause

A public or protected method in a public type has a `ref` parameter that takes a primitive type, a reference type, or a value type that is not one of the built-in types.

Rule description

Passing types by reference (using `out` or `ref`) requires experience with pointers, understanding how value types and reference types differ, and handling methods that have multiple return values. Also, the difference between `out` and `ref` parameters is not widely understood.

When a reference type is passed "by reference," the method intends to use the parameter to return a different instance of the object. (Passing a reference type by reference is also known as using a double pointer, pointer to a pointer, or double indirection.) Using the default calling convention, which is pass "by value," a parameter that takes a reference type already receives a pointer to the object. The pointer, not the object to which it points, is passed by value. Passing by value means that the method cannot change the pointer to have it point to a new instance of the reference type, but can change the contents of the object to which it points. For most applications this is sufficient and yields the behavior that you want.

If a method must return a different instance, use the return value of the method to accomplish this. See the [System.String](#) class for a variety of methods that operate on strings and return a new instance of a string. By using this model, it is left to the caller to decide whether the original object is preserved.

Although return values are commonplace and heavily used, the correct application of `out` and `ref` parameters requires intermediate design and coding skills. Library architects who design for a general audience should not expect users to become proficient in working with `out` or `ref` parameters.

NOTE

When you work with parameters that are large structures, the additional resources that are required to copy these structures could cause a performance effect when you pass by value. In these cases, you might consider using `ref` or `out` parameters.

How to fix violations

To fix a violation of this rule that is caused by a value type, have the method return the object as its return value. If the method must return multiple values, redesign it to return a single instance of an object that holds the values.

To fix a violation of this rule that is caused by a reference type, make sure that the behavior that you want is to return a new instance of the reference. If it is, the method should use its return value to do this.

When to suppress warnings

It is safe to suppress a warning from this rule; however, this design could cause usability issues.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1045
// The code that's violating the rule is on this line.
#pragma warning restore CA1045
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.cs,vb]
dotnet_diagnostic.CA1045.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.cs,vb]
dotnet_analyzer_diagnostic.category-Design.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

Use the following option to configure which parts of your codebase to run this rule on.

- [Include specific API surfaces](#)

You can configure this option for just this rule, for all rules, or for all rules in this category ([Design](#)). For more information, see [Code quality rule configuration options](#).

Include specific API surfaces

You can configure which parts of your codebase to run this rule on, based on their accessibility. For example, to specify that the rule should run only against the non-public API surface, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.api_surface = private, internal
```

Example 1

The following library shows two implementations of a class that generates responses to the feedback of the user. The first implementation (`BadRefAndOut`) forces the library user to manage three return values. The second implementation (`RedesignedRefAndOut`) simplifies the user experience by returning an instance of a container class (`ReplyData`) that manages the data as a single unit.

```
public enum Actions
{
```

```

        Unknown,
        Discard,
        ForwardToManagement,
        ForwardToDeveloper
    }

    public enum TypeOfFeedback
    {
        Complaint,
        Praise,
        Suggestion,
        Incomprehensible
    }

    public class BadRefAndOut
    {
        // Violates rule: DoNotPassTypesByReference.

        public static bool ReplyInformation(TypeOfFeedback input,
            out string reply, ref Actions action)
        {
            bool returnReply = false;
            string replyText = "Your feedback has been forwarded " +
                "to the product manager.";

            reply = String.Empty;
            switch (input)
            {
                case TypeOfFeedback.Complaint:
                case TypeOfFeedback.Praise:
                    action = Actions.ForwardToManagement;
                    reply = "Thank you. " + replyText;
                    returnReply = true;
                    break;
                case TypeOfFeedback.Suggestion:
                    action = Actions.ForwardToDeveloper;
                    reply = replyText;
                    returnReply = true;
                    break;
                case TypeOfFeedback.Incomprehensible:
                default:
                    action = Actions.Discard;
                    returnReply = false;
                    break;
            }
            return returnReply;
        }
    }

    // Redesigned version does not use out or ref parameters;
    // instead, it returns this container type.

    public class ReplyData
    {
        string reply;
        Actions action;
        bool returnReply;

        // Constructors.
        public ReplyData()
        {
            this.reply = String.Empty;
            this.action = Actions.Discard;
            this.returnReply = false;
        }

        public ReplyData(Actions action, string reply, bool returnReply)
        {
            this.reply = reply;

```

```

        this.action = action;
        this.returnReply = returnReply;
    }

    // Properties.
    public string Reply { get { return reply; } }
    public Actions Action { get { return action; } }

    public override string ToString()
    {
        return String.Format("Reply: {0} Action: {1} return? {2}",
            reply, action.ToString(), returnReply.ToString());
    }
}

public class RedesignedRefAndOut
{
    public static ReplyData ReplyInformation(TypeOfFeedback input)
    {
        ReplyData answer;
        string replyText = "Your feedback has been forwarded " +
            "to the product manager.";

        switch (input)
        {
            case TypeOfFeedback.Complaint:
            case TypeOfFeedback.Praise:
                answer = new ReplyData(
                    Actions.ForwardToManagement,
                    "Thank you. " + replyText,
                    true);
                break;
            case TypeOfFeedback.Suggestion:
                answer = new ReplyData(
                    Actions.ForwardToDeveloper,
                    replyText,
                    true);
                break;
            case TypeOfFeedback.Incomprehensible:
            default:
                answer = new ReplyData();
                break;
        }
        return answer;
    }
}

```

Example 2

The following application illustrates the experience of the user. The call to the redesigned library (`UseTheSimplifiedClass` method) is more straightforward, and the information that is returned by the method is easily managed. The output from the two methods is identical.

```

public class UseComplexMethod
{
    static void UseTheComplicatedClass()
    {
        // Using the version with the ref and out parameters.
        // You do not have to initialize an out parameter.

        string[] reply = new string[5];

        // You must initialize a ref parameter.
        Actions[] action = {Actions.Unknown,Actions.Unknown,
                            Actions.Unknown,Actions.Unknown,
                            Actions.Unknown,Actions.Unknown};
        bool[] disposition = new bool[5];
        int i = 0;

        foreach (TypeOfFeedback t in Enum.GetValues(typeof(TypeOfFeedback)))
        {
            // The call to the library.
            disposition[i] = BadRefAndOut.ReplyInformation(
                t, out reply[i], ref action[i]);
            Console.WriteLine("Reply: {0} Action: {1}  return? {2} ",
                reply[i], action[i], disposition[i]);
            i++;
        }
    }

    static void UseTheSimplifiedClass()
    {
        ReplyData[] answer = new ReplyData[5];
        int i = 0;
        foreach (TypeOfFeedback t in Enum.GetValues(typeof(TypeOfFeedback)))
        {
            // The call to the library.
            answer[i] = RedesignedRefAndOut.ReplyInformation(t);
            Console.WriteLine(answer[i++]);
        }
    }

    public static void Main1045()
    {
        UseTheComplicatedClass();

        // Print a blank line in output.
        Console.WriteLine("");

        UseTheSimplifiedClass();
    }
}

```

Example 3

The following example library illustrates how `ref` parameters for reference types are used, and shows a better way to implement this functionality.

```

public class ReferenceTypesAndParameters
{
    // The following syntax will not work. You cannot make a
    // reference type that is passed by value point to a new
    // instance. This needs the ref keyword.

    public static void BadPassTheObject(string argument)
    {
        argument = argument + " ABCDE";
    }

    // The following syntax will work, but is considered bad design.
    // It reassigned the argument to point to a new instance of string.
    // Violates rule DoNotPassTypesByReference.

    public static void PassTheReference(ref string argument)
    {
        argument = argument + " ABCDE";
    }

    // The following syntax will work and is a better design.
    // It returns the altered argument as a new instance of string.

    public static string BetterThanPassTheReference(string argument)
    {
        return argument + " ABCDE";
    }
}

```

Example 4

The following application calls each method in the library to demonstrate the behavior.

```

public class Test
{
    public static void Main1045()
    {
        string s1 = "12345";
        string s2 = "12345";
        string s3 = "12345";

        Console.WriteLine("Changing pointer - passed by value:");
        Console.WriteLine(s1);
        ReferenceTypesAndParameters.BadPassTheObject(s1);
        Console.WriteLine(s1);

        Console.WriteLine("Changing pointer - passed by reference:");
        Console.WriteLine(s2);
        ReferenceTypesAndParameters.PassTheReference(ref s2);
        Console.WriteLine(s2);

        Console.WriteLine("Passing by return value:");
        s3 = ReferenceTypesAndParameters.BetterThanPassTheReference(s3);
        Console.WriteLine(s3);
    }
}

```

This example produces the following output:

```
Changing pointer - passed by value:  
12345  
12345  
Changing pointer - passed by reference:  
12345  
12345 ABCDE  
Passing by return value:  
12345 ABCDE
```

Related rules

[CA1021: Avoid out parameters](#)

CA1046: Do not overload operator equals on reference types

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA1046
Category	Design
Fix is breaking or non-breaking	Breaking

Cause

A public or nested public reference type overloads the equality operator.

Rule description

For reference types, the default implementation of the equality operator is almost always correct. By default, two references are equal only if they point to the same object.

How to fix violations

To fix a violation of this rule, remove the implementation of the equality operator.

When to suppress warnings

It is safe to suppress a warning from this rule when the reference type behaves like a built-in value type. If it is meaningful to do addition or subtraction on instances of the type, it is probably correct to implement the equality operator and suppress the violation.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1046
// The code that's violating the rule is on this line.
#pragma warning restore CA1046
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1046.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Design.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

Use the following option to configure which parts of your codebase to run this rule on.

- [Include specific API surfaces](#)

You can configure this option for just this rule, for all rules, or for all rules in this category (Design). For more information, see [Code quality rule configuration options](#).

Include specific API surfaces

You can configure which parts of your codebase to run this rule on, based on their accessibility. For example, to specify that the rule should run only against the non-public API surface, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.api_surface = private, internal
```

Example 1

The following example demonstrates the default behavior when comparing two references.

```
public class MyReferenceType
{
    private int a, b;
    public MyReferenceType(int a, int b)
    {
        this.a = a;
        this.b = b;
    }

    public override string ToString()
    {
        return String.Format("({0},{1})", a, b);
    }
}
```

Example 2

The following application compares some references.

```
public class ReferenceTypeEquality
{
    public static void Main1046()
    {
        MyReferenceType a = new MyReferenceType(2, 2);
        MyReferenceType b = new MyReferenceType(2, 2);
        MyReferenceType c = a;

        Console.WriteLine("a = new {0} and b = new {1} are equal? {2}", a, b, a.Equals(b) ? "Yes" : "No");
        Console.WriteLine("c and a are equal? {0}", c.Equals(a) ? "Yes" : "No");
        Console.WriteLine("b and a are == ? {0}", b == a ? "Yes" : "No");
        Console.WriteLine("c and a are == ? {0}", c == a ? "Yes" : "No");
    }
}
```

This example produces the following output:

```
a = new (2,2) and b = new (2,2) are equal? No
c and a are equal? Yes
b and a are == ? No
c and a are == ? Yes
```

See also

- [System.Object.Equals](#)
- [Equality Operators](#)

CA1047: Do not declare protected members in sealed types

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA1047
Category	Design
Fix is breaking or non-breaking	Non-breaking

Cause

A public type is `sealed` (`NotInheritable` in Visual basic) and declares a protected member or a protected nested type. This rule does not report violations for `Finalize` methods, which must follow this pattern.

Rule description

Types declare protected members so that inheriting types can access or override the member. By definition, you cannot inherit from a sealed type, which means that protected methods on sealed types cannot be called.

The C# compiler issues a warning for this error.

How to fix violations

To fix a violation of this rule, change the access level of the member to private, or make the type inheritable.

When to suppress warnings

Do not suppress a warning from this rule. Leaving the type in its current state can cause maintenance issues and does not provide any benefits.

Configure code to analyze

Use the following option to configure which parts of your codebase to run this rule on.

- [Include specific API surfaces](#)

You can configure this option for just this rule, for all rules, or for all rules in this category ([Design](#)). For more information, see [Code quality rule configuration options](#).

Include specific API surfaces

You can configure which parts of your codebase to run this rule on, based on their accessibility. For example, to specify that the rule should run only against the non-public API surface, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.api_surface = private, internal
```

Example

The following example shows a type that violates this rule.

```
public sealed class SealedClass
{
    protected void ProtectedMethod(){}
}
```

```
Public NotInheritable Class BadSealedType
    Protected Sub MyMethod
    End Sub
End Class
```

CA1050: Declare types in namespaces

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA1050
Category	Design
Fix is breaking or non-breaking	Breaking

Cause

A public or protected type is defined outside the scope of a named namespace.

Rule description

Types are declared in namespaces to prevent name collisions, and as a way to organize related types in an object hierarchy. Types that are outside any named namespace are in a global namespace that cannot be referenced in code.

How to fix violations

To fix a violation of this rule, place the type in a namespace.

When to suppress warnings

Although you never have to suppress a warning from this rule, it is safe to do this when the assembly will never be used together with other assemblies.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1050
// The code that's violating the rule is on this line.
#pragma warning restore CA1050
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1050.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Design.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Example 1

The following example shows a library that has a type incorrectly declared outside a namespace, and a type that has the same name declared in a namespace.

```
// Violates rule: DeclareTypesInNamespaces.
using System;

public class Test
{
    public override string ToString()
    {
        return "Test does not live in a namespace!";
    }
}

namespace ca1050
{
    public class Test
    {
        public override string ToString()
        {
            return "Test lives in a namespace!";
        }
    }
}
```

```
' Violates rule: DeclareTypesInNamespaces.
Public Class Test

    Public Overrides Function ToString() As String
        Return "Test does not live in a namespace!"
    End Function

End Class

Namespace ca1050

    Public Class Test

        Public Overrides Function ToString() As String
            Return "Test lives in a namespace!"
        End Function

    End Class

End Namespace
```

Example 2

The following application uses the library that was defined previously. The type that's declared outside a namespace is created when the name `Test` is not qualified by a namespace. To access the `Test` type that's declared inside a namespace, the namespace name is required.

```
public class MainHolder
{
    public static void Main1050()
    {
        Test t1 = new Test();
        Console.WriteLine(t1.ToString());

        ca1050.Test t2 = new ca1050.Test();
        Console.WriteLine(t2.ToString());
    }
}
```

```
Public Class MainHolder

    Public Shared Sub Main1050()
        Dim t1 As New Test()
        Console.WriteLine(t1.ToString())

        Dim t2 As New ca1050.Test()
        Console.WriteLine(t2.ToString())
    End Sub

End Class
```

CA1051: Do not declare visible instance fields

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA1051
Category	Design
Fix is breaking or non-breaking	Breaking

Cause

A type has a non-private instance field.

By default, this rule only looks at externally visible types, but this is [configurable](#).

Rule description

The primary use of a field should be as an implementation detail. Fields should be `private` or `internal` and should be exposed by using properties. It's as easy to access a property as it is to access a field, and the code in the accessors of a property can change as the features of the type expand without introducing breaking changes.

Properties that just return the value of a private or internal field are optimized to perform on par with accessing a field; the performance gain from using externally visible fields instead of properties is minimal. *Externally visible* refers to `public`, `protected`, and `protected internal` (`Public`, `Protected`, and `Protected Friend` in Visual Basic) accessibility levels.

Additionally, public fields cannot be protected by [Link demands](#). (Link demands don't apply to .NET Core apps.)

How to fix violations

To fix a violation of this rule, make the field `private` or `internal` and expose it by using an externally visible property.

When to suppress warnings

Only suppress this warning if you're certain that consumers need direct access to the field. For most applications, exposed fields do not provide performance or maintainability benefits over properties.

Consumers may need field access in the following situations:

- In ASP.NET Web Forms content controls.
- When the target platform makes use of `ref` to modify fields, such as model-view-viewmodel (MVVM) frameworks for WPF and UWP.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1051
// The code that's violating the rule is on this line.
#pragma warning restore CA1051
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1051.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Design.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Include or exclude APIs

Use the following options to configure which parts of your codebase to run this rule on.

- [Include specific API surfaces](#)
- [Exclude structs](#)

You can configure these options for just this rule, for all rules, or for all rules in this category ([Design](#)). For more information, see [Code quality rule configuration options](#).

Include specific API surfaces

You can configure which parts of your codebase to run this rule on, based on their accessibility. For example, to specify that the rule should run only against the non-public API surface, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.api_surface = private, internal
```

Exclude structs

You can exclude `struct` (`Structure` in Visual Basic) fields from being analyzed.

```
dotnet_code_quality.ca1051.exclude_structs = true
```

Example

The following example shows a type (`BadPublicInstanceFields`) that violates this rule. `GoodPublicInstanceFields` shows the corrected code.

```
public class BadPublicInstanceFields
{
    // Violates rule DoNotDeclareVisibleInstanceFields.
    public int instanceData = 32;
}

public class GoodPublicInstanceFields
{
    private int instanceData = 32;

    public int InstanceData
    {
        get { return instanceData; }
        set { instanceData = value; }
    }
}
```

See also

- [Link Demands](#)

CA1052: Static holder types should be Static or NotInheritable

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA1052
Category	Design
Fix is breaking or non-breaking	Breaking

Cause

A non-abstract type contains only static members (other than a possible default constructor) and is not declared with the [static](#) or [Shared](#) modifier.

By default, this rule only looks at externally visible types, but this is [configurable](#).

Rule description

Rule CA1052 assumes that a type that contains only static members is not designed to be inherited, because the type does not provide any functionality that can be overridden in a derived type. A type that is not meant to be inherited should be marked with the `static` modifier in C# to prohibit its use as a base type. Additionally, its default constructor should be removed. In Visual Basic, the class should be converted to a [module](#).

This rule does not fire for abstract classes or classes that have a base class. However, the rule does fire for classes that support an empty interface.

NOTE

In the latest analyzer implementation of this rule, it also encompasses the functionality of [rule CA1053](#).

How to fix violations

To fix a violation of this rule, mark the type as `static` and remove the default constructor (C#), or convert it to a module (Visual Basic).

When to suppress warnings

You can suppress violations in the following cases:

- The type is designed to be inherited. The absence of the `static` modifier suggests that the type is useful as a base type.
- The type is used as a type argument. Static types can't be used as type arguments.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then

re-enable the rule.

```
#pragma warning disable CA1052
// The code that's violating the rule is on this line.
#pragma warning restore CA1052
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.cs,vb]
dotnet_diagnostic.CA1052.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.cs,vb]
dotnet_analyzer_diagnostic.category-Design.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

Use the following option to configure which parts of your codebase to run this rule on.

- [Include specific API surfaces](#)

You can configure this option for just this rule, for all rules, or for all rules in this category ([Design](#)). For more information, see [Code quality rule configuration options](#).

Include specific API surfaces

You can configure which parts of your codebase to run this rule on, based on their accessibility. For example, to specify that the rule should run only against the non-public API surface, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXX.api_surface = private, internal
```

Example of a violation

The following example shows a type that violates the rule:

```
public class StaticMembers
{
    public static int SomeProperty { get; set; }
    public static void SomeMethod() { }
}
```

```
Imports System

Namespace ca1052

    Public Class StaticMembers

        Shared Property SomeProperty As Integer

        Private Sub New()
        End Sub

        Shared Sub SomeMethod()
        End Sub

    End Class

End Namespace
```

Fix with the static modifier

The following example shows how to fix a violation of this rule by marking the type with the `static` modifier in C#:

```
public static class StaticMembers
{
    public static int SomeProperty { get; set; }
    public static void SomeMethod() { }
}
```

CA1053: Static holder types should not have default constructors

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA1053
Category	Design
Fix is breaking or non-breaking	Breaking

NOTE

Rule CA1053 only applies to legacy Visual Studio code analysis. In the .NET code-quality analyzers, it's combined into rule [CA1052: Static holder types should be Static or NotInheritable](#).

Cause

A public or nested public type declares only static members and has a default constructor.

Rule description

The default constructor is unnecessary because calling static members does not require an instance of the type. Also, because the type does not have non-static members, creating an instance does not provide access to any of the type's members.

How to fix violations

To fix a violation of this rule, remove the default constructor.

When to suppress warnings

Do not suppress a warning from this rule. The presence of the default constructor suggests that the type is not a static type.

CA1054: URI parameters should not be strings

9/20/2022 • 5 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA1054
Category	Design
Fix is breaking or non-breaking	Breaking

Cause

A type declares a method with a string parameter whose name contains "uri", "Uri", "urn", "Urn", "url", or "Url", and the type does not declare a corresponding overload that takes a [System.Uri](#) parameter.

By default, this rule only looks at externally visible types, but this is [configurable](#).

Rule description

This rule splits the parameter name into tokens based on the camel casing convention and checks whether each token equals "uri", "Uri", "urn", "Urn", "url", or "Url". If there's a match, the rule assumes that the parameter represents a uniform resource identifier (URI). A string representation of a URI is prone to parsing and encoding errors, and can lead to security vulnerabilities. If a method takes a string representation of a URI, a corresponding overload should be provided that takes an instance of the [Uri](#) class, which provides these services in a safe and secure manner.

How to fix violations

To fix a violation of this rule, change the parameter to a [Uri](#) type; this is a breaking change. Alternately, provide an overload of the method that takes a [Uri](#) parameter; this is a non-breaking change.

When to suppress warnings

It's safe to suppress a warning from this rule if the parameter doesn't represent a URI.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1054
// The code that's violating the rule is on this line.
#pragma warning restore CA1054
```

To disable the rule for a file, folder, or project, set its severity to [none](#) in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1054.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Design.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

Use the following options to configure which parts of your codebase to run this rule on.

- [Include specific API surfaces](#)
- [Exclude specific symbols](#)
- [Exclude specific types and their derived types](#)

You can configure these options for just this rule, for all rules, or for all rules in this category ([Design](#)). For more information, see [Code quality rule configuration options](#).

Include specific API surfaces

You can configure which parts of your codebase to run this rule on, based on their accessibility. For example, to specify that the rule should run only against the non-public API surface, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.api_surface = private, internal
```

Exclude specific symbols

You can exclude specific symbols, such as types and methods, from analysis. For example, to specify that the rule should not run on any code within types named `MyType`, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Symbol name only (includes all symbols with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#). Each symbol name requires a symbol-kind prefix, such as `M:` for methods, `T:` for types, and `N:` for namespaces.
- `.ctor` for constructors and `.cctor` for static constructors.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType</code>	Matches all symbols named <code>MyType</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType1 MyType2</code>	Matches all symbols named either <code>MyType1</code> or <code>MyType2</code> .

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS.MyType.MyMethod(ParamType)</code>	Matches specific method <code>MyMethod</code> with the specified fully qualified signature.
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS1.MyType1.MyMethod1(ParamType) M:NS2.MyType2.MyMethod2(ParamType)</code>	Matches specific methods <code>MyMethod1</code> and <code>MyMethod2</code> with the respective fully qualified signatures.

Exclude specific types and their derived types

You can exclude specific types and their derived types from analysis. For example, to specify that the rule should not run on any methods within types named `MyType` and their derived types, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Type name only (includes all types with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#), with an optional `T:` prefix.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType</code>	Matches all types named <code>MyType</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType1 MyType2</code>	Matches all types named either <code>MyType1</code> or <code>MyType2</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS.MyType</code>	Matches specific type <code>MyType</code> with given fully qualified name and all of its derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS1.MyType1 M:NS2.MyType2</code>	Matches specific types <code>MyType1</code> and <code>MyType2</code> with the respective fully qualified names, and all of their derived types.

Example

The following example shows a type, `ErrorProne`, that violates this rule, and a type, `SaferWay`, that satisfies the rule.

```
public class ErrorProne
{
    // Violates rule UriPropertiesShouldNotBeStrings.
    public string SomeUri { get; set; }

    // Violates rule UriParametersShouldNotBeStrings.
    public void AddToHistory(string uriString) { }

    // Violates rule UriReturnValuesShouldNotBeStrings.
    public string GetRefererUri(string httpHeader)
    {
        return "http://www.adventure-works.com";
    }
}

public class SaferWay
{
    // To retrieve a string, call SomeUri.ToString().
    // To set using a string, call SomeUri = new Uri(string).
    public Uri SomeUri { get; set; }

    public void AddToHistory(string uriString)
    {
        // Check for UriFormatException.
        AddToHistory(new Uri(uriString));
    }

    public void AddToHistory(Uri uriType) { }

    public Uri GetRefererUri(string httpHeader)
    {
        return new Uri("http://www.adventure-works.com");
    }
}
```

```

Imports System

Namespace ca1054

    Public Class ErrorProne

        ' Violates rule UriPropertiesShouldNotBeStrings.
        Property SomeUri As String

        ' Violates rule UriParametersShouldNotBeStrings.
        Sub AddToHistory(uriString As String)
            End Sub

        ' Violates rule UriReturnValuesShouldNotBeStrings.
        Function GetRefererUri(httpHeader As String) As String
            Return "http://www.adventure-works.com"
        End Function

    End Class

    Public Class SaferWay

        ' To retrieve a string, call SomeUri.ToString().
        ' To set using a string, call SomeUri = New Uri(string).
        Property SomeUri As Uri

        Sub AddToHistory(uriString As String)
            ' Check for UriFormatException.
            AddToHistory(New Uri(uriString))
        End Sub

        Sub AddToHistory(uriString As Uri)
        End Sub

        Function GetRefererUri(httpHeader As String) As Uri
            Return New Uri("http://www.adventure-works.com")
        End Function

    End Class

End Namespace

```

Related rules

- [CA1056: URI properties should not be strings](#)
- [CA1055: URI return values should not be strings](#)
- [CA2234: Pass System.Uri objects instead of strings](#)

CA1055: URI return values should not be strings

9/20/2022 • 4 minutes to read • [Edit Online](#)

	Value
Rule ID	CA1055
Category	Design
Fix is breaking or non-breaking	Breaking

Cause

The name of a method contains "uri", "Uri", "urn", "Urn", "url", or "Url", and the method returns a string.

By default, this rule only looks at externally visible methods, but this is [configurable](#).

Rule description

This rule splits the method name into tokens based on the Pascal casing convention and checks whether each token equals "uri", "Uri", "urn", "Urn", "url", or "Url". If there's a match, the rule assumes that the method returns a uniform resource identifier (URI). A string representation of a URI is prone to parsing and encoding errors, and can lead to security vulnerabilities. The [System.Uri](#) class provides these services in a safe and secure manner.

How to fix violations

To fix a violation of this rule, change the return type to a [Uri](#).

When to suppress warnings

It's safe to suppress a warning from this rule if the return value does not represent a URI.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1055
// The code that's violating the rule is on this line.
#pragma warning restore CA1055
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1055.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Design.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

Use the following options to configure which parts of your codebase to run this rule on.

- [Include specific API surfaces](#)
- [Exclude specific symbols](#)
- [Exclude specific types and their derived types](#)

You can configure these options for just this rule, for all rules, or for all rules in this category ([Design](#)). For more information, see [Code quality rule configuration options](#).

Include specific API surfaces

You can configure which parts of your codebase to run this rule on, based on their accessibility. For example, to specify that the rule should run only against the non-public API surface, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.api_surface = private, internal
```

Exclude specific symbols

You can exclude specific symbols, such as types and methods, from analysis. For example, to specify that the rule should not run on any code within types named `MyType`, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Symbol name only (includes all symbols with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#). Each symbol name requires a symbol-kind prefix, such as `M:` for methods, `T:` for types, and `N:` for namespaces.
- `.ctor` for constructors and `.cctor` for static constructors.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType</code>	Matches all symbols named <code>MyType</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType1 MyType2</code>	Matches all symbols named either <code>MyType1</code> or <code>MyType2</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS.MyType.MyMethod(ParamType)</code>	Matches specific method <code>MyMethod</code> with the specified fully qualified signature.
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS1.MyType1.MyMethod1(ParamType) M:NS2.MyType2.MyMethod2(ParamType)</code>	Matches specific methods <code>MyMethod1</code> and <code>MyMethod2</code> with the respective fully qualified signatures.

Exclude specific types and their derived types

You can exclude specific types and their derived types from analysis. For example, to specify that the rule should not run on any methods within types named `MyType` and their derived types, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Type name only (includes all types with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#), with an optional `T:` prefix.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType</code>	Matches all types named <code>MyType</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType1 MyType2</code>	Matches all types named either <code>MyType1</code> or <code>MyType2</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS.MyType</code>	Matches specific type <code>MyType</code> with given fully qualified name and all of its derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS1.MyType1 M:NS2.MyType2</code>	Matches specific types <code>MyType1</code> and <code>MyType2</code> with the respective fully qualified names, and all of their derived types.

Example

The following example shows a type, `ErrorProne`, that violates this rule, and a type, `SaferWay`, that satisfies the rule.

```
public class ErrorProne
{
    // Violates rule UriPropertiesShouldNotBeStrings.
    public string SomeUri { get; set; }

    // Violates rule UriParametersShouldNotBeStrings.
    public void AddToHistory(string uriString) { }

    // Violates rule UriReturnValuesShouldNotBeStrings.
    public string GetReferrerUri(string httpHeader)
    {
        return "http://www.adventure-works.com";
    }
}

public class SaferWay
{
    // To retrieve a string, call SomeUri.ToString().
    // To set using a string, call SomeUri = new Uri(string).
    public Uri SomeUri { get; set; }

    public void AddToHistory(string uriString)
    {
        // Check for UriFormatException.
        AddToHistory(new Uri(uriString));
    }

    public void AddToHistory(Uri uriType) { }

    public Uri GetReferrerUri(string httpHeader)
    {
        return new Uri("http://www.adventure-works.com");
    }
}
```

```

Imports System

Namespace ca1055

    Public Class ErrorProne

        ' Violates rule UriPropertiesShouldNotBeStrings.
        Property SomeUri As String

        ' Violates rule UriParametersShouldNotBeStrings.
        Sub AddToHistory(uriString As String)
            End Sub

        ' Violates rule UriReturnValuesShouldNotBeStrings.
        Function GetRefererUri(httpHeader As String) As String
            Return "http://www.adventure-works.com"
        End Function

    End Class

    Public Class SaferWay

        ' To retrieve a string, call SomeUri.ToString().
        ' To set using a string, call SomeUri = New Uri(string).
        Property SomeUri As Uri

        Sub AddToHistory(uriString As String)
            ' Check for UriFormatException.
            AddToHistory(New Uri(uriString))
        End Sub

        Sub AddToHistory(uriString As Uri)
        End Sub

        Function GetRefererUri(httpHeader As String) As Uri
            Return New Uri("http://www.adventure-works.com")
        End Function

    End Class

End Namespace

```

Related rules

- [CA1056: URI properties should not be strings](#)
- [CA1054: URI parameters should not be strings](#)
- [CA2234: Pass System.Uri objects instead of strings](#)

CA1056: URI properties should not be strings

9/20/2022 • 4 minutes to read • [Edit Online](#)

	Value
Rule ID	CA1056
Category	Design
Fix is breaking or non-breaking	Breaking

Cause

A type declares a string property whose name contains "uri", "Uri", "urn", "Urn", "url", or "Url".

By default, this rule only looks at externally visible types, but this is [configurable](#).

Rule description

This rule splits the property name into tokens based on the Pascal casing convention and checks whether each token equals "uri", "Uri", "urn", "Urn", "url", or "Url". If there's a match, the rule assumes that the property represents a uniform resource identifier (URI). A string representation of a URI is prone to parsing and encoding errors, and can lead to security vulnerabilities. The [System.Uri](#) class provides these services in a safe and secure manner.

How to fix violations

To fix a violation of this rule, change the property to a [Uri](#) type.

When to suppress warnings

It's safe to suppress a warning from this rule if the property doesn't represent a URI.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1056
// The code that's violating the rule is on this line.
#pragma warning restore CA1056
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1056.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Design.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

Use the following options to configure which parts of your codebase to run this rule on.

- [Include specific API surfaces](#)
- [Exclude specific symbols](#)
- [Exclude specific types and their derived types](#)

You can configure these options for just this rule, for all rules, or for all rules in this category ([Design](#)). For more information, see [Code quality rule configuration options](#).

Include specific API surfaces

You can configure which parts of your codebase to run this rule on, based on their accessibility. For example, to specify that the rule should run only against the non-public API surface, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.api_surface = private, internal
```

Exclude specific symbols

You can exclude specific symbols, such as types and methods, from analysis. For example, to specify that the rule should not run on any code within types named `MyType`, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Symbol name only (includes all symbols with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#). Each symbol name requires a symbol-kind prefix, such as `M:` for methods, `T:` for types, and `N:` for namespaces.
- `.ctor` for constructors and `.cctor` for static constructors.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType</code>	Matches all symbols named <code>MyType</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType1 MyType2</code>	Matches all symbols named either <code>MyType1</code> or <code>MyType2</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS.MyType.MyMethod(ParamType)</code>	Matches specific method <code>MyMethod</code> with the specified fully qualified signature.
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS1.MyType1.MyMethod1(ParamType) M:NS2.MyType2.MyMethod2(ParamType)</code>	Matches specific methods <code>MyMethod1</code> and <code>MyMethod2</code> with the respective fully qualified signatures.

Exclude specific types and their derived types

You can exclude specific types and their derived types from analysis. For example, to specify that the rule should not run on any methods within types named `MyType` and their derived types, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Type name only (includes all types with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#), with an optional `T:` prefix.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType</code>	Matches all types named <code>MyType</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType1 MyType2</code>	Matches all types named either <code>MyType1</code> or <code>MyType2</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS.MyType</code>	Matches specific type <code>MyType</code> with given fully qualified name and all of its derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS1.MyType1 M:NS2.MyType2</code>	Matches specific types <code>MyType1</code> and <code>MyType2</code> with the respective fully qualified names, and all of their derived types.

Example

The following example shows a type, `ErrorProne`, that violates this rule, and a type, `SaferWay`, that satisfies the rule.

```
public class ErrorProne
{
    // Violates rule UriPropertiesShouldNotBeStrings.
    public string SomeUri { get; set; }

    // Violates rule UriParametersShouldNotBeStrings.
    public void AddToHistory(string uriString) { }

    // Violates rule UriReturnValuesShouldNotBeStrings.
    public string GetRefererUri(string httpHeader)
    {
        return "http://www.adventure-works.com";
    }
}

public class SaferWay
{
    // To retrieve a string, call SomeUri.ToString().
    // To set using a string, call SomeUri = new Uri(string).
    public Uri SomeUri { get; set; }

    public void AddToHistory(string uriString)
    {
        // Check for UriFormatException.
        AddToHistory(new Uri(uriString));
    }

    public void AddToHistory(Uri uriType) { }

    public Uri GetRefererUri(string httpHeader)
    {
        return new Uri("http://www.adventure-works.com");
    }
}
```

```

Imports System

Namespace ca1056

    Public Class ErrorProne

        ' Violates rule UriPropertiesShouldNotBeStrings.
        Property SomeUri As String

        ' Violates rule UriParametersShouldNotBeStrings.
        Sub AddToHistory(uriString As String)
            End Sub

        ' Violates rule UriReturnValuesShouldNotBeStrings.
        Function GetRefererUri(httpHeader As String) As String
            Return "http://www.adventure-works.com"
        End Function

    End Class

    Public Class SaferWay

        ' To retrieve a string, call SomeUri.ToString().
        ' To set using a string, call SomeUri = New Uri(string).
        Property SomeUri As Uri

        Sub AddToHistory(uriString As String)
            ' Check for UriFormatException.
            AddToHistory(New Uri(uriString))
        End Sub

        Sub AddToHistory(uriString As Uri)
        End Sub

        Function GetRefererUri(httpHeader As String) As Uri
            Return New Uri("http://www.adventure-works.com")
        End Function

    End Class

End Namespace

```

Related rules

- [CA1054: URI parameters should not be strings](#)
- [CA1055: URI return values should not be strings](#)
- [CA2234: Pass System.Uri objects instead of strings](#)

CA1058: Types should not extend certain base types

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA1058
Category	Design
Fix is breaking or non-breaking	Breaking

Cause

A type extends one of the following base types:

- [System.ApplicationException](#)
- [System.Xml.XmlDocument](#)
- [System.Collections.CollectionBase](#)
- [System.Collections.DictionaryBase](#)
- [System.Collections.Queue](#)
- [System.Collections.ReadOnlyCollectionBase](#)
- [System.Collections.SortedList](#)
- [System.Collections.Stack](#)

By default, this rule only looks at externally visible types, but this is [configurable](#).

Rule description

Exceptions should derive from [System.Exception](#) or one of its subclasses in the [System](#) namespace.

Do not create a subclass of [XmlDocument](#) if you want to create an XML view of an underlying object model or data source.

Non-generic collections

Use and/or extend generic collections whenever possible. Do not extend non-generic collections in your code, unless you shipped it previously.

Examples of Incorrect Usage

```
public class MyCollection : CollectionBase
{
}

public class MyReadOnlyCollection : ReadOnlyCollectionBase
{
}
```

Examples of Correct Usage

```
public class MyCollection : Collection<T>
{
}

public class MyReadOnlyCollection : ReadOnlyCollection<T>
{
}
```

How to fix violations

To fix a violation of this rule, derive the type from a different base type or a generic collection.

When to suppress warnings

Do not suppress a warning from this rule for violations about [ApplicationException](#). It is safe to suppress a warning from this rule for violations about [XmlDocument](#). It is safe to suppress a warning about a non-generic collection if the code was released previously.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1058
// The code that's violating the rule is on this line.
#pragma warning restore CA1058
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1058.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Design.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

Use the following option to configure which parts of your codebase to run this rule on.

- [Include specific API surfaces](#)

You can configure this option for just this rule, for all rules, or for all rules in this category ([Design](#)). For more information, see [Code quality rule configuration options](#).

Include specific API surfaces

You can configure which parts of your codebase to run this rule on, based on their accessibility. For example, to specify that the rule should run only against the non-public API surface, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXX.api_surface = private, internal
```

CA1060: Move P/Invokes to NativeMethods class

9/20/2022 • 5 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA1060
Category	Design
Fix is breaking or non-breaking	Breaking

Cause

A method uses Platform Invocation Services to access unmanaged code and is not a member of one of the **NativeMethods** classes.

Rule description

Platform Invocation methods, such as those that are marked by using the [System.Runtime.InteropServices.DllImportAttribute](#) attribute, or methods that are defined by using the [Declare](#) keyword in Visual Basic, access unmanaged code. These methods should be in one of the following classes:

- **NativeMethods** - This class does not suppress stack walks for unmanaged code permission. ([System.Security.SuppressUnmanagedCodeSecurityAttribute](#) must not be applied to this class.) This class is for methods that can be used anywhere because a stack walk will be performed.
- **SafeNativeMethods** - This class suppresses stack walks for unmanaged code permission. ([System.Security.SuppressUnmanagedCodeSecurityAttribute](#) is applied to this class.) This class is for methods that are safe for anyone to call. Callers of these methods are not required to perform a full security review to make sure that the usage is secure because the methods are harmless for any caller.
- **UnsafeNativeMethods** - This class suppresses stack walks for unmanaged code permission. ([System.Security.SuppressUnmanagedCodeSecurityAttribute](#) is applied to this class.) This class is for methods that are potentially dangerous. Any caller of these methods must perform a full security review to make sure that the usage is secure because no stack walk will be performed.

These classes are declared as `internal` (`Friend` in Visual Basic) and declare a private constructor to prevent new instances from being created. The methods in these classes should be `static` and `internal` (`Shared` and `Friend` in Visual Basic).

How to fix violations

To fix a violation of this rule, move the method to the appropriate **NativeMethods** class. For most applications, moving P/Invokes to a new class that is named **NativeMethods** is enough.

However, if you are developing libraries for use in other applications, you should consider defining two other classes that are called **SafeNativeMethods** and **UnsafeNativeMethods**. These classes resemble the **NativeMethods** class; however, they are marked by using a special attribute called [SuppressUnmanagedCodeSecurityAttribute](#). When this attribute is applied, the runtime does not perform a full stack walk to make sure that all callers have the **UnmanagedCode** permission. The runtime ordinarily

checks for this permission at startup. Because the check is not performed, it can greatly improve performance for calls to these unmanaged methods. It also enables code that has limited permissions to call these methods.

However, you should use this attribute with great care. It can have serious security implications if it is implemented incorrectly.

For information about how to implement the methods, see the **NativeMethods** example, **SafeNativeMethods** example, and **UnsafeNativeMethods** example.

When to suppress warnings

Do not suppress a warning from this rule.

Example

The following example declares a method that violates this rule. To correct the violation, the **RemoveDirectory** P/Invoke should be moved to an appropriate class that is designed to hold only P/Invokes.

```
' Violates rule: MovePIvokesToNativeMethodsClass.  
Friend Class UnmanagedApi  
    Friend Declare Function RemoveDirectory Lib "kernel32" (  
        ByVal Name As String) As Boolean  
End Class
```

```
// Violates rule: MovePIvokesToNativeMethodsClass.  
internal class UnmanagedApi  
{  
    [DllImport("kernel32.dll", CharSet = CharSet.Unicode)]  
    internal static extern bool RemoveDirectory(string name);  
}
```

NativeMethods example

Because the **NativeMethods** class should not be marked by using **SuppressUnmanagedCodeSecurityAttribute**, P/Invokes that are put in it will require **UnmanagedCode** permission. Because most applications run from the local computer and run together with full trust, this is usually not a problem. However, if you are developing reusable libraries, you should consider defining a **SafeNativeMethods** or **UnsafeNativeMethods** class.

The following example shows an **Interaction.Beep** method that wraps the **MessageBeep** function from user32.dll. The **MessageBeep** P/Invoke is put in the **NativeMethods** class.

```

Public NotInheritable Class Interaction

    Private Sub New()
        End Sub

        ' Callers require Unmanaged permission
        Public Shared Sub Beep()
            ' No need to demand a permission as callers of Interaction.Beep
            ' will require UnmanagedCode permission
            If Not NativeMethods.MessageBeep(-1) Then
                Throw New Win32Exception()
            End If

        End Sub

    End Class

    Friend NotInheritable Class NativeMethods

        Private Sub New()
            End Sub

            <DllImport("user32.dll", CharSet:=CharSet.Auto)>
            Friend Shared Function MessageBeep(ByVal uType As Integer) As <MarshalAs(UnmanagedType.Bool)> Boolean
            End Function

    End Class

```

```

public static class Interaction
{
    // Callers require Unmanaged permission
    public static void Beep()
    {
        // No need to demand a permission as callers of Interaction.Beep
        // will require UnmanagedCode permission
        if (!NativeMethods.MessageBeep(-1))
            throw new Win32Exception();
    }
}

internal static class NativeMethods
{
    [DllImport("user32.dll", CharSet = CharSet.Auto)]
    [return: MarshalAs(UnmanagedType.Bool)]
    internal static extern bool MessageBeep(int uType);
}

```

SafeNativeMethods example

P/Invoke methods that can be safely exposed to any application and that do not have any side effects should be put in a class that is named **SafeNativeMethods**. You do not have to pay much attention to where they are called from.

The following example shows an **Environment.TickCount** property that wraps the **GetTickCount** function from kernel32.dll.

```

Public NotInheritable Class Environment

    Private Sub New()
        End Sub

        ' Callers do not require Unmanaged permission
        Public Shared ReadOnly Property TickCount() As Integer
            Get
                ' No need to demand a permission in place of
                ' UnmanagedType as GetTickCount is considered
                ' a safe method
                Return SafeNativeMethods.GetTickCount()
            End Get
        End Property

    End Class

<SuppressUnmanagedCodeSecurityAttribute()>
Friend NotInheritable Class SafeNativeMethods

    Private Sub New()
        End Sub

    <DllImport("kernel32.dll", CharSet:=CharSet.Auto, ExactSpelling:=True)>
    Friend Shared Function GetTickCount() As Integer
        End Function

    End Class

```

```

public static class Environment
{
    // Callers do not require UnmanagedType permission
    public static int TickCount
    {
        get
        {
            // No need to demand a permission in place of
            // UnmanagedType as GetTickCount is considered
            // a safe method
            return SafeNativeMethods.GetTickCount();
        }
    }
}

[SuppressUnmanagedCodeSecurityAttribute]
internal static class SafeNativeMethods
{
    [DllImport("kernel32.dll", CharSet = CharSet.Auto, ExactSpelling = true)]
    internal static extern int GetTickCount();
}

```

UnsafeNativeMethods example

P/Invoke methods that cannot be safely called and that could cause side effects should be put in a class that is named **UnsafeNativeMethods**. These methods should be rigorously checked to make sure that they are not exposed to the user unintentionally.

The following example shows a **Cursor.Hide** method that wraps the **ShowCursor** function from user32.dll.

```

Public NotInheritable Class Cursor

    Private Sub New()
        End Sub

        ' Callers do not require Unmanaged permission, however,
        ' they do require UIPermission.AllWindows
        Public Shared Sub Hide()
            ' Need to demand an appropriate permission
            ' in place of UnmanagedCode permission as
            ' ShowCursor is not considered a safe method
            Dim permission As New UIPermission(UIPermissionWindow.AllWindows)
            permission.Demand()
            UnsafeNativeMethods.ShowCursor(False)

        End Sub

    End Class

<SuppressUnmanagedCodeSecurityAttribute()>
Friend NotInheritable Class UnsafeNativeMethods

    Private Sub New()
        End Sub

        <DllImport("user32.dll", CharSet:=CharSet.Auto, ExactSpelling:=True)>
        Friend Shared Function ShowCursor(<MarshalAs(UnmanagedType.Bool)> ByVal bShow As Boolean) As Integer
            End Function

    End Class

```

```

public static class Cursor
{
    public static void Hide()
    {
        UnsafeNativeMethods.ShowCursor(false);
    }
}

[SuppressUnmanagedCodeSecurityAttribute]
internal static class UnsafeNativeMethods
{
    [DllImport("user32.dll", CharSet = CharSet.Auto, ExactSpelling = true)]
    internal static extern int ShowCursor([MarshalAs(UnmanagedType.Bool)] bool bShow);
}

```

See also

- [Design rules](#)

CA1061: Do not hide base class methods

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA1061
Category	Design
Fix is breaking or non-breaking	Breaking

Cause

A derived type declares a method with the same name and with the same number of parameters as one of its base methods; one or more of the parameters is a base type of the corresponding parameter in the base method; and any remaining parameters have types that are identical to the corresponding parameters in the base method.

Rule description

A method in a base type is hidden by an identically named method in a derived type when the parameter signature of the derived method differs only by types that are more weakly derived than the corresponding types in the parameter signature of the base method.

How to fix violations

To fix a violation of this rule, remove or rename the method, or change the parameter signature so that the method does not hide the base method.

When to suppress warnings

Do not suppress a warning from this rule.

Example

The following example shows a method that violates the rule.

```
class BaseType
{
    internal void MethodOne(string inputOne, object inputTwo)
    {
        Console.WriteLine("Base: {0}, {1}", inputOne, inputTwo);
    }

    internal void MethodTwo(string inputOne, string inputTwo)
    {
        Console.WriteLine("Base: {0}, {1}", inputOne, inputTwo);
    }
}

class DerivedType : BaseType
{
    internal void MethodOne(string inputOne, string inputTwo)
    {
        Console.WriteLine("Derived: {0}, {1}", inputOne, inputTwo);
    }

    // This method violates the rule.
    internal void MethodTwo(string inputOne, object inputTwo)
    {
        Console.WriteLine("Derived: {0}, {1}", inputOne, inputTwo);
    }
}

class Test
{
    static void Main1061()
    {
        DerivedType derived = new DerivedType();

        // Calls DerivedType.MethodOne.
        derived.MethodOne("string1", "string2");

        // Calls BaseType.MethodOne.
        derived.MethodOne("string1", (object)"string2");

        // Both of these call DerivedType.MethodTwo.
        derived.MethodTwo("string1", "string2");
        derived.MethodTwo("string1", (object)"string2");
    }
}
```

CA1062: Validate arguments of public methods

9/20/2022 • 6 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA1062
Category	Design
Fix is breaking or non-breaking	Non-breaking

Cause

An externally visible method dereferences one of its reference arguments without verifying whether that argument is `null` (`Nothing` in Visual Basic).

You can [configure](#) this rule to exclude certain types and parameters from analysis. You can also [indicate null-check validation methods](#).

Rule description

All reference arguments that are passed to externally visible methods should be checked against `null`. If appropriate, throw an [ArgumentNullException](#) when the argument is `null`.

If a method can be called from an unknown assembly because it is declared public or protected, you should validate all parameters of the method. If the method is designed to be called only by known assemblies, mark the method `internal` and apply the [InternalsVisibleToAttribute](#) attribute to the assembly that contains the method.

How to fix violations

To fix a violation of this rule, validate each reference argument against `null`.

When to suppress warnings

You can suppress a warning from this rule if you are sure that the dereferenced parameter has been validated by another method call in the function.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1062
// The code that's violating the rule is on this line.
#pragma warning restore CA1062
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1062.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Design.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

Use the following options to configure which parts of your codebase to run this rule on.

- [Include specific API surfaces](#)
- [Exclude specific symbols](#)
- [Exclude specific types and their derived types](#)
- [Exclude extension method 'this' parameter](#)
- [Null check validation methods](#)

These options can be configured for just this rule, for all rules, or for all rules in this category ([Design](#)). For more information, see [Code quality rule configuration options](#).

Include specific API surfaces

You can configure which parts of your codebase to run this rule on, based on their accessibility. For example, to specify that the rule should run only against the non-public API surface, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.api_surface = private, internal
```

NOTE

This option is supported for CA1062 in .NET 7 and later versions only.

Exclude specific symbols

You can exclude specific symbols, such as types and methods, from analysis. For example, to specify that the rule should not run on any code within types named `MyType`, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Symbol name only (includes all symbols with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#). Each symbol name requires a symbol-kind prefix, such as `M:` for methods, `T:` for types, and `N:` for namespaces.
- `.ctor` for constructors and `.cctor` for static constructors.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType</code>	Matches all symbols named <code>MyType</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType1 MyType2</code>	Matches all symbols named either <code>MyType1</code> or <code>MyType2</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS.MyType.MyMethod(ParamType)</code>	Matches specific method <code>MyMethod</code> with the specified fully qualified signature.
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS1.MyType1.MyMethod1(ParamType) M:NS2.MyType2.MyMethod2(ParamType)</code>	Matches specific methods <code>MyMethod1</code> and <code>MyMethod2</code> with the respective fully qualified signatures.

Exclude specific types and their derived types

You can exclude specific types and their derived types from analysis. For example, to specify that the rule should not run on any methods within types named `MyType` and their derived types, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Type name only (includes all types with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#), with an optional `T:` prefix.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType</code>	Matches all types named <code>MyType</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType1 MyType2</code>	Matches all types named either <code>MyType1</code> or <code>MyType2</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS.MyType</code>	Matches specific type <code>MyType</code> with given fully qualified name and all of its derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS1.MyType1 M:NS2.MyType2</code>	Matches specific types <code>MyType1</code> and <code>MyType2</code> with the respective fully qualified names, and all of their derived types.

Exclude extension method 'this' parameter

By default, this rule analyzes and flags the `this` parameter for extension methods. You can exclude analysis of the `this` parameter for extension methods by adding the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CA1062.exclude_extension_method_this_parameter = true
```

Null check validation methods

This rule can lead to false positives if your code calls special null-check validation methods in referenced libraries or projects. You can avoid these false positives by specifying the name or signature of null-check

validation methods. The analysis assumes that arguments passed to these methods are non-null after the call. For example, to mark all methods named `Validate` as null-check validation methods, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CA1062.null_check_validation_methods = Validate
```

Allowed method name formats in the option value (separated by `|`):

- Method name only (includes all methods with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#), with an optional `M:` prefix.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CA1062.null_check_validation_methods = Validate</code>	Matches all methods named <code>Validate</code> in the compilation
<code>dotnet_code_quality.CA1062.null_check_validation_methods = Validate1 Validate2</code>	Matches all methods named either <code>Validate1</code> or <code>Validate2</code> in the compilation
<code>dotnet_code_quality.CA1062.null_check_validation_methods = NS.MyType.Validate(ParamType)</code>	Matches specific method <code>Validate</code> with given fully qualified signature
<code>dotnet_code_quality.CA1062.null_check_validation_methods = NS1.MyType1.Validate1(ParamType) NS2.MyType2.Validate2(ParamType)</code>	Matches specific methods <code>Validate1</code> and <code>Validate2</code> with respective fully qualified signature

Example 1

The following example shows a method that violates the rule and a method that satisfies the rule.

```

using System;

namespace DesignLibrary
{
    public class Test
    {
        // This method violates the rule.
        public void DoNotValidate(string input)
        {
            if (input.Length != 0)
            {
                Console.WriteLine(input);
            }
        }

        // This method satisfies the rule.
        public void Validate(string input)
        {
            if (input == null)
            {
                throw new ArgumentNullException(nameof(input));
            }
            if (input.Length != 0)
            {
                Console.WriteLine(input);
            }
        }
    }
}

```

```

Imports System

Namespace DesignLibrary

    Public Class Test

        ' This method violates the rule.
        Sub DoNotValidate(ByVal input As String)

            If input.Length <> 0 Then
                Console.WriteLine(input)
            End If

        End Sub

        ' This method satisfies the rule.
        Sub Validate(ByVal input As String)

            If input Is Nothing Then
                Throw New ArgumentNullException(NameOf(input))
            End If

            If input.Length <> 0 Then
                Console.WriteLine(input)
            End If

        End Sub

    End Class

End Namespace

```

Example 2

Copy constructors that populate fields or properties that are reference objects can also violate rule CA1062. The violation occurs because the copied object that's passed to the copy constructor might be `null` (`Nothing` in Visual Basic). To resolve the violation, use a `static` (`Shared` in Visual Basic) method to check that the copied object is not null.

In the following `Person` class example, the `other` object that is passed to the `Person` copy constructor might be `null`.

```
public class Person
{
    public string Name { get; private set; }
    public int Age { get; private set; }

    public Person(string name, int age)
    {
        Name = name;
        Age = age;
    }

    // Copy constructor CA1062 fires because other is dereferenced
    // without being checked for null
    public Person(Person other)
        : this(other.Name, other.Age)
    {
    }
}
```

Example 3

In the following revised `Person` example, the `other` object that's passed to the copy constructor is first checked for null in the `PassThroughNonNull` method.

```
public class Person
{
    public string Name { get; private set; }
    public int Age { get; private set; }

    public Person(string name, int age)
    {
        Name = name;
        Age = age;
    }

    // Copy constructor
    public Person(Person other)
        : this(PassThroughNonNull(other).Name, other.Age)
    {
    }

    // Null check method
    private static Person PassThroughNonNull(Person person)
    {
        if (person == null)
            throw new ArgumentNullException(nameof(person));
        return person;
    }
}
```

CA1063: Implement IDisposable correctly

9/20/2022 • 3 minutes to read • [Edit Online](#)

	Value
Rule ID	CA1063
Category	Design
Fix is breaking or non-breaking	Non-breaking

Cause

The `System.IDisposable` interface is not implemented correctly. Possible reasons for this include:

- `IDisposable` is reimplemented in the class.
- `Finalize` is overridden again.
- `Dispose()` is overridden.
- The `Dispose()` method is not public, `sealed`, or named `Dispose`.
- `Dispose(bool)` is not protected, virtual, or unsealed.
- In unsealed types, `Dispose()` must call `Dispose(true)`.
- For unsealed types, the `Finalize` implementation does not call either or both `Dispose(bool)` or the base class finalizer.

Violation of any one of these patterns triggers warning CA1063.

Every unsealed type that declares and implements the `IDisposable` interface must provide its own `protected virtual void Dispose(bool)` method. `Dispose()` should call `Dispose(true)`, and the finalizer should call `Dispose(false)`. If you create an unsealed type that declares and implements the `IDisposable` interface, you must define `Dispose(bool)` and call it. For more information, see [Clean up unmanaged resources \(.NET guide\)](#) and [Dispose pattern](#).

By default, this rule only looks at externally visible types, but this is [configurable](#).

Rule description

All `IDisposable` types should implement the [Dispose pattern](#) correctly.

How to fix violations

Examine your code and determine which of the following resolutions will fix this violation:

- Remove `IDisposable` from the list of interfaces that are implemented by your type, and override the base class `Dispose` implementation instead.
- Remove the finalizer from your type, override `Dispose(bool disposing)`, and put the finalization logic in the code path where 'disposing' is false.

- Override `Dispose(bool disposing)`, and put the dispose logic in the code path where 'disposing' is true.
- Make sure that `Dispose()` is declared as public and [sealed](#).
- Rename your dispose method to **Dispose** and make sure that it's declared as public and [sealed](#).
- Make sure that `Dispose(bool)` is declared as protected, virtual, and unsealed.
- Modify `Dispose()` so that it calls `Dispose(true)`, then calls [SuppressFinalize](#) on the current object instance (`this`, or `Me` in Visual Basic), and then returns.
- Modify your finalizer so that it calls `Dispose(false)` and then returns.
- If you create an unsealed type that declares and implements the [IDisposable](#) interface, make sure that the implementation of [IDisposable](#) follows the pattern that is described earlier in this section.

When to suppress warnings

Do not suppress a warning from this rule.

Configure code to analyze

Use the following option to configure which parts of your codebase to run this rule on.

- [Include specific API surfaces](#)

You can configure this option for just this rule, for all rules, or for all rules in this category ([Design](#)). For more information, see [Code quality rule configuration options](#).

Include specific API surfaces

You can configure which parts of your codebase to run this rule on, based on their accessibility. For example, to specify that the rule should run only against the non-public API surface, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXX.api_surface = private, internal
```

Pseudo-code example

The following pseudo-code provides a general example of how `Dispose(bool)` should be implemented in a class that uses managed and native resources.

```
public class Resource : IDisposable
{
    private bool isDisposed;
    private IntPtr nativeResource = Marshal.AllocHGlobal(100);
    private AnotherResource managedResource = new AnotherResource();

    // Dispose() calls Dispose(true)
    public void Dispose()
    {
        Dispose(true);
        GC.SuppressFinalize(this);
    }

    // The bulk of the clean-up code is implemented in Dispose(bool)
    protected virtual void Dispose(bool disposing)
    {
        if (isDisposed) return;

        if (disposing)
        {
            // free managed resources
            managedResource.Dispose();
        }

        // free native resources if there are any.
        if (nativeResource != IntPtr.Zero)
        {
            Marshal.FreeHGlobal(nativeResource);
            nativeResource = IntPtr.Zero;
        }

        isDisposed = true;
    }

    // NOTE: Leave out the finalizer altogether if this class doesn't
    // own unmanaged resources, but leave the other methods
    // exactly as they are.
    ~Resource()
    {
        // Finalizer calls Dispose(false)
        Dispose(false);
    }
}
```

See also

- [Dispose pattern \(framework design guidelines\)](#)
- [Clean up unmanaged resources \(.NET guide\)](#)

CA1064: Exceptions should be public

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA1064
Category	Design
Fix is breaking or non-breaking	Non-breaking

Cause

A non-public exception derives directly from [Exception](#), [SystemException](#), or [ApplicationException](#).

Rule description

An internal exception is only visible inside its own internal scope. After the exception falls outside the internal scope, only the base exception can be used to catch the exception. If the internal exception is inherited from [Exception](#), [SystemException](#), or [ApplicationException](#), the external code will not have sufficient information to know what to do with the exception.

But, if the code has a public exception that later is used as the base for an internal exception, it is reasonable to assume the code further out will be able to do something intelligent with the base exception. The public exception will have more information than what is provided by [Exception](#), [SystemException](#), or [ApplicationException](#).

How to fix violations

Make the exception public, or derive the internal exception from a public exception that is not [Exception](#), [SystemException](#), or [ApplicationException](#).

When to suppress warnings

Suppress a message from this rule if you are sure in all cases that the private exception will be caught within its own internal scope.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1064
// The code that's violating the rule is on this line.
#pragma warning restore CA1064
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1064.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Design.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Example

This rule fires on the first example method, `FirstCustomException` because the exception class derives directly from `Exception` and is internal. The rule does not fire on the `SecondCustomException` class because although the class also derives directly from `Exception`, the class is declared public. The third class also does not fire the rule because it does not derive directly from [System.Exception](#), [System.SystemException](#), or [System.ApplicationException](#).

```
// Violates this rule
[Serializable]
internal class FirstCustomException : Exception
{
    internal FirstCustomException()
    {
    }

    internal FirstCustomException(string message)
        : base(message)
    {
    }

    internal FirstCustomException(string message, Exception innerException)
        : base(message, innerException)
    {
    }

    protected FirstCustomException(SerializationInfo info, StreamingContext context)
        : base(info, context)
    {
    }
}

// Does not violate this rule because
// SecondCustomException is public
[Serializable]
public class SecondCustomException : Exception
{
    public SecondCustomException()
    {
    }

    public SecondCustomException(string message)
        : base(message)
    {
    }

    public SecondCustomException(string message, Exception innerException)
        : base(message, innerException)
    {
    }
}
```

```
protected SecondCustomException(SerializationInfo info, StreamingContext context)
    : base(info, context)
{
}

// Does not violate this rule because
// ThirdCustomException it does not derive directly from
// Exception, SystemException, or ApplicationException
[Serializable]
internal class ThirdCustomException : SecondCustomException
{
    internal ThirdCustomException()
    {

    }

    internal ThirdCustomException(string message)
        : base(message)
    {

    }

    internal ThirdCustomException(string message, Exception innerException)
        : base(message, innerException)
    {

    }
}

protected ThirdCustomException(SerializationInfo info, StreamingContext context)
    : base(info, context)
{
}
}
```

CA1065: Do not raise exceptions in unexpected locations

9/20/2022 • 3 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA1065
Category	Design
Fix is breaking or non-breaking	Non-breaking

Cause

A method that is not expected to throw exceptions throws an exception.

Rule description

Methods that are not expected to throw exceptions can be categorized as follows:

- Property Get Methods
- Event Accessor Methods
- Equals Methods
- GetHashCode Methods
- ToString Methods
- Static Constructors
- Finalizers
- Dispose Methods
- Equality Operators
- Implicit Cast Operators

The following sections discuss these method types.

Property Get Methods

Properties are basically smart fields. Therefore, they should behave like a field as much as possible. Fields don't throw exceptions and neither should properties. If you have a property that throws an exception, consider making it a method.

The following exceptions can be thrown from a property get method:

- [System.InvalidOperationException](#) and all derivatives (including [System.ObjectDisposedException](#))
- [System.NotSupportedException](#) and all derivatives
- [System.ArgumentException](#) (only from indexed get)

- `KeyNotFoundException` (only from indexed get)

Event Accessor Methods

Event accessors should be simple operations that don't throw exceptions. An event should not throw an exception when you try to add or remove an event handler.

The following exceptions can be thrown from an event accessor:

- `System.InvalidOperationException` and all derivatives (including `System.ObjectDisposedException`)
- `System.NotSupportedException` and all derivatives
- `ArgumentException` and derivatives

Equals Methods

The following `Equals` methods should not throw exceptions:

- `System.Object.Equals`
- `Equals`

An `Equals` method should return `true` or `false` instead of throwing an exception. For example, if `Equals` is passed two mismatched types it should just return `false` instead of throwing an `ArgumentException`.

GetHashCode Methods

The following `GetHashCode` methods should usually not throw exceptions:

- `GetHashCode`
- `GetHashCode`

`GetHashCode` should always return a value. Otherwise, you can lose items in the hash table.

The versions of `GetHashCode` that take an argument can throw an `ArgumentException`. However, `Object.GetHashCode` should never throw an exception.

ToString Methods

The debugger uses `System.Object.ToString` to help display information about objects in string format. Therefore, `ToString` should not change the state of an object, and it shouldn't throw exceptions.

Static Constructors

Throwing exceptions from a static constructor causes the type to be unusable in the current application domain. You should have a good reason (such as a security issue) for throwing an exception from a static constructor.

Finalizers

Throwing an exception from a finalizer causes the CLR to fail fast, which tears down the process. Therefore, throwing exceptions in a finalizer should always be avoided.

Dispose Methods

A `System.IDisposable.Dispose` method should not throw an exception. `Dispose` is often called as part of the cleanup logic in a `finally` clause. Therefore, explicitly throwing an exception from `Dispose` forces the user to add exception handling inside the `finally` clause.

The `Dispose(false)` code path should never throw exceptions, because `Dispose` is almost always called from a finalizer.

Equality Operators (`==`, `!=`)

Like `Equals` methods, equality operators should return either `true` or `false`, and should not throw exceptions.

Implicit Cast Operators

Because the user is often unaware that an implicit cast operator has been called, an exception thrown by the implicit cast operator is unexpected. Therefore, no exceptions should be thrown from implicit cast operators.

How to fix violations

For property getters, either change the logic so that it no longer has to throw an exception, or change the property into a method.

For all other method types listed previously, change the logic so that it no longer must throw an exception.

When to suppress warnings

If the violation was caused by an exception declaration instead of a thrown exception, it is safe to suppress a warning from this rule.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1065
// The code that's violating the rule is on this line.
#pragma warning restore CA1065
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1065.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Design.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Related rules

- [CA2219: Do not raise exceptions in exception clauses](#)

See also

- [Design rules](#)

CA1066: Implement IEquatable when overriding Equals

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA1066
Category	Design
Fix is breaking or non-breaking	Non-breaking

Cause

A value type (struct) overrides [Equals](#) method, but does not implement [IEquatable<T>](#).

Rule description

A value type overriding [Equals](#) method indicates that it supports comparing two instances of the type for value equality. Consider implementing the [IEquatable<T>](#) interface to support strongly typed tests for equality. This ensures that callers performing equality checks invoke the strongly typed [System.IEquatable<T>.Equals](#) method and avoid boxing the argument, improving performance. For more information, see [here](#).

Your [System.IEquatable<T>.Equals](#) implementation should return results that are consistent with [Equals](#).

How to fix violations

To fix a violation, implement [IEquatable<T>](#) and update [Equals](#) override to invoke this implemented method. For example, the following two code snippets show a violation of the rule and how to fix it:

```
public struct S
{
    private readonly int _value;
    public S(int f)
    {
        _value = f;
    }

    public override int GetHashCode()
        => _value.GetHashCode();

    public override bool Equals(object other)
        => other is S otherS && _value == otherS._value;
}
```

```
using System;

public struct S : IEquatable<S>
{
    private readonly int _value;
    public S(int f)
    {
        _value = f;
    }

    public override int GetHashCode()
        => _value.GetHashCode();

    public override bool Equals(object other)
        => other is S otherS && Equals(otherS);

    public bool Equals(S other)
        => _value == other._value;
}
```

When to suppress warnings

It is safe to suppress violations from this rule if the design and performance benefits from implementing the interface are not critical.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1066
// The code that's violating the rule is on this line.
#pragma warning restore CA1066
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1066.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Design.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Related rules

- [CA1067: Override Equals when implementing IEquatable](#)

See also

- [Design rules](#)

CA1067: Override Equals when implementing IEquatable

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA1067
Category	Design
Fix is breaking or non-breaking	Non-breaking

Cause

A type implements [IEquatable<T>](#), but does not override [Equals](#) method.

Rule description

A type implementing [IEquatable<T>](#) interface indicates that it supports comparing two instances of the type for equality. You should also override the base class implementations of [Equals](#) and [GetHashCode\(\)](#) methods so that their behavior is consistent with that of the [System.IEquatable<T>.Equals](#) implementation. See [here](#) for more details.

Your [Equals](#) implementation should return results that are consistent with [System.IEquatable<T>.Equals](#) implementation.

How to fix violations

To fix a violation, override [Equals](#) and implement it by invoking the [System.IEquatable<T>.Equals](#) implementation. For example, the following two code snippets show a violation of the rule and how to fix it:

```
using System;

public struct S : IEquatable<S>
{
    private readonly int _value;
    public S(int f)
    {
        _value = f;
    }

    public bool Equals(S other)
        => _value == other._value;
}
```

```
using System;

public struct S : IEquatable<S>
{
    private readonly int _value;
    public S(int f)
    {
        _value = f;
    }

    public bool Equals(S other)
    => _value == other._value;

    public override bool Equals(object obj)
    => obj is S objs && Equals(objs);

    public override int GetHashCode()
    => _value.GetHashCode();
}
```

When to suppress warnings

Do not suppress violations of this rule.

Related rules

- [CA1066: Implement IEquatable when overriding Equals](#)

See also

- [Design rules](#)

CA1068: CancellationToken parameters must come last

9/20/2022 • 4 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA1068
Category	Design
Fix is breaking or non-breaking	Breaking

Cause

A method has a [CancellationToken](#) parameter that is not the last parameter.

By default, this rule analyzes the entire codebase, but this is [configurable](#).

Rule description

Methods that perform long running operations or asynchronous operations and are cancelable normally take a cancellation token parameter. Each cancellation token has a [CancellationTokenSource](#) that creates the token and uses it for cancelable computations. It is common practice to have a long chain of method calls that pass around the cancellation token from the callers to the callees. Hence, a large number of methods that take part in a cancelable computation end up having a cancellation token parameter. However, the cancellation token itself is not usually relevant to the core functionality of a majority of these methods. It's considered a good API design practice to have such parameters be the last parameter in the list.

Special cases

Rule CA1068 does not fire in the following special cases:

- Method has one or more [optional](#) parameters ([Optional](#) in Visual Basic) following a non-optional cancellation token parameter. The compiler requires that all optional parameters are defined after all the non-optional parameters.
- Method has one or more [ref](#) or [out](#) parameters ([ByRef](#) in Visual Basic) following a cancellation token parameter. It is common practice to have [ref](#) or [out](#) parameters be at the end of the list, because they usually indicate output values for the method.

How to fix violations

Change the method signature to move the cancellation token parameter to the end of the list. For example, the following two code snippets show a violation of the rule and how to fix it:

```
// Violates CA1068
public void LongRunningOperation(CancellationToken token, string usefulParameter)
{
    ...
}
```

```
// Does not violate CA1068
public void LongRunningOperation(string usefulParameter, CancellationToken token)
{
    ...
}
```

When to suppress warnings

If the method is an externally visible public API that is already part of a shipped library, then it is safe to suppress a warning from this rule to avoid a breaking change for the library consumers.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1068
// The code that's violating the rule is on this line.
#pragma warning restore CA1068
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1068.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Design.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

Use the following options to configure which parts of your codebase to run this rule on.

- [Include specific API surfaces](#)
- [Exclude specific symbols](#)
- [Exclude specific types and their derived types](#)

You can configure these options for just this rule, for all rules, or for all rules in this category ([Design](#)). For more information, see [Code quality rule configuration options](#).

Include specific API surfaces

You can configure which parts of your codebase to run this rule on, based on their accessibility. For example, to specify that the rule should run only against the non-public API surface, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.api_surface = private, internal
```

Exclude specific symbols

You can exclude specific symbols, such as types and methods, from analysis. For example, to specify that the rule should not run on any code within types named `MyType`, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Symbol name only (includes all symbols with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#). Each symbol name requires a symbol-kind prefix, such as `M:` for methods, `T:` for types, and `N:` for namespaces.
- `.ctor` for constructors and `.cctor` for static constructors.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType</code>	Matches all symbols named <code>MyType</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType1 MyType2</code>	Matches all symbols named either <code>MyType1</code> or <code>MyType2</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS.MyType.MyMethod(ParamType)</code>	Matches specific method <code>MyMethod</code> with the specified fully qualified signature.
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS1.MyType1.MyMethod1(ParamType) M:NS2.MyType2.MyMethod2(ParamType)</code>	Matches specific methods <code>MyMethod1</code> and <code>MyMethod2</code> with the respective fully qualified signatures.

Exclude specific types and their derived types

You can exclude specific types and their derived types from analysis. For example, to specify that the rule should not run on any methods within types named `MyType` and their derived types, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Type name only (includes all types with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#), with an optional `T:` prefix.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType</code>	Matches all types named <code>MyType</code> and all of their derived types.

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType1 MyType2</code>	Matches all types named either <code>MyType1</code> or <code>MyType2</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS.MyType</code>	Matches specific type <code>MyType</code> with given fully qualified name and all of its derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS1.MyType1 M:NS2.MyType2</code>	Matches specific types <code>MyType1</code> and <code>MyType2</code> with the respective fully qualified names, and all of their derived types.

Related rules

- [CA1021: Avoid out parameters](#)

See also

- [Recommended patterns for CancellationToken](#)

CA1069: Enums should not have duplicate values

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA1069
Category	Design
Fix is breaking or non-breaking	Breaking

Cause

An [enumeration](#) has multiple members which are explicitly assigned the same constant value.

Rule description

Every enum member should either have a unique constant value or be explicitly assigned with a prior member in the enum to indicate explicit intent of sharing value. For example:

```
enum E
{
    Field1 = 1,
    AnotherNameForField1 = Field1,    // This is fine
    Field2 = 2,
    Field3 = 2,    // CA1069: This is not fine. Either assign a different constant value or 'Field2' to
    indicate explicit intent of sharing value.
}
```

This rule helps in catching functional bugs introduced from the following scenarios:

- Accidental typing mistakes, where the user accidentally typed the same constant value for multiple members.
- Copy paste mistakes, where the user copied an existing member definition, then renamed the member but forgot to change the value.
- Merge resolution from multiple branches, where a new member was added with a different name but the same value in different branches.

How to fix violations

To fix a violation, either assign a new unique constant value or assign with a prior member in the enum to indicate explicit intent of sharing the same value. For example, the following code snippet shows a violation of the rule and couple of ways to fix the violation:

```
enum E
{
    Field1 = 1,
    AnotherNameForField1 = Field1,    // This is fine
    Field2 = 2,
    Field3 = 2,    // CA1069: This is not fine. Either assign a different constant value or 'Field2' to
    indicate explicit intent of sharing value.
}
```

```
enum E
{
    Field1 = 1,
    AnotherNameForField1 = Field1,    // This is fine
    Field2 = 2,
    Field3 = 3,    // This is now fine
}
```

```
enum E
{
    Field1 = 1,
    AnotherNameForField1 = Field1,    // This is fine
    Field2 = 2,
    Field3 = Field2,    // This is also fine
}
```

When to suppress warnings

Do not suppress violations of this rule.

See also

- [Design rules](#)

CA1070: Do not declare event fields as virtual

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA1070
Category	Design
Fix is breaking or non-breaking	Breaking

Cause

A [field-like event](#) was declared as virtual.

By default, this rule only looks at externally visible types, but this is [configurable](#).

Rule description

Follow these [.NET design guidelines](#) to raise base class events in derived classes. Do not declare virtual events in a base class. Overridden events in a derived class have undefined behavior. The C# compiler does not handle this correctly and it is unpredictable whether a subscriber to the derived event will actually be subscribing to the base class event.

```
using System;
public class C
{
    // CA1070: Event 'ThresholdReached' should not be declared virtual.
    public virtual event EventHandler ThresholdReached;
}
```

How to fix violations

Follow these [.NET design guidelines](#) and avoid virtual field-like events.

When to suppress warnings

If the event is an externally visible public API that is already part of a shipped library, then it is safe to suppress a warning from this rule to avoid a breaking change for the library consumers.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1070
// The code that's violating the rule is on this line.
#pragma warning restore CA1070
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1070.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Design.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

Use the following option to configure which parts of your codebase to run this rule on.

- [Include specific API surfaces](#)

You can configure this option for just this rule, for all rules, or for all rules in this category ([Design](#)). For more information, see [Code quality rule configuration options](#).

Include specific API surfaces

You can configure which parts of your codebase to run this rule on, based on their accessibility. For example, to specify that the rule should run only against the non-public API surface, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.api_surface = private, internal
```

See also

- [Design rules](#)

Documentation rules

9/20/2022 • 2 minutes to read • [Edit Online](#)

Documentation rules support writing well-documented libraries through the correct use of [XML documentation comments](#) for externally visible APIs.

In this section

RULE	DESCRIPTION
CA1200: Avoid using cref tags with a prefix	The <code>cref</code> attribute in an XML documentation tag means "code reference". It specifies that the inner text of the tag is a code element, such as a type, method, or property. Avoid using <code>cref</code> tags with prefixes, because it prevents the compiler from verifying references. It also prevents the Visual Studio integrated development environment (IDE) from finding and updating these symbol references during refactorings.

CA1200: Avoid using cref tags with a prefix

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA1200
Category	Documentation
Fix is breaking or non-breaking	Non-breaking

Cause

The `cref` tag in an XML documentation comment uses a [prefix](#).

Rule description

The `cref` attribute in an XML documentation tag means "code reference". It specifies that the inner text of the tag is a code element, such as a type, method, or property. Avoid using `cref` tags with prefixes, because it prevents the compiler from verifying references. It also prevents the Visual Studio integrated development environment (IDE) from finding and updating these symbol references during refactorings. It is recommended that you use the full syntax without prefixes to reference symbol names in cref tags.

How to fix violations

To fix a violation of this rule, remove the prefix from the `cref` tag. For example, the following two code snippets show a violation of the rule and how to fix it:

```
// Violates CA1200
/// <summary>
/// Type <see cref="T:C" /> contains method <see cref="F:C.F" />
/// </summary>
class C
{
    public void F() { }
}
```

```
// Does not violate CA1200
/// <summary>
/// Type <see cref="C" /> contains method <see cref="F" />
/// </summary>
class C
{
    public void F() { }
}
```

When to suppress warnings

It's safe to suppress this warning if the code reference must use a prefix because the referenced type is not findable by the compiler. For example, if a code reference references a special attribute in the full framework, but

the file compiles against the portable framework, you can suppress this warning.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1200
// The code that's violating the rule is on this line.
#pragma warning restore CA1200
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1200.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Documentation.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Documenting your code with XML comments](#)

Globalization rules

9/20/2022 • 2 minutes to read • [Edit Online](#)

Globalization rules support world-ready libraries and applications.

In this section

RULE	DESCRIPTION
CA1303: Do not pass literals as localized parameters	An externally visible method passes a string literal as a parameter to a .NET constructor or method, and that string should be localizable.
CA1304: Specify CultureInfo	A method or constructor calls a member that has an overload that accepts a System.Globalization.CultureInfo parameter, and the method or constructor does not call the overload that takes the CultureInfo parameter. When a CultureInfo or System.IFormatProvider object is not supplied, the default value that is supplied by the overloaded member might not have the effect that you want in all locales.
CA1305: Specify IFormatProvider	A method or constructor calls one or more members that have overloads that accept a System.IFormatProvider parameter, and the method or constructor does not call the overload that takes the IFormatProvider parameter. When a System.Globalization.CultureInfo or IFormatProvider object is not supplied, the default value that is supplied by the overloaded member might not have the effect that you want in all locales.
CA1307: Specify StringComparison for clarity	A string comparison operation uses a method overload that does not set a StringComparison parameter.
CA1308: Normalize strings to uppercase	Strings should be normalized to uppercase. A small group of characters cannot make a round trip when they are converted to lowercase.
CA1309: Use ordinal StringComparison	A string comparison operation that is nonlinguistic does not set the StringComparison parameter to either Ordinal or OrdinalIgnoreCase. By explicitly setting the parameter to either StringComparison.Ordinal or StringComparison.OrdinalIgnoreCase, your code often gains speed, becomes more correct, and becomes more reliable.
CA1310: Specify StringComparison for correctness	A string comparison operation uses a method overload that does not set a StringComparison parameter and uses culture-specific string comparison by default.
CA2101: Specify marshalling for P/Invoke string arguments	A platform invoke member allows for partially trusted callers, has a string parameter, and does not explicitly marshal the string. This can cause a potential security vulnerability.

CA1303: Do not pass literals as localized parameters

9/20/2022 • 4 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA1303
Category	Globalization
Fix is breaking or non-breaking	Non-breaking

Cause

A method passes a string literal as a parameter to a .NET constructor or method and that string should be localizable.

This warning is raised when a literal string is passed as a value to a parameter or property and one or more of the following situations is true:

- The [LocalizableAttribute](#) attribute of the parameter or property is set to `true`.
- The literal string is passed to the `string value` or `string format` parameter of a [Console.WriteLine](#) or [Console.WriteLine](#) method overload.
- Rule CA1303 is [configured to use the naming heuristic](#), and a parameter or property name contains the phrase `Text`, `Message`, or `Caption`.

By default, this rule analyzes the entire codebase, but this is [configurable](#).

Rule description

String literals that are embedded in source code are difficult to localize.

How to fix violations

To fix a violation of this rule, replace the string literal with a string retrieved through an instance of the [ResourceManager](#) class.

For methods that don't require localized strings, you can eliminate unnecessary CA1303 warnings in the following ways:

- If the [naming heuristic option](#) is enabled, rename the parameter or property.
- Remove the [LocalizableAttribute](#) attribute on the parameter or property, or set it to `false` (`[Localizable(false)]`).

When to suppress warnings

It's safe to suppress a warning from this rule if either of the following statements applies:

- The code library will not be localized.
- The string is not exposed to the end user or a developer using the code library.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1303
// The code that's violating the rule is on this line.
#pragma warning restore CA1303
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.cs,vb]
dotnet_diagnostic.CA1303.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.cs,vb]
dotnet_analyzer_diagnostic.category-Globalization.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

Use the following options to configure which parts of your codebase to run this rule on.

- [Exclude specific symbols](#)
- [Exclude specific types and their derived types](#)
- [Use naming heuristic](#)

You can configure these options for just this rule, for all rules, or for all rules in this category ([Globalization](#)). For more information, see [Code quality rule configuration options](#).

Exclude specific symbols

You can exclude specific symbols, such as types and methods, from analysis. For example, to specify that the rule should not run on any code within types named `MyType`, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Symbol name only (includes all symbols with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#). Each symbol name requires a symbol-kind prefix, such as `M:` for methods, `T:` for types, and `N:` for namespaces.
- `.ctor` for constructors and `.cctor` for static constructors.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType</code>	Matches all symbols named <code>MyType</code> .

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXX.excluded_symbol_names = MyType1 MyType2</code>	Matches all symbols named either <code>MyType1</code> or <code>MyType2</code> .
<code>dotnet_code_quality.CAXXX.excluded_symbol_names = M:NS.MyType.MyMethod(ParamType)</code>	Matches specific method <code>MyMethod</code> with the specified fully qualified signature.
<code>dotnet_code_quality.CAXXX.excluded_symbol_names = M:NS1.MyType1.MyMethod1(ParamType) M:NS2.MyType2.MyMethod2(ParamType)</code>	Matches specific methods <code>MyMethod1</code> and <code>MyMethod2</code> with the respective fully qualified signatures.

Exclude specific types and their derived types

You can exclude specific types and their derived types from analysis. For example, to specify that the rule should not run on any methods within types named `MyType` and their derived types, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXX.excluded_type_names_with_derived_types = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Type name only (includes all types with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#), with an optional `T:` prefix.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXX.excluded_type_names_with_derived_types = MyType</code>	Matches all types named <code>MyType</code> and all of their derived types.
<code>dotnet_code_quality.CAXXX.excluded_type_names_with_derived_types = MyType1 MyType2</code>	Matches all types named either <code>MyType1</code> or <code>MyType2</code> and all of their derived types.
<code>dotnet_code_quality.CAXXX.excluded_type_names_with_derived_types = M:NS.MyType</code>	Matches specific type <code>MyType</code> with given fully qualified name and all of its derived types.
<code>dotnet_code_quality.CAXXX.excluded_type_names_with_derived_types = M:NS1.MyType1 M:NS2.MyType2</code>	Matches specific types <code>MyType1</code> and <code>MyType2</code> with the respective fully qualified names, and all of their derived types.

Use naming heuristic

You can configure whether parameters or property names containing `Text`, `Message`, or `Caption` will trigger this rule.

```
dotnet_code_quality.CA1303.use_naming_heuristic = true
```

Example

The following example shows a method that writes to the console when either of its two arguments are out of range. For the `hour` argument check, a literal string is passed to `Console.WriteLine`, which violates this rule. For the `minute` argument check, a string that's retrieved through a [ResourceManager](#) is passed to `Console.WriteLine`, which satisfies the rule.

```

<Assembly: System.Resources.NeutralResourcesLanguageAttribute("en-US")>
Namespace GlobalizationLibrary

Public Class DoNotPassLiterals

    Dim stringManager As System.Resources.ResourceManager

    Sub New()
        stringManager = New System.Resources.ResourceManager(
            "en-US", System.Reflection.Assembly.GetExecutingAssembly())
    End Sub

    Sub TimeMethod(hour As Integer, minute As Integer)

        If (hour < 0 Or hour > 23) Then
            'CA1303 fires because a literal string
            'is passed as the 'value' parameter.
            Console.WriteLine("The valid range is 0 - 23.")
        End If

        If (minute < 0 Or minute > 59) Then
            Console.WriteLine(
                stringManager.GetString("minuteOutOfRangeMessage",
                    System.Globalization.CultureInfo.CurrentCulture))
        End If

    End Sub

End Class

End Namespace

```

```

public class DoNotPassLiterals
{
    ResourceManager stringManager;
    public DoNotPassLiterals()
    {
        stringManager = new ResourceManager("en-US", Assembly.GetExecutingAssembly());
    }

    public void TimeMethod(int hour, int minute)
    {
        if (hour < 0 || hour > 23)
        {
            // CA1303 fires because a literal string
            // is passed as the 'value' parameter.
            Console.WriteLine("The valid range is 0 - 23.");
        }

        if (minute < 0 || minute > 59)
        {
            Console.WriteLine(stringManager.GetString(
                "minuteOutOfRangeMessage", CultureInfo.CurrentCulture));
        }
    }
}

```

See also

- [Resources in .NET apps](#)
- [Community request for change of behavior](#)

CA1304: Specify CultureInfo

9/20/2022 • 4 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA1304
Category	Globalization
Fix is breaking or non-breaking	Non-breaking

Cause

A method or constructor calls a member that has an overload that accepts a `System.Globalization.CultureInfo` parameter, and the method or constructor does not call the overload that takes the `CultureInfo` parameter. This rule ignores calls to the following methods:

- `Activator.CreateInstance`
- `ResourceManager.GetObject`
- `ResourceManager.GetString`

You can also [configure](#) more symbols to be excluded by this rule.

Rule description

When a `CultureInfo` or `System.IFormatProvider` object is not supplied, the default value that is supplied by the overloaded member might not have the effect that you want in all locales. Also, .NET members choose default culture and formatting based on assumptions that might not be correct for your code. To ensure the code works as expected for your scenarios, you should supply culture-specific information according to the following guidelines:

- If the value will be displayed to the user, use the current culture. See `CultureInfo.CurrentCulture`.
- If the value will be stored and accessed by software, that is, persisted to a file or database, use the invariant culture. See `CultureInfo.InvariantCulture`.
- If you do not know the destination of the value, have the data consumer or provider specify the culture.

Even if the default behavior of the overloaded member is appropriate for your needs, it is better to explicitly call the culture-specific overload so that your code is self-documenting and more easily maintained.

NOTE

`CultureInfo.CurrentCulture` is used only to retrieve localized resources by using an instance of the `System.Resources.ResourceManager` class.

How to fix violations

To fix a violation of this rule, use the overload that takes a `CultureInfo` argument.

When to suppress warnings

It is safe to suppress a warning from this rule when it is certain that the default culture is the correct choice, and where code maintainability is not an important development priority.

SUPPRESS A WARNING

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1304
// The code that's violating the rule is on this line.
#pragma warning restore CA1304
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1304.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Globalization.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

CONFIGURE CODE TO ANALYZE

Use the following options to configure which parts of your codebase to run this rule on.

- [Exclude specific symbols](#)
- [Exclude specific types and their derived types](#)

You can configure these options for just this rule, for all rules, or for all rules in this category ([Globalization](#)). For more information, see [Code quality rule configuration options](#).

Exclude specific symbols

You can exclude specific symbols, such as types and methods, from analysis. For example, to specify that the rule should not run on any code within types named `MyType`, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXX.excluded_symbol_names = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Symbol name only (includes all symbols with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#). Each symbol name requires a symbol-kind prefix, such as `M:` for methods, `T:` for types, and `N:` for namespaces.
- `.ctor` for constructors and `.cctor` for static constructors.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType</code>	Matches all symbols named <code>MyType</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType1 MyType2</code>	Matches all symbols named either <code>MyType1</code> or <code>MyType2</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS.MyType.MyMethod(ParamType)</code>	Matches specific method <code>MyMethod</code> with the specified fully qualified signature.
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS1.MyType1.MyMethod1(ParamType) M:NS2.MyType2.MyMethod2(ParamType)</code>	Matches specific methods <code>MyMethod1</code> and <code>MyMethod2</code> with the respective fully qualified signatures.

Exclude specific types and their derived types

You can exclude specific types and their derived types from analysis. For example, to specify that the rule should not run on any methods within types named `MyType` and their derived types, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Type name only (includes all types with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#), with an optional `T:` prefix.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType</code>	Matches all types named <code>MyType</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType1 MyType2</code>	Matches all types named either <code>MyType1</code> or <code>MyType2</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS.MyType</code>	Matches specific type <code>MyType</code> with given fully qualified name and all of its derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS1.MyType1 M:NS2.MyType2</code>	Matches specific types <code>MyType1</code> and <code>MyType2</code> with the respective fully qualified names, and all of their derived types.

Example showing how to fix violations

In the following example, `BadMethod` causes two violations of this rule. `GoodMethod` corrects the first violation by passing the invariant culture to [String.Compare](#), and corrects the second violation by passing the current culture to [String.ToLower](#) because `string3` is displayed to the user.

```

public class CultureInfoTest
{
    public void BadMethod(String string1, String string2, String string3)
    {
        if (string.Compare(string1, string2, false) == 0)
        {
            Console.WriteLine(string3.ToLower());
        }
    }

    public void GoodMethod(String string1, String string2, String string3)
    {
        if (string.Compare(string1, string2, false,
                           CultureInfo.InvariantCulture) == 0)
        {
            Console.WriteLine(string3.ToLower(CultureInfo.CurrentCulture));
        }
    }
}

```

Example showing formatted output

The following example shows the effect of current culture on the default [IFormatProvider](#) that is selected by the [DateTime](#) type.

```

public class IFormatProviderTest
{
    public static void Main1304()
    {
        string dt = "6/4/1900 12:15:12";

        // The default behavior of DateTime.Parse is to use
        // the current culture.

        // Violates rule: SpecifyIFormatProvider.
        DateTime myDateTime = DateTime.Parse(dt);
        Console.WriteLine(myDateTime);

        // Change the current culture to the French culture,
        // and parsing the same string yields a different value.

        Thread.CurrentThread.CurrentCulture = new CultureInfo("Fr-fr", true);
        myDateTime = DateTime.Parse(dt);

        Console.WriteLine(myDateTime);
    }
}

```

This example produces the following output:

```

6/4/1900 12:15:12 PM
06/04/1900 12:15:12

```

Related rules

- [CA1305: Specify IFormatProvider](#)

See also

- Use the `CultureInfo` class

CA1305: Specify IFormatProvider

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA1305
Category	Globalization
Fix is breaking or non-breaking	Non-breaking

Cause

A call is made to a method that has an overload that accepts a [System.IFormatProvider](#) parameter, and that overload isn't called.

This rule ignores calls to .NET methods that are documented as ignoring the [IFormatProvider](#) parameter. The rule also ignores the following methods:

- [Activator.CreateInstance](#)
- [ResourceManager.GetObject](#)
- [ResourceManager.GetString](#)
- [Boolean.ToString](#)
- [Char.ToString](#)
- [Guid.ToString](#)

Rule description

When a [System.Globalization.CultureInfo](#) or [IFormatProvider](#) object is not supplied, the default value that is supplied by the overloaded member might not have the effect that you want in all locales. Also, .NET members choose default culture and formatting based on assumptions that might not be correct for your code. To make sure that the code works as expected for your scenarios, you should supply culture-specific information according to the following guidelines:

- If the value will be displayed to the user, use the current culture. See [CultureInfo.CurrentCulture](#).
- If the value will be stored and accessed by software (persisted to a file or database), use the invariant culture. See [CultureInfo.InvariantCulture](#).
- If you do not know the destination of the value, have the data consumer or provider specify the culture.

Even if the default behavior of the overloaded member is appropriate for your needs, it is better to explicitly call the culture-specific overload so that your code is self-documenting and more easily maintained.

How to fix violations

To fix a violation of this rule, use the overload that takes an [IFormatProvider](#) argument. Or, use a [C# interpolated string](#) and pass it to the [FormattableString.Invariant](#) method.

When to suppress warnings

It is safe to suppress a warning from this rule when it is certain that the default format is the correct choice, and where code maintainability is not an important development priority.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1305
// The code that's violating the rule is on this line.
#pragma warning restore CA1305
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1305.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Globalization.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Example

In the following code, the `example1` string violates rule CA1305. The `example2` string satisfies rule CA1305 by passing `CultureInfo.CurrentCulture`, which implements `IFormatProvider`, to `String.Format(IFormatProvider, String, Object)`. The `example3` string satisfies rule CA1305 by passing an interpolated string to `Invariant`.

```
string name = "Georgette";

// Violates CA1305
string example1 = String.Format("Hello {0}", name);

// Satisfies CA1305
string example2 = String.Format(CultureInfo.CurrentCulture, "Hello {0}", name);

// Satisfies CA1305
string example3 = FormattableString.Invariant($"Hello {name}");
```

Related rules

- [CA1304: Specify CultureInfo](#)

See also

- [Use the `CultureInfo` class](#)

CA1307: Specify StringComparison for clarity

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA1307
Category	Globalization
Fix is breaking or non-breaking	Non-breaking

Cause

A string comparison operation uses a method overload that does not set a `StringComparison` parameter.

Rule description

Many string compare operations provide an overload that accepts a `StringComparison` enumeration value as a parameter.

Whenever an overload exists that takes a `StringComparison` parameter, it should be used instead of an overload that does not take this parameter. By explicitly setting this parameter, your code is often made clearer and easier to maintain. For more information, see [Specifying string comparisons explicitly](#).

NOTE

This rule does not consider the default `StringComparison` value used by the comparison method. Hence, it can be potentially noisy for methods that use the `Ordinal` string comparison by default and the user intended to use this default compare mode. If you only want to see violations only for known string methods that use culture-specific string comparison by default, please use [CA1310: Specify StringComparison for correctness](#) instead.

How to fix violations

To fix a violation of this rule, change string comparison methods to overloads that accept the `StringComparison` enumeration as a parameter. For example, change `str1.IndexOf(ch1)` to
`str1.IndexOf(ch1, StringComparison.Ordinal)`.

When to suppress warnings

It is safe to suppress a warning from this rule when clarity of intent is not required. For example, test code or non-localizable code may not require it.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1307
// The code that's violating the rule is on this line.
#pragma warning restore CA1307
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1307.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Globalization.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Best Practices for Using Strings in .NET](#)
- [Globalization rules](#)
- [CA1310: Specify StringComparison for correctness](#)
- [CA1309: Use ordinal StringComparison](#)

CA1308: Normalize strings to uppercase

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA1308
Category	Globalization
Fix is breaking or non-breaking	Non-breaking

Cause

An operation normalizes a string to lowercase.

Rule description

Strings should be normalized to uppercase. A small group of characters, when they are converted to lowercase, cannot make a round trip. To make a round trip means to convert the characters from one locale to another locale that represents character data differently, and then to accurately retrieve the original characters from the converted characters.

How to fix violations

Change operations that convert strings to lowercase so that the strings are converted to uppercase instead. For example, change `String.ToLower(CultureInfo.InvariantCulture)` to `String.ToUpper(CultureInfo.InvariantCulture)`.

When to suppress warnings

It's safe to suppress a warning when you're not making security decisions based on the result of the normalization (for example, when you're displaying the result in the UI).

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1308
// The code that's violating the rule is on this line.
#pragma warning restore CA1308
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1308.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]\ndotnet_analyzer_diagnostic.category-Globalization.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Best practices for comparing strings](#)
- [Globalization rules](#)

CA1309: Use ordinal StringComparison

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA1309
Category	Globalization
Fix is breaking or non-breaking	Non-breaking

Cause

A string comparison operation that is nonlinguistic does not set the `StringComparison` parameter to either `Ordinal` or `OrdinalIgnoreCase`.

Rule description

Many string operations, most importantly the `System.String.Compare` and `System.String.Equals` methods, now provide an overload that accepts a `System.StringComparison` enumeration value as a parameter.

When you specify either `StringComparison.Ordinal` or `StringComparison.OrdinalIgnoreCase`, the string comparison is non-linguistic. That is, the features that are specific to the natural language are ignored when comparison decisions are made. Ignoring natural language features means the decisions are based on simple byte comparisons and not on casing or equivalence tables that are parameterized by culture. As a result, by explicitly setting the parameter to either the `StringComparison.Ordinal` or `StringComparison.OrdinalIgnoreCase`, your code often gains speed, increases correctness, and becomes more reliable.

How to fix violations

To fix a violation of this rule, change the string comparison method to an overload that accepts the `System.StringComparison` enumeration as a parameter, and specify either `Ordinal` or `OrdinalIgnoreCase`. For example, change `String.Compare(str1, str2)` to `String.Compare(str1, str2, StringComparison.Ordinal)`.

When to suppress warnings

It is safe to suppress a warning from this rule when the library or application is intended for a limited local audience, or when the semantics of the current culture should be used.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1309
// The code that's violating the rule is on this line.
#pragma warning restore CA1309
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1309.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Globalization.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Globalization rules](#)
- [CA1307: Specify StringComparison](#)

CA1310: Specify StringComparison for correctness

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA1310
Category	Globalization
Fix is breaking or non-breaking	Non-breaking

Cause

A string comparison operation uses a method overload that does not set a [StringComparison](#) parameter and uses culture-specific string comparison by default. Hence, its behavior will vary based on the current user's locale settings.

Rule description

A string comparison method that uses culture-specific string comparison by default can have potentially unintentional runtime behavior that does not match user intent. It is recommended that you use the overload with the [StringComparison](#) parameter for correctness and clarity of intent.

This rule flags string comparison methods that use the culture-specific [StringComparison](#) value by default. For more information, see [String comparisons that use the current culture](#).

NOTE

If you want to see violations for all string comparison methods, regardless of the default string comparison used by the method, please use [CA1307: Specify StringComparison for clarity](#) instead.

How to fix violations

To fix a violation of this rule, change string comparison methods to overloads that accept the [StringComparison](#) enumeration as a parameter. For example, change `String.Compare(str1, str2)` to
`String.Compare(str1, str2, StringComparison.Ordinal)`.

When to suppress warnings

It is safe to suppress a warning from this rule when the library or application is not intended to be localized.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1310
// The code that's violating the rule is on this line.
#pragma warning restore CA1310
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1310.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Globalization.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Best Practices for Using Strings in .NET](#)
- [Globalization rules](#)
- [CA1307: Specify StringComparison for clarity](#)
- [CA1309: Use ordinal StringComparison](#)

CA2101: Specify marshalling for P/Invoke string arguments

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA2101
Category	Globalization
Fix is breaking or non-breaking	Non-breaking

Cause

A platform invoke member allows for partially trusted callers, has a string parameter, and does not explicitly marshal the string.

Rule description

When you convert from Unicode to ANSI, it is possible that not all Unicode characters can be represented in a specific ANSI code page. *Best-fit mapping* tries to solve this problem by substituting a character for the character that cannot be represented. The use of this feature can cause a potential security vulnerability because you cannot control the character that is chosen. For example, malicious code could intentionally create a Unicode string that contains characters that are not found in a particular code page, which are converted to file system special characters such as '..' or '/'. Note also that security checks for special characters frequently occur before the string is converted to ANSI.

Best-fit mapping is the default for the unmanaged conversion, WChar to MByte. Unless you explicitly disable best-fit mapping, your code might contain an exploitable security vulnerability because of this issue.

Caution

[Code Access Security](#) (CAS) should not be considered a security boundary.

How to fix violations

To fix a violation of this rule, explicitly marshal string data types.

When to suppress warnings

Do not suppress a warning from this rule.

Example

The following example shows a method that violates this rule, and then shows how to fix the violation.

```
class NativeMethods
{
    // Violates rule: SpecifyMarshalingForPInvokeStringArguments.
    [DllImport("advapi32.dll", CharSet = CharSet.Auto)]
    internal static extern int RegCreateKey(IntPtr key, String subKey, out IntPtr result);

    // Satisfies rule: SpecifyMarshalingForPInvokeStringArguments.
    [DllImport("advapi32.dll", CharSet = CharSet.Unicode)]
    internal static extern int RegCreateKey2(IntPtr key, String subKey, out IntPtr result);
}
```

Portability and interoperability rules

9/20/2022 • 2 minutes to read • [Edit Online](#)

Portability rules support portability across different platforms. Interoperability rules support interaction with COM clients.

In this section

RULE	DESCRIPTION
CA1401: P/Invokes should not be visible	A public or protected method in a public type has the <code>System.Runtime.InteropServices.DllImportAttribute</code> attribute (also implemented by the <code>Declare</code> keyword in Visual Basic). Such methods should not be exposed.
CA1416: Validate platform compatibility	Using platform-dependent APIs on a component makes the code no longer work across all platforms.
CA1417: Do not use <code>outAttribute</code> on string parameters for P/Invokes	String parameters passed by value with the <code>outAttribute</code> can destabilize the runtime if the string is an interned string.
CA1418: Use valid platform string	Platform compatibility analyzer requires a valid platform name and version.
CA1419: Provide a parameterless constructor that is as visible as the containing type for concrete types derived from 'System.Runtime.InteropServices.SafeHandle'	Providing a parameterless constructor that is as visible as the containing type for a type derived from <code>System.Runtime.InteropServices.SafeHandle</code> enables better performance and usage with source-generated interop solutions.

CA1401: P/Invokes should not be visible

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA1401
Category	Interoperability
Fix is breaking or non-breaking	Breaking

Cause

A public or protected method in a public type has the [System.Runtime.InteropServices.DllImportAttribute](#) attribute (also implemented by the `Declare` keyword in Visual Basic).

Rule description

Methods that are marked with the [DllImportAttribute](#) attribute (or methods that are defined by using the `Declare` keyword in Visual Basic) use Platform Invocation Services to access unmanaged code. Such methods should not be exposed. By keeping these methods private or internal, you make sure that your library cannot be used to breach security by allowing callers access to unmanaged APIs that they could not call otherwise.

How to fix violations

To fix a violation of this rule, change the access level of the method.

When to suppress warnings

Do not suppress a warning from this rule.

Example

The following example declares a method that violates this rule.

```
// Violates rule: PInvokesShouldNotBeVisible.
public class NativeMethods
{
    [DllImport("kernel32.dll", CharSet = CharSet.Unicode)]
    public static extern bool RemoveDirectory(string name);
}
```

```
Imports System

Namespace ca1401

    ' Violates rule: PInvokesShouldNotBeVisible.
    Public Class NativeMethods
        Public Declare Function RemoveDirectory Lib "kernel32" (
            ByVal Name As String) As Boolean
    End Class

End Namespace
```

CA1416: Validate platform compatibility

9/20/2022 • 8 minutes to read • [Edit Online](#)

	Value
Rule ID	CA1416
Category	Interoperability
Fix is breaking or non-breaking	Non-breaking

Cause

Violations are reported if a platform-specific API is used in the context of a different platform or if the platform isn't verified (platform-neutral). Violations are also reported if an API that's not supported for the target platform of the project is used.

This rule is enabled by default only for projects that target .NET 5 or later. However, you can [enable](#) it for projects that target other frameworks.

Rule description

.NET 5.0 added new attributes, [SupportedOSPlatformAttribute](#) and [UnsupportedOSPlatformAttribute](#), to annotate platform-specific APIs. Both attributes can be instantiated with or without version numbers as part of the platform name. They can also be applied multiple times with different platforms.

- An unannotated API is considered to work on all operating system (OS) platforms.
- An API marked with `[SupportedOSPlatform("platformName")]` is considered to be portable to the specified OS platforms only. If the platform is a [subset of another platform](#), the attribute implies that that platform is also supported.
- An API marked with `[UnsupportedOSPlatform("platformName")]` is considered to be unsupported on the specified OS platforms. If the platform is a [subset of another platform](#), the attribute implies that that platform is also unsupported.

You can combine `[SupportedOSPlatform]` and `[UnsupportedOSPlatform]` attributes on a single API. In this case, the following rules apply:

- **Allow list.** If the lowest version for each OS platform is a `[SupportedOSPlatform]` attribute, the API is considered to only be supported by the listed platforms and unsupported by all other platforms. The list can have an `[UnsupportedOSPlatform]` attribute with the same platform, but only with a higher version, which denotes that the API is removed from that version.
- **Deny list.** If the lowest version for each OS platform is an `[UnsupportedOSPlatform]` attribute, then the API is considered to only be unsupported by the listed platforms and supported by all other platforms. The list can have a `[SupportedOSPlatform]` attribute with the same platform, but only with a higher version, which denotes that the API is supported since that version.
- **Inconsistent list.** If the lowest version for some platforms is `[SupportedOSPlatform]` but `[UnsupportedOSPlatform]` for other platforms, this combination is considered inconsistent. Some annotations on the API are ignored. In the future, we may introduce an analyzer that produces a warning in case of inconsistency.

If you access an API annotated with these attributes from the context of a different platform, you can see CA1416 violations.

TFM target platforms

The analyzer does not check target framework moniker (TFM) target platforms from MSBuild properties, such as `<TargetFramework>` or `<TargetFrameworks>`. If the TFM has a target platform, the .NET SDK injects a `SupportedOSPlatform` attribute with the targeted platform name in the `AssemblyInfo.cs` file, which is consumed by the analyzer. For example, if the TFM is `net5.0-windows10.0.0.19041`, the SDK injects the `[assembly: System.Runtime.Versioning.SupportedOSPlatform("windows10.0.0.19041")]` attribute into the `AssemblyInfo.cs` file, and the entire assembly is considered to be Windows only. Therefore, calling Windows-only APIs versioned with 7.0 or below would not cause any warnings in the project.

NOTE

If the `AssemblyInfo.cs` file generation is disabled for the project (that is, the `<GenerateAssemblyInfo>` property is set to `false`), the required assembly level `SupportedOSPlatform` attribute can't be added by the SDK. In this case, you could see warnings for a platform-specific APIs usage even if you're targeting that platform. To resolve the warnings, enable the `AssemblyInfo.cs` file generation or add the attribute manually in your project.

Violations

- If you access an API that's supported only on a specified platform (`[SupportedOSPlatform("platformName")]`) from code reachable on other platforms, you'll see the following violation: '**API**' is supported on '**platformName**'.

```
// An API supported only on Linux.  
[SupportedOSPlatform("linux")]  
public void LinuxOnlyApi() { }  
  
// API is supported on Windows, iOS from version 14.0, and MacCatalyst from version 14.0.  
[SupportedOSPlatform("windows")]  
[SupportedOSPlatform("ios14.0")] // MacCatalyst is a superset of iOS, therefore it's also supported.  
public void SupportedOnWindowsIos14AndMacCatalyst14() { }  
  
public void Caller()  
{  
    LinuxOnlyApi(); // This call site is reachable on all platforms. 'LinuxOnlyApi()' is only  
    supported on: 'linux'  
  
    SupportedOnWindowsIos14AndMacCatalyst14(); // This call site is reachable on all platforms.  
    'SupportedOnWindowsIos14AndMacCatalyst14()' // is only supported on: 'windows', 'ios' 14.0 and  
    later, 'MacCatalyst' 14.0 and later.  
}
```

NOTE

A violation only occurs if the project does not target the supported platform (`net5.0-differentPlatform`). This also applies to multi-targeted projects. No violation occurs if the project targets the specified platform (`net5.0-platformName`) and the `AssemblyInfo.cs` file generation is enabled for the project.

- Accessing an API that's attributed with `[UnsupportedOSPlatform("platformName")]` from a context that targets the unsupported platform could produce a violation: '**API**' is unsupported on '**platformName**'.

```

// An API not supported on Android but supported on all other platforms.
[UnsupportedOSPlatform("android")]
public void DoesNotWorkOnAndroid() { }

// An API was unsupported on Windows until version 10.0.18362.
// The API is considered supported everywhere else without constraints.
[UnsupportedOSPlatform("windows")]
[SupportedOSPlatform("windows10.0.18362")]
public void StartedWindowsSupportFromCertainVersion() { }

public void Caller()
{
    DoesNotWorkOnAndroid(); // This call site is reachable on all platforms. 'DoesNotWorkOnAndroid()' is unsupported on: 'android'

    StartedWindowsSupportFromCertainVersion(); // This call site is reachable on all platforms. 'StartedWindowsSupportFromCertainVersion()' is unsupported on: 'windows' 10.0.18362 and before
}

```

NOTE

If you're building an app that doesn't target the unsupported platform, you won't get any violations. A violation only occurs in the following cases:

- The project targets the platform attributed as unsupported.
- The `platformName` is included in the default MSBuild `<SupportedPlatform>` items group.
- `platformName` is manually included in the MSBuild `<SupportedPlatform>` items group.

```

<ItemGroup>
    <SupportedPlatform Include="platformName" />
</ItemGroup>

```

How to fix violations

The recommended way to deal with violations is to make sure you only call platform-specific APIs when running on an appropriate platform. You can achieve this by excluding the code at build time using `#if` and multi-targeting, or by conditionally calling the code at run time. The analyzer recognizes the platform guards in the [OperatingSystem](#) class and [System.Runtime.InteropServices.RuntimeInformation.IsOSPlatform](#).

- Suppress violations by surrounding the call site with the standard platform guard methods or custom guard APIs annotated with `SupportedOSPlatformGuardAttribute` OR `UnsupportedOSPlatformGuardAttribute`.

```

// An API supported only on Linux.
[SupportedOSPlatform("linux")]
public void LinuxOnlyApi() { }

// API is supported on Windows, iOS from version 14.0, and MacCatalyst from version 14.0.
[SupportedOSPlatform("windows")]
[SupportedOSPlatform("ios14.0")] // MacCatalyst is a superset of iOS, therefore it's also supported.
public void SupportedOnWindowsIos14AndMacCatalyst14() { }

public void Caller()
{
    LinuxOnlyApi(); // This call site is reachable on all platforms. 'LinuxOnlyApi()' is only supported on: 'linux'.

    SupportedOnWindowsIos14AndMacCatalyst14(); // This call site is reachable on all platforms.
    'SupportedOnWindowsIos14AndMacCatalyst14()' // is only supported on: 'windows', 'ios' 14.0 and later, 'MacCatalyst' 14.0 and later.
}

```

```
}

[SupportedOSPlatformGuard("windows")] // The platform guard attributes used
[SupportedOSPlatformGuard("ios14.0")]
private readonly bool _isWindowOrIOS14 = OperatingSystem.IsWindows() ||
OperatingSystem.IsIOSVersionAtLeast(14);

// The warnings are avoided using platform guard methods.
public void Caller()
{
    if (OperatingSystem.IsLinux()) // standard guard examples
    {
        LinuxOnlyApi(); // no diagnostic
    }

    if (OperatingSystem.IsIOSVersionAtLeast(14))
    {
        SupportedOnWindowsAndIos14(); // no diagnostic
    }

    if (RuntimeInformation.IsOSPlatform(OSPlatform.Windows))
    {
        SupportedOnWindowsAndIos14(); // no diagnostic
    }

    if (_isWindowOrMacOS14) // custom guard example
    {
        SupportedOnWindowsAndIos14(); // no diagnostic
    }
}

// An API not supported on Android but supported on all other platforms.
[UnsupportedOSPlatform("android")]
public void DoesNotWorkOnAndroid() { }

// An API was unsupported on Windows until version 10.0.18362.
// The API is considered supported everywhere else without constraints.
[UnsupportedOSPlatform("windows")]
[SupportedOSPlatform("windows10.0.18362")]
public void StartedWindowsSupportFromCertainVersion();

public void Caller()
{
    DoesNotWorkOnAndroid(); // This call site is reachable on all platforms.'DoesNotWorkOnAndroid()'
    is unsupported on: 'android'

    StartedWindowsSupportFromCertainVersion(); // This call site is reachable on all platforms.
    'StartedWindowsSupportFromCertainVersion()' is unsupported on: 'windows' 10.0.18362 and before.
}

[UnsupportedOSPlatformGuard("android")] // The platform guard attribute
bool IsNotAndroid => !OperatingSystem.IsAndroid();

public void Caller()
{
    if (!OperatingSystem.IsAndroid()) // using standard guard methods
    {
        DoesNotWorkOnAndroid(); // no diagnostic
    }

    // Use the && and || logical operators to guard combined attributes.
    if (!OperatingSystem.IsWindows() || OperatingSystem.IsWindowsVersionAtLeast(10, 0, 18362))
    {
        StartedWindowsSupportFromCertainVersion(); // no diagnostic
    }

    if (IsNotAndroid) // custom guard example
    {
        DoesNotWorkOnAndroid(); // no diagnostic
    }
}
```

```
        DOESNOTCONTAINDEBUG(), // No diagnostic
    }
}
```

- The analyzer also respects [System.Diagnostics.Debug.Assert](#) as a means for preventing the code from being reached on unsupported platforms. Using `Debug.Assert` allows the check to be trimmed out of release builds, if desired.

```
// An API supported only on Linux.
[SupportedOSPlatform("linux")]
public void LinuxOnlyApi() { }

public void Caller()
{
    Debug.Assert(OperatingSystem.IsLinux());

    LinuxOnlyApi(); // No diagnostic
}
```

- You can choose to mark your own APIs as being platform-specific, effectively forwarding the requirements to your callers. You can apply platform attributes to any of the following APIs:

- Types
- Members (methods, fields, properties, and events)
- Assemblies

```
[SupportedOSPlatform("windows")]
[SupportedOSPlatform("ios14.0")]
public void SupportedOnWindowsAndIos14() { }

[SupportedOSPlatform("ios15.0")] // call site version should be equal to or higher than the API
// version
public void Caller()
{
    SupportedOnWindowsAndIos14(); // No diagnostics
}

[UnsupportedOSPlatform("windows")]
[SupportedOSPlatform("windows10.0.18362")]
public void StartedWindowsSupportFromCertainVersion();

[UnsupportedOSPlatform("windows")]
[SupportedOSPlatform("windows10.0.18362")]
public void Caller()
{
    StartedWindowsSupportFromCertainVersion(); // No diagnostics
}
```

- When an assembly or type-level attribute is applied, all members within the assembly or type are considered to be platform specific.

```
[assembly:SupportedOSPlatform("windows")]
public namespace ns
{
    public class Sample
    {
        public void SupportedOnWindows() { }

        public void Caller()
        {
            SupportedOnWindows(); // No diagnostic as call site and calling method both windows only
        }
    }
}
```

When to suppress warnings

Referencing platform-specific APIs without a proper platform context or guard is not recommended. However, you can suppress these diagnostics using `#pragma` or the **NoWarn** compiler flag, or by [setting the rule's severity](#) to `none` in an `.editorconfig` file.

```
[SupportedOSPlatform("linux")]
public void LinuxOnlyApi() { }

public void Caller()
{
    #pragma warning disable CA1416
    LinuxOnlyApi();
    #pragma warning restore CA1416
}
```

Configure code to analyze

The analyzer is enabled by default only for projects that target .NET 5 or later and have an [AnalysisLevel](#) of 5 or higher. You can enable it for target frameworks lower than `net5.0` by adding the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.enable_platform_analyzer_on_pre_net5_target=true
```

See also

- [Platform compatibility analyzer \(conceptual\)](#)
- [Annotating platform-specific APIs and detecting its use](#)
- [Annotating APIs as unsupported on specific platforms](#)
- [Target Framework Names in .NET 5](#)
- [Interoperability rules](#)

CA1417: Do not use `[OutAttribute]` on string parameters for P/Invokes

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA1417
Category	Interoperability
Fix is breaking or non-breaking	Non-breaking

Cause

A [P/Invoke](#) string parameter is passed by value and marked with [OutAttribute](#).

Rule description

The .NET runtime automatically performs [string interning](#). If an interned string marked with [OutAttribute](#) is passed by value to a P/Invoke, the runtime can be destabilized.

How to fix violations

If marshalling of modified string data back to the caller is required, pass the string by reference instead. Otherwise, the [OutAttribute](#) can be removed without any other changes.

```
// Violation
[DllImport("MyLibrary")]
private static extern void Foo([Out] string s);

// Fixed: passed by reference
[DllImport("MyLibrary")]
private static extern void Foo(out string s);

// Fixed: marshalling data back to caller is not required
[DllImport("MyLibrary")]
private static extern void Foo(string s);
```

When to suppress warnings

It is not safe to suppress a warning from this rule.

See also

- [Interoperability rules](#)
- [Native interoperability best practices](#)

CA1418: Validate platform compatibility

9/20/2022 • 3 minutes to read • [Edit Online](#)

	Value
Rule ID	CA1418
Category	Interoperability
Fix is breaking or non-breaking	Non-breaking

Cause

The platform compatibility analyzer requires a valid platform name and version. Violations are reported if the platform string provided to the [OSPlatformAttribute](#) constructor consists of an unknown platform name or if the optional version part is invalid.

Rule description

The platform compatibility attributes derived from [OSPlatformAttribute](#) use string literals for operating system (OS) platform names with an optional version part. The string should consist of a known platform name and either no version part or a valid version part.

The known platform names list is populated from two places:

- The `PlatformName` part of [OperatingSystem](#) guard methods named `OperatingSystem.Is<PlatformName>[VersionAtLeast]()`. For example, guard method `OperatingSystem.IsWindows()` adds `Windows` to the known platform names list.
- The project's MSBuild item group of `SupportedPlatform` items, including the default [MSBuild SupportedPlatforms list](#). This is the project specific knowledge of known platforms. It allows class library authors to add more platforms into the known platforms list. For example:

```
<ItemGroup>
  <SupportedPlatform Include="PlatformName" />
</ItemGroup>
```

If the platform string contains a *version* part, it should be a valid [Version](#) with the following format:

```
major.minor[.build[.revision]]
```

Violations

- `Solaris` is an **unknown** platform name because it's not included in the default [MSBuild SupportedPlatforms list](#) and there's no guard method named `OperatingSystem.IsSolaris()` in the [OperatingSystem](#) class.

```
[SupportedOSPlatform("Solaris")] // Warns: The platform 'Solaris' is not a known platform name.
public void SolarisApi() { }
```

- `Android` is a **known** platform because there is a `OperatingSystem.IsAndroid()` guard method in the

[OperatingSystem](#) type. However, the version part is not a valid version. It should have at least two integers separated by a dot.

```
[UnsupportedOSPlatform("Android10")] // Warns: Version '10' is not valid for platform 'Android'. Use  
a version with 2-4 parts for this platform.  
public void DoesNotWorkOnAndroid() { }
```

- `Linux` is a **known** platform because it is included in the default [MSBuild SupportedPlatforms list](#), and there's also a guard method named [OperatingSystem.IsLinux\(\)](#). However, there are no versioned guard methods like `System.OperatingSystem.IsLinuxVersionAtLeast(int,int)` for the `Linux` platform, therefore no version part is supported on Linux.

```
[SupportedOSPlatform("Linux4.8")] // Warns: Version '4.8' is not valid for platform 'Linux'. Do not  
use versions for this platform.  
public void LinuxApi() { }
```

How to fix violations

- Change the platform to a known platform name.
- If the platform name is correct and you want to make it a known platform, then add it to the MSBuild SupportedPlatforms list in your project file:

```
<ItemGroup>  
  <SupportedPlatform Include="Solaris" />  
</ItemGroup>
```

```
[SupportedOSPlatform("Solaris")] // No warning  
public void SolarisApi() { }
```

- Fix the invalid version. For example, for `Android`, `10` is not a valid version, but `10.0` is valid.

```
// Before  
[UnsupportedOSPlatform("Android10")] // Warns: Version '10' is not valid for platform 'Android'. Use  
a version with 2-4 parts for this platform.  
public void DoesNotWorkOnAndroid() { }  
  
// After  
[UnsupportedOSPlatform("Android10.0")] // No warning.  
public void DoesNotWorkOnAndroid() { }
```

- If the platform doesn't support a version, remove the version part.

```
// Before  
[SupportedOSPlatform("Linux4.8")] // Warns: Version '4.8' is not valid for platform 'Linux'. Do not  
use versions for this platform.  
public void LinuxApi() { }  
  
// After  
[SupportedOSPlatform("Linux")] // No warning.  
public void LinuxApi() { }
```

When to suppress warnings

Using an unknown platform name or an invalid version is not recommended.

Suppress a warning

One way to suppress a warning is to disable the rule for a file, folder, or project by setting its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1418.severity = none
```

```
#pragma warning disable CA1418
[SupportedOSPlatform("platform")]
#pragma warning restore CA1418
public void PlatformSpecificApi() { }
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [CA1416 Platform compatibility analyzer](#)
- [Platform compatibility analyzer \(conceptual\)](#)
- [Interoperability rules](#)

CA1419: Provide a parameterless constructor that is as visible as the containing type for concrete types derived from

'System.Runtime.InteropServices.SafeHandle'

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA1419
Category	Interoperability
Fix is breaking or non-breaking	Non-breaking

Cause

A concrete [SafeHandle](#) type requires a parameterless constructor that is at least as visible as the containing type.

Rule description

Providing a public parameterless constructor for a type derived from [SafeHandle](#) enables better performance and usage with source-generated interop solutions.

How to fix violations

Add a parameterless constructor to your type.

When to suppress warnings

Do not suppress a warning from this rule.

See also

- [Interoperability rules](#)
- [Native interoperability best practices](#)

Maintainability rules

9/20/2022 • 2 minutes to read • [Edit Online](#)

Maintainability rules support library and application maintenance.

In this section

RULE	DESCRIPTION
CA1501: Avoid excessive inheritance	A type is more than four levels deep in its inheritance hierarchy. Deeply nested type hierarchies can be difficult to follow, understand, and maintain.
CA1502: Avoid excessive complexity	This rule measures the number of linearly independent paths through the method, which is determined by the number and complexity of conditional branches.
CA1505: Avoid unmaintainable code	A type or method has a low maintainability index value. A low maintainability index indicates that a type or method is probably difficult to maintain and would be a good candidate for redesign.
CA1506: Avoid excessive class coupling	This rule measures class coupling by counting the number of unique type references that a type or method contains.
CA1507: Use nameof in place of string	A string literal is used as an argument where a <code>nameof</code> expression could be used.
CA1508: Avoid dead conditional code	A method has conditional code that always evaluates to <code>true</code> or <code>false</code> at run time. This leads to dead code in the <code>false</code> branch of the condition.
CA1509: Invalid entry in code metrics configuration file	Code metrics rules, such as CA1501 , CA1502 , CA1505 and CA1506 , supplied a configuration file named <code>CodeMetricsConfig.txt</code> that has an invalid entry.

See also

- [Measure Complexity and Maintainability of Managed Code](#)

CA1501: Avoid excessive inheritance

9/20/2022 • 3 minutes to read • [Edit Online](#)

	Value
Rule ID	CA1501
Category	Maintainability
Fix is breaking or non-breaking	Breaking

Cause

A type is more than four levels deep in its inheritance hierarchy.

By default, the rule only excludes types from the `System` namespace, but this is [configurable](#).

Rule description

Deeply nested type hierarchies can be difficult to follow, understand, and maintain. This rule limits analysis to hierarchies in the same module.

How to fix violations

To fix a violation of this rule, derive the type from a base type that is less deep in the inheritance hierarchy or eliminate some of the intermediate base types.

When to suppress warnings

It is safe to suppress a warning from this rule. However, the code might be more difficult to maintain. Depending on the visibility of base types, resolving violations of this rule might create breaking changes. For example, removing public base types is a breaking change.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1501
// The code that's violating the rule is on this line.
#pragma warning restore CA1501
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1501.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Maintainability.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

Use the following option to configure which parts of your codebase to run this rule on.

- [Inheritance excluded type or namespace names](#)

You can configure this option for just this rule, for all rules, or for all rules in this category ([Maintainability](#)). For more information, see [Code quality rule configuration options](#).

Inheritance excluded type or namespace names

You can configure the rule to exclude certain types or namespaces from the inheritance hierarchy tree. By default, all types from the `System.*` namespace are excluded. No matter what value you set, this default value is added.

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CA1501.additional_inheritance_excluded = MyType</code>	Matches all types named <code>MyType</code> or whose containing namespace contains <code>MyType</code> (and all types from the <code>System</code> namespace)
<code>dotnet_code_quality.CA1501.additional_inheritance_excluded = MyType1 MyType2</code>	Matches all types named either <code>MyType1</code> or <code>MyType2</code> or whose containing namespace contains either <code>MyType1</code> or <code>MyType2</code> (and all types from the <code>System</code> namespace)
<code>dotnet_code_quality.CA1501.additional_inheritance_excluded = T:NS.MyType</code>	Matches specific type <code>MyType</code> in the namespace <code>NS</code> (and all types from the <code>System</code> namespace)
<code>dotnet_code_quality.CA1501.additional_inheritance_excluded = T:NS1.MyType1 T:NS2.MyType2</code>	Matches specific types <code>MyType1</code> and <code>MyType2</code> with respective fully qualified names (and all types from the <code>System</code> namespace)
<code>dotnet_code_quality.CA1501.additional_inheritance_excluded = N:NS</code>	Matches all types from the <code>NS</code> namespace (and all types from the <code>System</code> namespace)
<code>dotnet_code_quality.CA1501.additional_inheritance_excluded = My*</code>	Matches all types whose name starts with <code>My</code> or whose containing namespace parts starts with <code>My</code> (and all types from the <code>System</code> namespace)
<code>dotnet_code_quality.CA1501.additional_inheritance_excluded = T:NS.My*</code>	Matches all types whose name starts with <code>My</code> in the namespace <code>NS</code> (and all types from the <code>System</code> namespace)
<code>dotnet_code_quality.CA1501.additional_inheritance_excluded = N:My*</code>	Matches all types whose containing namespace starts with <code>My</code> (and all types from the <code>System</code> namespace)

Example

The following example shows a type that violates the rule:

```
class BaseClass {}
class FirstDerivedClass : BaseClass {}
class SecondDerivedClass : FirstDerivedClass {}
class ThirdDerivedClass : SecondDerivedClass {}
class FourthDerivedClass : ThirdDerivedClass {}

// This class violates the rule.
class FifthDerivedClass : FourthDerivedClass {}
```

```
Imports System

Namespace ca1501

    Class BaseClass
        End Class

    Class FirstDerivedClass
        Inherits BaseClass
        End Class

    Class SecondDerivedClass
        Inherits FirstDerivedClass
        End Class

    Class ThirdDerivedClass
        Inherits SecondDerivedClass
        End Class

    Class FourthDerivedClass
        Inherits ThirdDerivedClass
        End Class

    ' This class violates the rule.
    Class FifthDerivedClass
        Inherits FourthDerivedClass
        End Class

End Namespace
```

CA1502: Avoid excessive complexity

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA1502
Category	Maintainability
Fix is breaking or non-breaking	Non-breaking

Cause

A method has an excessive cyclomatic complexity.

Rule description

Cyclomatic complexity measures the number of linearly independent paths through the method, which is determined by the number and complexity of conditional branches. A low cyclomatic complexity generally indicates a method that is easy to understand, test, and maintain. The cyclomatic complexity is calculated from a control flow graph of the method and is given as follows:

cyclomatic complexity = the number of edges - the number of nodes + 1

A *node* represents a logic branch point and an *edge* represents a line between nodes.

The rule reports a violation when the cyclomatic complexity is more than 25.

You can learn more about code metrics at [Measure complexity of managed code](#).

How to fix violations

To fix a violation of this rule, refactor the method to reduce its cyclomatic complexity.

When to suppress warnings

It is safe to suppress a warning from this rule if the complexity cannot easily be reduced and the method is easy to understand, test, and maintain. In particular, a method that contains a large `switch` (`Select` in Visual Basic) statement is a candidate for exclusion. The risk of destabilizing the code base late in the development cycle or introducing an unexpected change in run-time behavior in previously shipped code might outweigh the maintainability benefits of refactoring the code.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1502
// The code that's violating the rule is on this line.
#pragma warning restore CA1502
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1502.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Maintainability.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

How cyclomatic complexity is calculated

The cyclomatic complexity is calculated by adding 1 to the following:

- Number of branches (such as `if`, `while`, and `do`)
- Number of `case` statements in a `switch`

Examples

The following examples show methods that have varying cyclomatic complexities.

Cyclomatic complexity of 1

```
public void Method()
{
    Console.WriteLine("Hello World!");
}
```

```
Public Sub Method()
    Console.WriteLine("Hello World!")
End Sub
```

Cyclomatic complexity of 2

```
void Method(bool condition)
{
    if (condition)
    {
        Console.WriteLine("Hello World!");
    }
}
```

```
Public Sub Method(ByVal condition As Boolean)
    If (condition) Then
        Console.WriteLine("Hello World!")
    End If
End Sub
```

Cyclomatic complexity of 3

```

public void Method(bool condition1, bool condition2)
{
    if (condition1 || condition2)
    {
        Console.WriteLine("Hello World!");
    }
}

```

```

Public Sub Method(ByVal condition1 As Boolean, ByVal condition2 As Boolean)
    If (condition1 OrElse condition2) Then
        Console.WriteLine("Hello World!")
    End If
End Sub

```

Cyclomatic complexity of 8

```

public void Method(DayOfWeek day)
{
    switch (day)
    {
        case DayOfWeek.Monday:
            Console.WriteLine("Today is Monday!");
            break;
        case DayOfWeek.Tuesday:
            Console.WriteLine("Today is Tuesday!");
            break;
        case DayOfWeek.Wednesday:
            Console.WriteLine("Today is Wednesday!");
            break;
        case DayOfWeek.Thursday:
            Console.WriteLine("Today is Thursday!");
            break;
        case DayOfWeek.Friday:
            Console.WriteLine("Today is Friday!");
            break;
        case DayOfWeek.Saturday:
            Console.WriteLine("Today is Saturday!");
            break;
        case DayOfWeek.Sunday:
            Console.WriteLine("Today is Sunday!");
            break;
    }
}

```

```
Public Sub Method(ByVal day As DayOfWeek)
    Select Case day
        Case DayOfWeek.Monday
            Console.WriteLine("Today is Monday!")
        Case DayOfWeek.Tuesday
            Console.WriteLine("Today is Tuesday!")
        Case DayOfWeek.Wednesday
            Console.WriteLine("Today is Wednesday!")
        Case DayOfWeek.Thursday
            Console.WriteLine("Today is Thursday!")
        Case DayOfWeek.Friday
            Console.WriteLine("Today is Friday!")
        Case DayOfWeek.Saturday
            Console.WriteLine("Today is Saturday!")
        Case DayOfWeek.Sunday
            Console.WriteLine("Today is Sunday!")
    End Select
End Sub
```

Related rules

[CA1501: Avoid excessive inheritance](#)

See also

- [Measuring Complexity and Maintainability of Managed Code](#)

CA1505: Avoid unmaintainable code

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA1505
Category	Maintainability
Fix is breaking or non-breaking	Non-breaking

Cause

A type or method has a low maintainability index value.

Rule description

The maintainability index is calculated by using the following metrics: lines of code, program volume, and cyclomatic complexity. Program volume is a measure of the difficulty of understanding of a type or method that's based on the number of operators and operands in the code. Cyclomatic complexity is a measure of the structural complexity of the type or method. You can learn more about code metrics at [Measure complexity and maintainability of managed code](#).

A low maintainability index indicates that a type or method is probably difficult to maintain and would be a good candidate to redesign.

How to fix violations

To fix this violation, redesign the type or method and try to split it into smaller and more focused types or methods.

When to suppress warnings

You can suppress this warning when the type or method cannot be split or is considered maintainable despite its large size.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1505
// The code that's violating the rule is on this line.
#pragma warning restore CA1505
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1505.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Maintainability.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Maintainability rules](#)
- [Measure complexity and maintainability of managed code](#)

CA1506: Avoid excessive class coupling

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA1506
Category	Maintainability
Fix is breaking or non-breaking	Breaking

Cause

A type or method is coupled with many other types. Compiler-generated types are excluded from this metric.

Rule description

This rule measures class coupling by counting the number of unique type references that a type or method contains. The default coupling threshold is 95 for types and 40 for methods.

Types and methods that have a high degree of class coupling can be difficult to maintain. It's a good practice to have types and methods that exhibit low coupling and high cohesion.

How to fix violations

To fix this violation, try to redesign the type or method to reduce the number of types to which it's coupled.

When to suppress warnings

You can suppress this warning when the type or method is considered maintainable despite its large number of dependencies on other types.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1506
// The code that's violating the rule is on this line.
#pragma warning restore CA1506
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1506.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]\ndotnet_analyzer_diagnostic.category-Maintainability.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Maintainability rules](#)
- [Measuring Complexity and Maintainability of Managed Code](#)

CA1507: Use `nameof` in place of string

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA1507
Category	Maintainability
Fix is breaking or non-breaking	Non-breaking

Cause

A `string` literal or constant that matches the name of a parameter of the containing method or the name of a property of the containing type is used as an argument to a method.

Rule description

Rule CA1507 flags the use of a `string` literal as an argument to a method or constructor where a `nameof` (`NameOf` in Visual Basic) expression would add maintainability. The rule fires if all of the following conditions are met:

- The argument is a `string` literal or constant.
- The argument corresponds to a `string`-typed parameter of the method or the constructor that's being invoked (that is, there is no conversion involved at the call site).
- Either:
 - The declared name of the parameter is `paramName` and the constant value of the `string` literal matches the name of a parameter of the method, lambda, or local function within which the method or constructor is being invoked.
 - The declared name of the parameter is `propertyName` and the constant value of the `string` literal matches the name of a property of the type within which the method or constructor is being invoked.

Rule CA1507 improves code maintainability in cases where the parameter may be renamed in the future, but the `string` literal is mistakenly not renamed. By using `nameof`, the symbol will be renamed when the parameter is renamed through a refactoring operation. In addition, any spelling mistakes in the name of the parameter are caught by the compiler.

How to fix violations

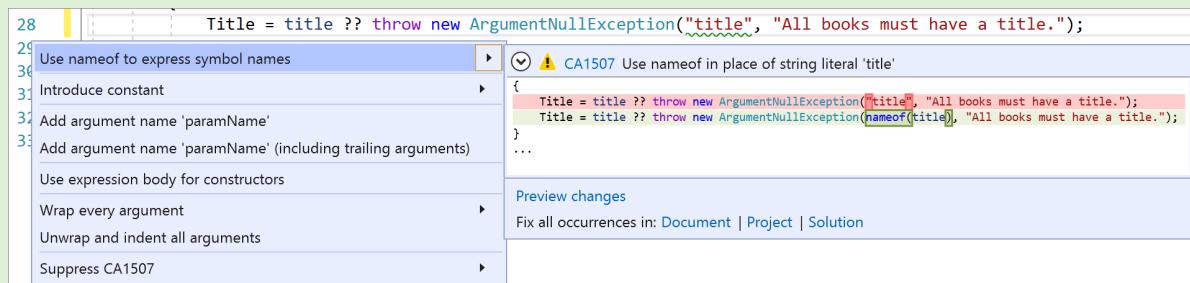
To fix a violation, replace the `string` literal with a `nameof` (`NameOf` in Visual Basic) expression. For example, the following two code snippets show a violation of the rule and how to fix it:

```
public Book(string title)
{
    // Violates rule CA1507
    Title = title ?? throw new ArgumentNullException("title", "All books must have a title.");
}
```

```
public Book(string title)
{
    // Resolves rule CA1507 violation
    Title = title ?? throw new ArgumentNullException(nameof(title), "All books must have a title.");
}
```

TIP

A code fix is available for this rule in Visual Studio. To use it, position the cursor on the `string` literal and press **Ctrl+.** (period). Choose **Use nameof to express symbol names** from the list of options that's presented.



When to suppress warnings

It's safe to suppress a violation of this rule if you're not concerned about the maintainability of your code.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1507
// The code that's violating the rule is on this line.
#pragma warning restore CA1507
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1507.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Maintainability.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Related rules

- [CA2208: Instantiate argument exceptions correctly](#)

See also

- [Maintainability rules](#)

CA1508: Avoid dead conditional code

9/20/2022 • 4 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA1508
Category	Maintainability
Fix is breaking or non-breaking	Non-Breaking

Cause

A method has conditional code that always evaluates to `true` or `false` at run time. This leads to dead code in the `false` branch of the condition.

By default, this rule analyzes the entire codebase, but this is [configurable](#).

Rule description

Methods can have conditional code, such as if statements, binary expressions (`==`, `!=`, `<`, `>`), null checks, etc. For example, consider the following code:

```
public void M(int i, int j)
{
    if (i != 0)
    {
        return;
    }

    if (j != 0)
    {
        return;
    }

    // Below condition will always evaluate to 'false' as 'i' and 'j' are both '0' here.
    if (i != j)
    {
        // Code in this 'if' branch is dead code.
        // It can either be removed or refactored.
        ...
    }
}
```

C# and VB compilers perform analysis of conditional checks involving compile-time *constant values* that always evaluate to `true` or `false`. This analyzer performs data flow analysis of non-constant variables to determine redundant conditional checks involving *non-constant values*. In the preceding code, the analyzer determines that `i` and `j` are both `0` for all code paths that reach `i != j` check. Hence, this check will always evaluate to `false` at run time. The code inside the if statement is dead code and can be removed or refactored. Similarly, the analyzer tracks nullness of variables and reports redundant null checks.

NOTE

This analyzer performs an expensive dataflow analysis of non-constant values. This can increase the overall compile time on certain code bases.

When to suppress warnings

It's safe to suppress a violation of this rule if you're not concerned about the maintainability of your code. It is also fine to suppress violations that are identified to be false positives. These are possible in the presence of concurrent code that can execute from multiple threads.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1508
// The code that's violating the rule is on this line.
#pragma warning restore CA1508
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1508.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Maintainability.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

Use the following options to configure which parts of your codebase to run this rule on.

- [Exclude specific symbols](#)
- [Exclude specific types and their derived types](#)

You can configure these options for just this rule, for all rules, or for all rules in this category ([Maintainability](#)).

For more information, see [Code quality rule configuration options](#).

Exclude specific symbols

You can exclude specific symbols, such as types and methods, from analysis. For example, to specify that the rule should not run on any code within types named `MyType`, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Symbol name only (includes all symbols with the name, regardless of the containing type or namespace).

- Fully qualified names in the symbol's [documentation ID format](#). Each symbol name requires a symbol-kind prefix, such as `M:` for methods, `T:` for types, and `N:` for namespaces.
- `.ctor` for constructors and `.cctor` for static constructors.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType</code>	Matches all symbols named <code>MyType</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType1 MyType2</code>	Matches all symbols named either <code>MyType1</code> or <code>MyType2</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS.MyType.MyMethod(ParamType)</code>	Matches specific method <code>MyMethod</code> with the specified fully qualified signature.
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS1.MyType1.MyMethod1(ParamType) M:NS2.MyType2.MyMethod2(ParamType)</code>	Matches specific methods <code>MyMethod1</code> and <code>MyMethod2</code> with the respective fully qualified signatures.

Exclude specific types and their derived types

You can exclude specific types and their derived types from analysis. For example, to specify that the rule should not run on any methods within types named `MyType` and their derived types, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Type name only (includes all types with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#), with an optional `T:` prefix.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType</code>	Matches all types named <code>MyType</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType1 MyType2</code>	Matches all types named either <code>MyType1</code> or <code>MyType2</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS.MyType</code>	Matches specific type <code>MyType</code> with given fully qualified name and all of its derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS1.MyType1 M:NS2.MyType2</code>	Matches specific types <code>MyType1</code> and <code>MyType2</code> with the respective fully qualified names, and all of their derived types.

See also

- [Maintainability rules](#)

CA1509: Invalid entry in code metrics configuration file

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA1509
Category	Maintainability
Fix is breaking or non-breaking	Non-Breaking

Cause

Code metrics rules, such as [CA1501](#), [CA1502](#), [CA1505](#) and [CA1506](#), supplied a configuration file named `CodeMetricsConfig.txt` that has an invalid entry.

Rule description

.NET code-quality analyzers implementation of [code metrics](#) analysis rules allow end users to supply an [additional file](#) named `CodeMetricsConfig.txt`. This file contains entries to configure code metric thresholds for analysis. Following rules are configurable in this file:

- [CA1501: Avoid excessive inheritance](#)
- [CA1502: Avoid excessive complexity](#)
- [CA1505: Avoid unmaintainable code](#)
- [CA1506: Avoid excessive class coupling](#)

This configuration file expects each entry to be in following format:

```
'RuleId'(Optional 'SymbolKind'): 'Threshold'
```

- Valid values for 'RuleId' are [CA1501](#), [CA1502](#), [CA1505](#), and [CA1506](#).
- Valid values for optional 'SymbolKind' are [Assembly](#), [Namespace](#), [Type](#), [Method](#), [Field](#), [Event](#), and [Property](#).
- Valid values for 'Threshold' are non-negative integers.
- Lines starting with '#' are treated as comment lines

For example, the following is a valid configuration file:

```
# Comment text

CA1501: 1

CA1502(Type): 4
CA1502(Method): 2
```

An invalid entry in this configuration file is flagged with the [CA1509](#) diagnostic.

How to fix violations

To fix a violation of this rule, make sure the invalid entry in `CodeMetricsConfig.txt` gets the required format.

When to suppress warnings

Do not suppress violations of this rule.

Related rules

- [CA1501: Avoid excessive inheritance](#)
- [CA1502: Avoid excessive complexity](#)
- [CA1505: Avoid unmaintainable code](#)
- [CA1506: Avoid excessive class coupling](#)

See also

- [Maintainability rules](#)
- [Measure complexity and maintainability of managed code](#)

Naming rules

9/20/2022 • 3 minutes to read • [Edit Online](#)

Naming rules support adherence to the [naming conventions of the .NET design guidelines](#).

In this section

RULE	DESCRIPTION
CA1700: Do not name enum values 'Reserved'	This rule assumes that an enumeration member that has a name that contains "reserved" is not currently used but is a placeholder to be renamed or removed in a future version. Renaming or removing a member is a breaking change.
CA1707: Identifiers should not contain underscores	By convention, identifier names do not contain the underscore (_) character. This rule checks namespaces, types, members, and parameters.
CA1708: Identifiers should differ by more than case	Identifiers for namespaces, types, members, and parameters cannot differ only by case because languages that target the common language runtime are not required to be case-sensitive.
CA1710: Identifiers should have correct suffix	By convention, the names of types that extend certain base types or that implement certain interfaces, or types derived from these types, have a suffix that is associated with the base type or interface.
CA1711: Identifiers should not have incorrect suffix	By convention, only the names of types that extend certain base types or that implement certain interfaces, or types that are derived from these types, should end with specific reserved suffixes. Other type names should not use these reserved suffixes.
CA1712: Do not prefix enum values with type name	Names of enumeration members are not prefixed with the type name because type information is expected to be provided by development tools.
CA1713: Events should not have before or after prefix	The name of an event starts with "Before" or "After". To name related events that are raised in a specific sequence, use the present or past tense to indicate the relative position in the sequence of actions.
CA1714: Flags enums should have plural names	A public enumeration has the System.FlagsAttribute attribute and its name does not end in "s". Types that are marked with FlagsAttribute have names that are plural because the attribute indicates that more than one value can be specified.
CA1715: Identifiers should have correct prefix	The name of an externally visible interface does not start with a capital "I". The name of a generic type parameter on an externally visible type or method does not start with a capital "T".

RULE	DESCRIPTION
CA1716: Identifiers should not match keywords	A namespace name or a type name matches a reserved keyword in a programming language. Identifiers for namespaces and types should not match keywords that are defined by languages that target the common language runtime.
CA1717: Only FlagsAttribute enums should have plural names	Naming conventions dictate that a plural name for an enumeration indicates that more than one value of the enumeration can be specified at the same time.
CA1720: Identifiers should not contain type names	The name of a parameter in an externally visible member contains a data type name, or the name of an externally visible member contains a language-specific data type name.
CA1721: Property names should not match get methods	The name of a public or protected member starts with "Get" and otherwise matches the name of a public or protected property. "Get" methods and properties should have names that clearly distinguish their function.
CA1724: Type Names Should Not Match Namespaces	Type names should not match the names of .NET namespaces. Violation of this rule can reduce the usability of the library.
CA1725: Parameter names should match base declaration	Consistent naming of parameters in an override hierarchy increases the usability of the method overrides. A parameter name in a derived method that differs from the name in the base declaration can cause confusion about whether the method is an override of the base method or a new overload of the method.
CA1727: Use PascalCase for named placeholders	Use PascalCase for named placeholders in the logging message template.

CA1700: Do not name enum values 'Reserved'

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA1700
Category	Naming
Fix is breaking or non-breaking	Breaking

Cause

The name of an enumeration member contains the word "reserved".

Rule description

This rule assumes that an enumeration member that has a name that contains "reserved" is not currently used but is a placeholder to be renamed or removed in a future version. Renaming or removing a member is a breaking change. You should not expect users to ignore a member just because its name contains "reserved", nor can you rely on users to read or abide by documentation. Furthermore, because reserved members appear in object browsers and smart integrated development environments, they can cause confusion about which members are actually being used.

Instead of using a reserved member, add a new member to the enumeration in the future version. In most cases the addition of the new member is not a breaking change, as long as the addition does not cause the values of the original members to change.

In a limited number of cases the addition of a member is a breaking change even when the original members retain their original values. Primarily, the new member cannot be returned from existing code paths without breaking callers that use a `switch` (`Select` in Visual Basic) statement on the return value that encompasses the whole member list and that throw an exception in the default case. A secondary concern is that client code might not handle the change in behavior from reflection methods such as `System.Enum.IsDefined`. Accordingly, if the new member has to be returned from existing methods or a known application incompatibility occurs because of poor reflection usage, the only nonbreaking solution is to:

1. Add a new enumeration that contains the original and new members.
2. Mark the original enumeration with the `System.ObsoleteAttribute` attribute.

Follow the same procedure for any externally visible types or members that expose the original enumeration.

How to fix violations

To fix a violation of this rule, remove or rename the member.

When to suppress warnings

It is safe to suppress a warning from this rule for a member that is currently used or for libraries that have previously shipped.

SUPPRESS A WARNING

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1700
// The code that's violating the rule is on this line.
#pragma warning restore CA1700
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.cs,vb]
dotnet_diagnostic.CA1700.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.cs,vb]
dotnet_analyzer_diagnostic.category-Naming.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

CONFIGURE CODE TO ANALYZE

Use the following option to configure which parts of your codebase to run this rule on.

- [Include specific API surfaces](#)

You can configure this option for just this rule, for all rules, or for all rules in this category ([Naming](#)). For more information, see [Code quality rule configuration options](#).

Include specific API surfaces

You can configure which parts of your codebase to run this rule on, based on their accessibility. For example, to specify that the rule should run only against the non-public API surface, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.api_surface = private, internal
```

RELATED RULES

[CA2217: Do not mark enums with FlagsAttribute](#)

[CA1712: Do not prefix enum values with type name](#)

[CA1028: Enum storage should be Int32](#)

[CA1008: Enums should have zero value](#)

[CA1027: Mark enums with FlagsAttribute](#)

CA1707: Identifiers should not contain underscores

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA1707
Category	Naming
Fix is breaking or non-breaking	Breaking - when raised on assemblies Non-breaking - when raised on type parameters

Cause

The name of an identifier contains the underscore (_) character.

Rule description

By convention, identifier names do not contain the underscore (_) character. The rule checks namespaces, types, members, and parameters.

Naming conventions provide a common look for libraries that target the common language runtime. This reduces the learning curve that is required for new software libraries, and increases customer confidence that the library was developed by someone who has expertise in developing managed code.

How to fix violations

Remove all underscore characters from the name.

When to suppress warnings

Do not suppress warnings for production code. However, it's safe to suppress this warning for test code.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1707
// The code that's violating the rule is on this line.
#pragma warning restore CA1707
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1707.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]\ndotnet_analyzer_diagnostic.category-Naming.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

For well-known methods in Microsoft code that currently use an underscore and cannot be modified, the rule should be suppressed.

Configure code to analyze

Use the following option to configure which parts of your codebase to run this rule on.

- [Include specific API surfaces](#)

You can configure this option for just this rule, for all rules, or for all rules in this category ([Naming](#)). For more information, see [Code quality rule configuration options](#).

Include specific API surfaces

You can configure which parts of your codebase to run this rule on, based on their accessibility. For example, to specify that the rule should run only against the non-public API surface, add the following key-value pair to an *.editorconfig* file in your project:

```
dotnet_code_quality.CAXXXX.api_surface = private, internal
```

Related rules

- [CA1708: Identifiers should differ by more than case](#)

CA1708: Identifiers should differ by more than case

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA1708
Category	Naming
Fix is breaking or non-breaking	Breaking

Cause

The names of two types, members, parameters, or fully qualified namespaces are identical when they're converted to lowercase.

By default, this rule only looks at externally visible types, members, and namespaces, but this is [configurable](#).

Rule description

Identifiers for namespaces, types, members, and parameters cannot differ only by case because languages that target the common language runtime are not required to be case-sensitive. For example, Visual Basic is a widely used case-insensitive language.

This rule fires on publicly visible members only.

How to fix violations

Select a name that is unique when it is compared to other identifiers in a case-insensitive manner.

When to suppress warnings

Do not suppress a warning from this rule. The library might not be usable in all available languages in .NET.

Configure code to analyze

Use the following option to configure which parts of your codebase to run this rule on.

- [Include specific API surfaces](#)

You can configure this option for just this rule, for all rules, or for all rules in this category ([Naming](#)). For more information, see [Code quality rule configuration options](#).

Include specific API surfaces

You can configure which parts of your codebase to run this rule on, based on their accessibility. For example, to specify that the rule should run only against the non-public API surface, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.api_surface = private, internal
```

Example of a violation

The following example demonstrates a violation of this rule.

```
public class Class1
{
    protected string someProperty;

    public string SomeProperty
    {
        get { return someProperty; }
    }
}
```

CA1710: Identifiers should have correct suffix

9/20/2022 • 4 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA1710
Category	Naming
Fix is breaking or non-breaking	Breaking

Cause

An identifier does not have the correct suffix.

By default, this rule only looks at externally visible identifiers, but this is [configurable](#).

Rule description

By convention, the names of types that extend certain base types or that implement certain interfaces, or types derived from these types, have a suffix that is associated with the base type or interface.

Naming conventions provide a common look for libraries that target the common language runtime. This reduces the learning curve that is required for new software libraries, and increases customer confidence that the library was developed by someone who has expertise in developing managed code.

The following table lists the base types and interfaces that have associated suffixes.

BASE TYPE/INTERFACE	SUFFIX
System.Attribute	Attribute
System.EventArgs	EventArgs
System.Exception	Exception
System.Collections.ICollection	Collection
System.Collections.IDictionary	Dictionary
System.Collections.IEnumerable	Collection
System.Collections.Generic.IReadOnlyDictionary< TKey, TValue >	Dictionary
System.Collections.Queue	Collection or Queue
System.Collections.Stack	Collection or Stack
System.Collections.Generic.ICollection< T >	Collection

BASE TYPE/INTERFACE	SUFFIX
<code>System.Collections.Generic.IDictionary< TKey, TValue ></code>	Dictionary
<code>System.Data.DataSet</code>	DataSet
<code>System.Data.DataTable</code>	Collection or DataTable
<code>System.IO.Stream</code>	Stream
<code>System.Security.IPermission</code>	Permission
<code>System.Security.Policy.IMembershipCondition</code>	Condition
An event-handler delegate.	EventHandler

Types that implement [ICollection](#) and are a generalized type of data structure, such as a dictionary, stack, or queue, are allowed names that provide meaningful information about the intended usage of the type.

Types that implement [ICollection](#) and are a collection of specific items have names that end with the word 'Collection'. For example, a collection of [Queue](#) objects would have the name 'QueueCollection'. The 'Collection' suffix signifies that the members of the collection can be enumerated by using the `foreach` ([For Each](#) in Visual Basic) statement.

Types that implement [IDictionary](#) or [IReadOnlyDictionary< TKey, TValue >](#) have names that end with the word 'Dictionary' even if the type also implements [IEnumerable](#) or [ICollection](#). The 'Collection' and 'Dictionary' suffix naming conventions enable users to distinguish between the following two enumeration patterns.

Types with the 'Collection' suffix follow this enumeration pattern.

```
foreach(SomeType x in SomeCollection) { }
```

Types with the 'Dictionary' suffix follow this enumeration pattern.

```
foreach(SomeType x in SomeDictionary.Values) { }
```

A [DataSet](#) object consists of a collection of [DataTable](#) objects, which consist of collections of [System.Data DataColumn](#) and [System.Data DataRow](#) objects, among others. These collections implement [ICollection](#) through the base [System.Data.InternalDataCollectionBase](#) class.

How to fix violations

Rename the type so that it is suffixed with the correct term.

When to suppress warnings

It is safe to suppress a warning to use the 'Collection' suffix if the type is a generalized data structure that might be extended or that will hold an arbitrary set of diverse items. In this case, a name that provides meaningful information about the implementation, performance, or other characteristics of the data structure might make sense (for example, [BinaryTree](#)). In cases where the type represents a collection of a specific type (for example, [StringCollection](#)), do not suppress a warning from this rule because the suffix indicates that the type can be enumerated by using a `foreach` statement.

For other suffixes, do not suppress a warning from this rule. The suffix allows the intended usage to be evident from the type name.

SUPPRESS A WARNING

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1710
// The code that's violating the rule is on this line.
#pragma warning restore CA1710
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1710.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Naming.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

CONFIGURE CODE TO ANALYZE

Use the following options to configure which parts of your codebase to run this rule on.

- [Include specific API surfaces](#)
- [Exclude indirect base types](#)
- [Additional required suffixes](#)

You can configure these options for just this rule, for all rules, or for all rules in this category ([Naming](#)). For more information, see [Code quality rule configuration options](#).

INCLUDE SPECIFIC API SURFACES

You can configure which parts of your codebase to run this rule on, based on their accessibility. For example, to specify that the rule should run only against the non-public API surface, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.api_surface = private, internal
```

EXCLUDE INDIRECT BASE TYPES

You can configure whether to exclude indirect base types from the rule. By default, this option is set to true, which restricts analysis to the current base type.

```
dotnet_code_quality.CA1710.exclude_indirect_base_types = false
```

ADDITIONAL REQUIRED SUFFIXES

You can provide additional required suffixes or override the behavior of some hardcoded suffixes by adding the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CA1710.additional_required_suffixes = [type]->[suffix]
```

Separate multiple values with a `|` character. Types can be specified in either of the following formats:

- Type name only (includes all types with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#) with an optional `T:` prefix.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CA1710.additional_required_suffixes = MyClass->Class</code>	All types that inherit from 'MyClass' are required to have the 'Class' suffix.
<code>dotnet_code_quality.CA1710.additional_required_suffixes = MyClass->Class MyNamespace.IPath->Path</code>	All types that inherit from 'MyClass' are required to have the 'Class' suffix AND all types that implement 'MyNamespace.IPath' are required to have the 'Path' suffix.
<code>dotnet_code_quality.CA1710.additional_required_suffixes = T:System.Data.IDataReader->{}</code>	Overrides built-in suffixes. In this case, all types that implement 'IDataReader' are no longer required to end in 'Collection'.

Related rules

[CA1711: Identifiers should not have incorrect suffix](#)

See also

- [Attributes](#)
- [Handling and raising events](#)

CA1711: Identifiers should not have incorrect suffix

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA1711
Category	Naming
Fix is breaking or non-breaking	Breaking

Cause

An identifier has an incorrect suffix.

By default, this rule only looks at externally visible identifiers, but this is [configurable](#).

Rule description

By convention, only the names of types that extend certain base types or that implement certain interfaces, or types derived from these types, should end with specific reserved suffixes. Other type names should not use these reserved suffixes.

The following table lists the reserved suffixes and the base types and interfaces with which they are associated.

SUFFIX	BASE TYPE/INTERFACE
Attribute	System.Attribute
Collection	System.Collections.ICollection System.Collections.IEnumerable System.Collections.Queue System.Collections.Stack System.Collections.Generic.ICollection<T> System.Data.DataSet System.Data.DataTable
Dictionary	System.Collections.IDictionary System.Collections.Generic.IDictionary< TKey, TValue >
EventArgs	System.EventArgs
EventHandler	An event-handler delegate
Exception	System.Exception

SUFFIX	BASE TYPE/INTERFACE
Permission	System.Security.IPermission
Queue	System.Collections.Queue
Stack	System.Collections.Stack
Stream	System.IO.Stream

In addition, the following suffixes should **not** be used:

- `Delegate`
- `Enum`
- `Impl` (use `Core` instead)
- `Ex` or similar suffix to distinguish it from an earlier version of the same type
- `Flag` or `Flags` for enum types

Naming conventions provide a common look for libraries that target the common language runtime. This reduces the learning curve that is required for new software libraries, and increases customer confidence that the library was developed by someone who has expertise in developing managed code. For more information, see [Naming guidelines: Classes, Structs, and Interfaces](#).

How to fix violations

Remove the suffix from the type name.

When to suppress warnings

Do not suppress a warning from this rule unless the suffix has an unambiguous meaning in the application domain.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1711
// The code that's violating the rule is on this line.
#pragma warning restore CA1711
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1711.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Naming.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

Use the following options to configure which parts of your codebase to run this rule on.

- [Include specific API surfaces](#)
- [Allow suffixes](#)

You can configure these options for just this rule, for all rules, or for all rules in this category ([Naming](#)). For more information, see [Code quality rule configuration options](#).

Include specific API surfaces

You can configure which parts of your codebase to run this rule on, based on their accessibility. For example, to specify that the rule should run only against the non-public API surface, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.api_surface = private, internal
```

Allow suffixes

You can configure a list of allowed suffixes, with each suffix separated by the pipe character ("|"). For example, to specify that the rule should not run against Flag and Flags suffixes, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.ca1711.allowed_suffixes = Flag|Flags
```

Related rules

- [CA1710: Identifiers should have correct suffix](#)

See also

- [Attributes](#)
- [Handling and raising events](#)
- [Naming guidelines: Classes, Structs, and Interfaces](#)

CA1712: Do not prefix enum values with type name

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA1712
Category	Naming
Fix is breaking or non-breaking	Breaking

Cause

An enumeration contains a member whose name starts with the type name of the enumeration.

Rule description

Names of enumeration members are not prefixed with the type name because type information is expected to be provided by development tools.

Naming conventions provide a common look for libraries that target the common language runtime. This reduces the time that is required for to learn a new software library, and increases customer confidence that the library was developed by someone who has expertise in developing managed code.

How to fix violations

To fix a violation of this rule, remove the type name prefix from the enumeration member.

When to suppress warnings

Do not suppress a warning from this rule.

Example

The following example shows an incorrectly named enumeration followed by the corrected version.

```
public enum DigitalImageMode
{
    DigitalImageModeBitmap = 0,
    DigitalImageModeGrayscale = 1,
    DigitalImageModeIndexed = 2,
    DigitalImageModeRGB = 3
}

public enum DigitalImageMode2
{
    Bitmap = 0,
    Grayscale = 1,
    Indexed = 2,
    RGB = 3
}
```

```

Imports System

Namespace ca1712

    Enum DigitalImageMode

        DigitalImageModeBitmap = 0
        DigitalImageModeGrayscale = 1
        DigitalImageModeIndexed = 2
        DigitalImageModeRGB = 3

    End Enum

    Enum DigitalImageMode2

        Bitmap = 0
        Grayscale = 1
        Indexed = 2
        RGB = 3

    End Enum

End Namespace

```

Related rules

- [CA1711: Identifiers should not have incorrect suffix](#)
- [CA1027: Mark enums with FlagsAttribute](#)
- [CA2217: Do not mark enums with FlagsAttribute](#)

Configure code to analyze

Use the following option to configure which parts of your codebase to run this rule on.

- [Enum values prefix trigger](#)

You can configure this option for just this rule, for all rules, or for all rules in this category ([Naming](#)). For more information, see [Code quality rule configuration options](#).

Enum values prefix trigger

You can configure the number of enumeration values required to trigger the rule. For example, to specify that the rule is triggered if one or more the enum values starts with the enum type name, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CA1712.enum_values_prefix_trigger = AnyEnumValue
```

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CA1712.enum_values_prefix_trigger = AnyEnumValue</code>	The rule is triggered if <i>any</i> of the enum values starts with the enum type name.
<code>dotnet_code_quality.CA1712.enum_values_prefix_trigger = AllEnumValues</code>	The rule is triggered if <i>all</i> of the enum values start with the enum type name.

OPTION VALUE	SUMMARY
<pre>dotnet_code_quality.CA1712.enum_values_prefix_trigger = Heuristic</pre>	The rule is triggered using the default heuristic, that is, when at least 75% of the enum values start with the enum type name.

See also

- [System.Enum](#)

CA1713: Events should not have before or after prefix

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA1713
Category	Naming
Fix is breaking or non-breaking	Breaking

Cause

The name of an event starts with 'Before' or 'After'.

Rule description

Event names should describe the action that raises the event. To name related events that are raised in a specific sequence, use the present or past tense to indicate the relative position in the sequence of actions. For example, when naming a pair of events that is raised when closing a resource, you might name it 'Closing' and 'Closed', instead of 'BeforeClose' and 'AfterClose'.

Naming conventions provide a common look for libraries that target the common language runtime. This reduces the learning curve that is required for new software libraries, and increases customer confidence that the library was developed by someone who has expertise in developing managed code.

How to fix violations

Remove the prefix from the event name, and consider changing the name to use the present or past tense of a verb.

When to suppress warnings

Do not suppress a warning from this rule.

CA1714: Flags enums should have plural names

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA1714
Category	Naming
Fix is breaking or non-breaking	Breaking

Cause

An enumeration has the [System.FlagsAttribute](#) and its name does not end in 's'.

By default, this rule only looks at externally visible enumerations, but this is [configurable](#).

Rule description

Types that are marked with [FlagsAttribute](#) have names that are plural because the attribute indicates that more than one value can be specified. For example, an enumeration that defines the days of the week might be intended for use in an application where you can specify multiple days. This enumeration should have the [FlagsAttribute](#) and could be called 'Days'. A similar enumeration that allows only a single day to be specified would not have the attribute, and could be called 'Day'.

Naming conventions provide a common look for libraries that target the common language runtime. This reduces the learning curve that is required for new software libraries, and increases customer confidence that the library was developed by someone who has expertise in developing managed code.

How to fix violations

Make the name of the enumeration a plural word, or remove the [FlagsAttribute](#) attribute if multiple enumeration values should not be specified simultaneously.

When to suppress warnings

It is safe to suppress a violation if the name is a plural word but does not end in 's'. For example, if the multiple-day enumeration that was described previously were named 'DaysOfTheWeek', this would violate the logic of the rule but not its intent. Such violations should be suppressed.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1714
// The code that's violating the rule is on this line.
#pragma warning restore CA1714
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1714.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Naming.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

Use the following option to configure which parts of your codebase to run this rule on.

- [Include specific API surfaces](#)

You can configure this option for just this rule, for all rules, or for all rules in this category ([Naming](#)). For more information, see [Code quality rule configuration options](#).

Include specific API surfaces

You can configure which parts of your codebase to run this rule on, based on their accessibility. For example, to specify that the rule should run only against the non-public API surface, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXX.api_surface = private, internal
```

Related rules

- [CA1027: Mark enums with FlagsAttribute](#)
- [CA2217: Do not mark enums with FlagsAttribute](#)

See also

- [System.FlagsAttribute](#)
- [Enum design](#)

CA1715: Identifiers should have correct prefix

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA1715
Category	Naming
Fix is breaking or non-breaking	Breaking - when fired on interfaces. Non-breaking - when raised on generic type parameters.

Cause

The name of an interface does not start with an uppercase 'I'.

-or-

The name of a [generic type parameter](#) on a type or method does not start with an uppercase 'T'.

By default, this rule only looks at externally visible interfaces, types, and methods, but this is [configurable](#).

Rule description

By convention, the names of certain programming elements start with a specific prefix.

Interface names should start with an uppercase 'I' followed by another uppercase letter. This rule reports violations for interface names such as 'MyInterface' and 'IsolatedInterface'.

Generic type parameter names should start with an uppercase 'T' and optionally may be followed by another uppercase letter. This rule reports violations for generic type parameter names such as 'V' and 'Type'.

Naming conventions provide a common look for libraries that target the common language runtime. This reduces the learning curve that is required for new software libraries, and increases customer confidence that the library was developed by someone who has expertise in developing managed code.

Configure code to analyze

Use the following options to configure which parts of your codebase to run this rule on.

- [Include specific API surfaces](#)
- [Single-character type parameters](#)

You can configure these options for just this rule, for all rules, or for all rules in this category ([Naming](#)). For more information, see [Code quality rule configuration options](#).

Include specific API surfaces

You can configure which parts of your codebase to run this rule on, based on their accessibility. For example, to specify that the rule should run only against the non-public API surface, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXX.api_surface = private, internal
```

Single-character type parameters

You can configure whether or not to exclude single-character type parameters from this rule. For example, to specify that this rule *should not* analyze single-character type parameters, add one of the following key-value pairs to an .editorconfig file in your project:

```
# Package version 2.9.0 and later  
dotnet_code_quality.CA1715.exclude_single_letter_type_parameters = true  
  
# Package version 2.6.3 and earlier  
dotnet_code_quality.CA2007.allow_single_letter_type_parameters = true
```

NOTE

This rule never fires for a type parameter named `T`, for example, `Collection<T>`.

How to fix violations

Rename the identifier so that it is correctly prefixed.

When to suppress warnings

Do not suppress a warning from this rule.

Interface naming example

The following code snippet shows an incorrectly named interface:

```
' Violates this rule  
Public Interface Book  
  
    ReadOnly Property Title() As String  
  
    Sub Read()  
  
End Interface
```

```
// Violation.  
public interface Book  
{  
    string Title  
    {  
        get;  
    }  
  
    void Read();  
}
```

The following code snippet fixes the previous violation by prefixing the interface with 'I':

```
// Fixes the violation by prefixing the interface with 'I'.
public interface IBook
{
    string Title
    {
        get;
    }

    void Read();
}
```

```
' Fixes the violation by prefixing the interface with 'I'
Public Interface IBook

    ReadOnly Property Title() As String

    Sub Read()

End Interface
```

Type parameter naming example

The following code snippet shows an incorrectly named generic type parameter:

```
' Violates this rule
Public Class Collection(Of Item)

End Class
```

```
// Violation.
public class Collection<Item>
{
```

The following code snippet fixes the previous violation by prefixing the generic type parameter with 'T':

```
// Fixes the violation by prefixing the generic type parameter with 'T'.
public class Collection<TItem>
{
```

```
' Fixes the violation by prefixing the generic type parameter with 'T'
Public Class Collection(Of TItem)

End Class
```

CA1716: Identifiers should not match keywords

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA1716
Category	Naming
Fix is breaking or non-breaking	Breaking

Cause

The name of a namespace, type, or virtual or interface member matches a reserved keyword in a programming language.

By default, this rule only looks at externally visible namespaces, types, and members, but you can [configure visibility](#) and [symbol kinds](#).

Rule description

Identifiers for namespaces, types, and virtual and interface members should not match keywords that are defined by languages that target the common language runtime. Depending on the language that is used and the keyword, compiler errors and ambiguities can make the library difficult to use.

This rule checks against keywords in the following languages:

- Visual Basic
- C#
- C++/CLI

Case-insensitive comparison is used for Visual Basic keywords, and case-sensitive comparison is used for the other languages.

How to fix violations

Select a name that does not appear in the list of keywords.

When to suppress warnings

You can suppress a warning from this rule if you're sure that the identifier won't confuse users of the API, and that the library is usable in all available languages in .NET.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1716
// The code that's violating the rule is on this line.
#pragma warning restore CA1716
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1716.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Naming.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

Use the following options to configure which parts of your codebase to run this rule on.

- [Include specific API surfaces](#)
- [Analyzed symbol kinds](#)

You can configure these options for just this rule, for all rules, or for all rules in this category ([Naming](#)). For more information, see [Code quality rule configuration options](#).

Include specific API surfaces

You can configure which parts of your codebase to run this rule on, based on their accessibility. For example, to specify that the rule should run only against the non-public API surface, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.api_surface = private, internal
```

Analyzed symbol kinds

You can configure the kinds of symbols that will be analyzed by this rule. The allowable values are:

- `Namespace`
- `NamedType`
- `Method`
- `Property`
- `Event`
- `Parameter`

Separate multiple values with a comma (`,`). The default value includes all of the symbol kinds in the previous list.

```
dotnet_code_quality.CA1716.analyzed_symbol_kinds = Namespace, NamedType, Method, Property, Event
```

CA1717: Only FlagsAttribute enums should have plural names

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA1717
Category	Naming
Fix is breaking or non-breaking	Breaking

Cause

The name of an enumeration ends in a plural word and the enumeration is not marked with the [System.FlagsAttribute](#) attribute.

By default, this rule only looks at externally visible enumerations, but this is [configurable](#).

Rule description

Naming conventions dictate that a plural name for an enumeration indicates that more than one value of the enumeration can be specified simultaneously. The [FlagsAttribute](#) tells compilers that the enumeration should be treated as a bit field that enables bitwise operations on the enumeration.

If only one value of an enumeration can be specified at a time, the name of the enumeration should be a singular word. For example, an enumeration that defines the days of the week might be intended for use in an application where you can specify multiple days. This enumeration should have the [FlagsAttribute](#) and could be called 'Days'. A similar enumeration that allows only a single day to be specified would not have the attribute, and could be called 'Day'.

Naming conventions provide a common look for libraries that target the common language runtime. This reduces the time that is required to learn a new software library, and increases customer confidence that the library was developed by someone who has expertise in developing managed code.

How to fix violations

Make the name of the enumeration a singular word or add the [FlagsAttribute](#).

When to suppress warnings

It is safe to suppress a warning from the rule if the name ends in a singular word.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1717
// The code that's violating the rule is on this line.
#pragma warning restore CA1717
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1717.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Naming.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

Use the following option to configure which parts of your codebase to run this rule on.

- [Include specific API surfaces](#)

You can configure this option for just this rule, for all rules, or for all rules in this category ([Naming](#)). For more information, see [Code quality rule configuration options](#).

Include specific API surfaces

You can configure which parts of your codebase to run this rule on, based on their accessibility. For example, to specify that the rule should run only against the non-public API surface, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.api_surface = private, internal
```

Related rules

- [CA1714: Flags enums should have plural names](#)
- [CA1027: Mark enums with FlagsAttribute](#)
- [CA2217: Do not mark enums with FlagsAttribute](#)

See also

- [System.FlagsAttribute](#)
- [Enum design](#)

CA1720: Identifiers should not contain type names

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA1720
Category	Naming
Fix is breaking or non-breaking	Breaking

Cause

The name of a parameter in a member contains a data type name.

-or-

The name of a member contains a language-specific data type name.

By default, this rule only looks at externally visible members, but this is [configurable](#).

Rule description

Names of parameters and members are better used to communicate their meaning than to describe their type, which is expected to be provided by development tools. For names of members, if a data type name must be used, use a language-independent name instead of a language-specific one. For example, instead of the C# type name `int`, use the language-independent data type name, `Int32`.

Each discrete token in the name of the parameter or member is checked against the following language-specific data type names in a case-insensitive manner:

- Bool
- WChar
- Int8
- UInt8
- Short
- UShort
- Int
- UInt
- Integer
- UInteger
- Long
- ULong
- Unsigned
- Signed
- Float
- Float32
- Float64

In addition, the names of a parameter are also checked against the following language-independent data type names in a case-insensitive manner:

- Object
- Boolean
- Char
- String
- SByte
- Byte
- UByte
- Int16
- UInt16
- Int32
- UInt32
- Int64
- UInt64
- IntPtr
- Ptr
- Pointer
- UIntptr
- UPtr
- UPointer
- Single
- Double
- Decimal
- Guid

How to fix violations

If fired against a parameter:

Replace the data type identifier in the name of the parameter with either a term that better describes its meaning or a more generic term, such as 'value'.

If fired against a member:

Replace the language-specific data type identifier in the name of the member with a term that better describes its meaning, a language-independent equivalent, or a more generic term, such as 'value'.

When to suppress warnings

Occasional use of type-based parameter and member names might be appropriate. However, for new development, no known scenarios occur where you should suppress a warning from this rule. For libraries that have previously shipped, you might have to suppress a warning from this rule.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1720
// The code that's violating the rule is on this line.
#pragma warning restore CA1720
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1720.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Naming.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

Use the following option to configure which parts of your codebase to run this rule on.

- [Include specific API surfaces](#)

You can configure this option for just this rule, for all rules, or for all rules in this category ([Naming](#)). For more information, see [Code quality rule configuration options](#).

Include specific API surfaces

You can configure which parts of your codebase to run this rule on, based on their accessibility. For example, to specify that the rule should run only against the non-public API surface, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.api_surface = private, internal
```

Related rules

- [CA1708: Identifiers should differ by more than case](#)
- [CA1707: Identifiers should not contain underscores](#)

CA1721: Property names should not match get methods

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA1721
Category	Naming
Fix is breaking or non-breaking	Breaking

Cause

The name of a member starts with 'Get' and otherwise matches the name of a property. For example, a type that contains a method that's named 'GetColor' and a property that's named 'Color' cause a rule violation. This rule will not fire if either the property or the method is marked with the [ObsoleteAttribute](#).

By default, this rule only looks at externally visible members and properties, but this is [configurable](#).

Rule description

"Get" methods and properties should have names that clearly distinguish their function.

Naming conventions provide a common look for libraries that target the common language runtime. This consistency reduces the time that's required to learn a new software library and increases customer confidence that the library was developed by someone who has expertise in developing managed code.

How to fix violations

Change the name so that it does not match the name of a method that is prefixed with 'Get'.

When to suppress warnings

Do not suppress a warning from this rule. One exception to that rule is if the "Get" method is caused by implementing the [IExtenderProvider](#) interface.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1721
// The code that's violating the rule is on this line.
#pragma warning restore CA1721
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1721.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Naming.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

Use the following option to configure which parts of your codebase to run this rule on.

- [Include specific API surfaces](#)

You can configure this option for just this rule, for all rules, or for all rules in this category ([Naming](#)). For more information, see [Code quality rule configuration options](#).

Include specific API surfaces

You can configure which parts of your codebase to run this rule on, based on their accessibility. For example, to specify that the rule should run only against the non-public API surface, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXX.api_surface = private, internal
```

Example

The following example contains a method and property that violate this rule.

```
public class Test
{
    public DateTime Date
    {
        get { return DateTime.Today; }
    }

    // Violates rule: PropertyNamesShouldNotMatchGetMethods.
    public string GetDate()
    {
        return this.Date.ToString();
    }
}
```

```
Imports System

Namespace ca1721

    Public Class Test

        Public ReadOnly Property [Date]() As DateTime
            Get
                Return DateTime.Today
            End Get
        End Property

        ' Violates rule: PropertyNamesShouldNotMatchGetMethods.
        Public Function GetDate() As String
            Return Me.Date.ToString()
        End Function

    End Class

End Namespace
```

Related rules

- [CA1024: Use properties where appropriate](#)

CA1724: Type names should not match namespaces

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA1724
Category	Naming
Fix is breaking or non-breaking	Breaking

Cause

A type name matches a referenced namespace name that has one or more externally visible types. The name comparison is case-insensitive.

Rule description

User-created type names should not match the names of referenced namespaces that have externally visible types. Violating this rule can reduce the usability of your library.

How to fix violations

Rename the type such that it doesn't match the name of a referenced namespace that has externally visible types.

When to suppress warnings

For new development, no known scenarios occur where you must suppress a warning from this rule. Before you suppress the warning, carefully consider how the users of your library might be confused by the matching name. For shipping libraries, you might have to suppress a warning from this rule.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1724
// The code that's violating the rule is on this line.
#pragma warning restore CA1724
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1724.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]\ndotnet_analyzer_diagnostic.category-Naming.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Example

```
namespace MyNamespace\n{\n    // This class violates the rule\n    public class System\n    {\n    }\n}
```

CA1725: Parameter names should match base declaration

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA1725
Category	Naming
Fix is breaking or non-breaking	Breaking

Cause

The name of a parameter in a method override does not match the name of the parameter in the base declaration of the method or the name of the parameter in the interface declaration of the method.

By default, this rule only looks at externally visible methods, but this is [configurable](#).

Rule description

Consistent naming of parameters in an override hierarchy increases the usability of the method overrides. A parameter name in a derived method that differs from the name in the base declaration can cause confusion about whether the method is an override of the base method or a new overload of the method.

How to fix violations

To fix a violation of this rule, rename the parameter to match the base declaration. The fix is a breaking change for COM visible methods.

When to suppress warnings

Do not suppress a warning from this rule except for COM visible methods in libraries that have previously shipped.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1725
// The code that's violating the rule is on this line.
#pragma warning restore CA1725
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1725.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Naming.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

Use the following option to configure which parts of your codebase to run this rule on.

- [Include specific API surfaces](#)

You can configure this option for just this rule, for all rules, or for all rules in this category ([Naming](#)). For more information, see [Code quality rule configuration options](#).

Include specific API surfaces

You can configure which parts of your codebase to run this rule on, based on their accessibility. For example, to specify that the rule should run only against the non-public API surface, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXX.api_surface = private, internal
```

CA1727: Use PascalCase for named placeholders

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA1727
Category	Naming
Fix is breaking or non-breaking	Non-breaking

Cause

A named placeholder used with [ILogger](#) is not PascalCase.

Rule description

A named placeholder used with [ILogger](#) should be PascalCase, a naming convention where the first letter of each compound word in a name is capitalized. This naming convention is recommended for structured logging, where each named placeholder is used as a property name in the structured data.

How to fix violations

Use PascalCase for named placeholders. For example, change `{firstName}` to `{FirstName}`.

When to suppress warnings

It is safe to suppress a warning from this rule.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1727
// The code that's violating the rule is on this line.
#pragma warning restore CA1727
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1727.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Naming.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Performance rules

9/20/2022 • 7 minutes to read • [Edit Online](#)

Performance rules support high-performance libraries and applications.

In this section

RULE	DESCRIPTION
CA1802: Use Literals Where Appropriate	A field is declared static and read-only (Shared and ReadOnly in Visual Basic), and is initialized with a value that is computable at compile time. Because the value that is assigned to the targeted field is computable at compile time, change the declaration to a const (Const in Visual Basic) field so that the value is computed at compile time instead of at run time.
CA1805: Do not initialize unnecessarily	The .NET runtime initializes all fields of reference types to their default values before running the constructor. In most cases, explicitly initializing a field to its default value is redundant, which adds to maintenance costs and may degrade performance (such as with increased assembly size).
CA1806: Do not ignore method results	A new object is created but never used, or a method that creates and returns a new string is called and the new string is never used, or a Component Object Model (COM) or P/Invoke method returns an HRESULT or error code that is never used.
CA1810: Initialize reference type static fields inline	When a type declares an explicit static constructor, the just-in-time (JIT) compiler adds a check to each static method and instance constructor of the type to make sure that the static constructor was previously called. Static constructor checks can decrease performance.
CA1812: Avoid uninstantiated internal classes	An instance of an assembly-level type is not created by code in the assembly.
CA1813: Avoid unsealed attributes	.NET provides methods for retrieving custom attributes. By default, these methods search the attribute inheritance hierarchy. Sealing the attribute eliminates the search through the inheritance hierarchy and can improve performance.
CA1814: Prefer jagged arrays over multidimensional	A jagged array is an array whose elements are arrays. The arrays that make up the elements can be of different sizes, which can result in less wasted space for some sets of data.
CA1815: Override equals and operator equals on value types	For value types, the inherited implementation of Equals uses the Reflection library and compares the contents of all fields. Reflection is computationally expensive, and comparing every field for equality might be unnecessary. If you expect users to compare or sort instances, or to use instances as hash table keys, your value type should implement Equals.

Rule	Description
CA1819: Properties should not return arrays	Arrays that are returned by properties are not write-protected, even if the property is read-only. To keep the array tamper-proof, the property must return a copy of the array. Typically, users will not understand the adverse performance implications of calling such a property.
CA1820: Test for empty strings using string length	Comparing strings by using the <code>String.Length</code> property or the <code>String.IsNullOrEmpty</code> method is significantly faster than using <code>Equals</code> .
CA1821: Remove empty finalizers	Whenever you can, avoid finalizers because of the additional performance overhead that is involved in tracking object lifetime. An empty finalizer incurs added overhead without any benefit.
CA1822: Mark members as static	Members that do not access instance data or call instance methods can be marked as static (Shared in Visual Basic). After you mark the methods as static, the compiler will emit nonvirtual call sites to these members. This can give you a measurable performance gain for performance-sensitive code.
CA1823: Avoid unused private fields	Private fields were detected that do not appear to be accessed in the assembly.
CA1824: Mark assemblies with <code>NeutralResourcesLanguageAttribute</code>	The <code>NeutralResourcesLanguage</code> attribute informs the Resource Manager of the language that was used to display the resources of a neutral culture for an assembly. This improves lookup performance for the first resource that you load and can reduce your working set.
CA1825: Avoid zero-length array allocations	Initializing a zero-length array leads to unnecessary memory allocation. Instead, use the statically allocated empty array instance by calling <code>Array.Empty</code> . The memory allocation is shared across all invocations of this method.
CA1826: Use property instead of Linq Enumerable method	<code>Enumerable</code> LINQ method was used on a type that supports an equivalent, more efficient property.
CA1827: Do not use Count/LongCount when Any can be used	<code>Count</code> or <code>LongCount</code> method was used where <code>Any</code> method would be more efficient.
CA1828: Do not use CountAsync/LongCountAsync when AnyAsync can be used	<code>CountAsync</code> or <code>LongCountAsync</code> method was used where <code>AnyAsync</code> method would be more efficient.
CA1829: Use Length/Count property instead of <code>Enumerable.Count</code> method	<code>Count</code> LINQ method was used on a type that supports an equivalent, more efficient <code>Length</code> or <code>Count</code> property.
CA1830: Prefer strongly-typed Append and Insert method overloads on <code>StringBuilder</code>	<code>Append</code> and <code>Insert</code> provide overloads for multiple types beyond <code>System.String</code> . When possible, prefer the strongly-typed overloads over using <code>ToString()</code> and the string-based overload.

RULE	DESCRIPTION
CA1831: Use AsSpan instead of Range-based indexers for string when appropriate	When using a range-indexer on a string and implicitly assigning the value to a <code>ReadOnlySpan<char></code> type, the method <code>Substring</code> will be used instead of <code>Slice</code> , which produces a copy of requested portion of the string.
CA1832: Use AsSpan or AsMemory instead of Range-based indexers for getting <code>ReadOnlySpan</code> or <code>ReadOnlyMemory</code> portion of an array	When using a range-indexer on an array and implicitly assigning the value to a <code>ReadOnlySpan<T></code> or <code>ReadOnlyMemory<T></code> type, the method <code>GetSubArray</code> will be used instead of <code>Slice</code> , which produces a copy of requested portion of the array.
CA1833: Use AsSpan or AsMemory instead of Range-based indexers for getting <code>Span</code> or <code>Memory</code> portion of an array	When using a range-indexer on an array and implicitly assigning the value to a <code>Span<T></code> or <code>Memory<T></code> type, the method <code>GetSubArray</code> will be used instead of <code>Slice</code> , which produces a copy of requested portion of the array.
CA1834: Use <code>StringBuilder.Append(char)</code> for single character strings	<code>StringBuilder</code> has an <code>Append</code> overload that takes a <code>char</code> as its argument. Prefer calling the <code>char</code> overload to improve performance.
CA1835: Prefer the 'Memory'-based overloads for 'ReadAsync' and 'WriteAsync'	'Stream' has a 'ReadAsync' overload that takes a 'Memory<Byte>' as the first argument, and a 'WriteAsync' overload that takes a 'ReadOnlyMemory<Byte>' as the first argument. Prefer calling the memory based overloads, which are more efficient.
CA1836: Prefer <code>IsEmpty</code> over <code>Count</code> when available	Prefer <code>IsEmpty</code> property that is more efficient than <code>Count</code> , <code>Length</code> , <code>Count<TSource>(IEnumerable<TSource>)</code> or <code>LongCount<TSource>(IEnumerable<TSource>)</code> to determine whether the object contains or not any items.
CA1837: Use <code>Environment.ProcessId</code> instead of <code>Process.GetCurrentProcess().Id</code>	<code>Environment.ProcessId</code> is simpler and faster than <code>Process.GetCurrentProcess().Id</code> .
CA1838: Avoid <code>StringBuilder</code> parameters for P/Invokes	Marshalling of <code>StringBuilder</code> always creates a native buffer copy, resulting in multiple allocations for one marshalling operation.
CA1839: Use <code>Environment.ProcessPath</code> instead of <code>Process.GetCurrentProcess().MainModule.FileName</code>	<code>Environment.ProcessPath</code> is simpler and faster than <code>Process.GetCurrentProcess().MainModule.FileName</code> .
CA1840: Use <code>Environment.CurrentManagedThreadId</code> instead of <code>Thread.CurrentThread.ManagedThreadId</code>	<code>Environment.CurrentManagedThreadId</code> is more compact and efficient than <code>Thread.CurrentThread.ManagedThreadId</code> .
CA1841: Prefer Dictionary Contains methods	Calling <code>Contains</code> on the <code>Keys</code> or <code>Values</code> collection may often be more expensive than calling <code>ContainsKey</code> or <code>ContainsValue</code> on the dictionary itself.
CA1842: Do not use 'WhenAll' with a single task	Using <code>WhenAll</code> with a single task may result in performance loss. Await or return the task instead.
CA1843: Do not use 'WaitAll' with a single task	Using <code>WaitAll</code> with a single task may result in performance loss. Await or return the task instead.

RULE	DESCRIPTION
CA1844: Provide memory-based overrides of async methods when subclassing 'Stream'	To improve performance, override the memory-based async methods when subclassing 'Stream'. Then implement the array-based methods in terms of the memory-based methods.
CA1845: Use span-based <code>string.Concat</code>	It is more efficient to use <code>AsSpan</code> and <code>string.Concat</code> , instead of <code>Substring</code> and a concatenation operator.
CA1846: Prefer <code>AsSpan</code> over <code>Substring</code>	<code>AsSpan</code> is more efficient than <code>Substring</code> . <code>Substring</code> performs an O(n) string copy, while <code>AsSpan</code> does not and has a constant cost. <code>AsSpan</code> also does not perform any heap allocations.
CA1847: Use char literal for a single character lookup	Use <code>string.Contains(char)</code> instead of <code>string.Contains(string)</code> when searching for a single character.
CA1848: Use the <code>LoggerMessage</code> delegates	For improved performance, use the <code>LoggerMessage</code> delegates.
CA1849: Call async methods when in an async method	In a method which is already asynchronous, calls to other methods should be to their async versions, where they exist.
CA1850: Prefer static <code>HashData</code> method over <code>ComputeHash</code>	It's more efficient to use the static <code>HashData</code> method over creating and managing a <code>HashAlgorithm</code> instance to call <code>ComputeHash</code> .
CA1851: Possible multiple enumerations of <code>IEnumerable</code> collection	Possible multiple enumerations of <code>IEnumerable</code> collection. Consider using an implementation that avoids multiple enumerations.
CA1854: Prefer the ' <code>IDictionary.TryGetValue(TKey, out TValue)</code> ' method	Prefer 'TryGetValue' over a Dictionary indexer access guarded by a 'ContainsKey' check. 'ContainsKey' and the indexer both look up the key, so using 'TryGetValue' avoids the extra lookup.

CA1802: Use Literals Where Appropriate

9/20/2022 • 3 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA1802
Category	Performance
Fix is breaking or non-breaking	Non-breaking

Cause

A field is declared `static` and `readonly` (`Shared` and `ReadOnly` in Visual Basic), and is initialized with a value that is computable at compile time.

By default, this rule only looks at externally visible, static, readonly fields, but this is [configurable](#).

Rule description

The value of a `static readonly` field is computed at run time when the static constructor for the declaring type is called. If the `static readonly` field is initialized when it is declared and a static constructor is not declared explicitly, the compiler emits a static constructor to initialize the field.

The value of a `const` field is computed at compile time and stored in the metadata, which increases run-time performance when it is compared to a `static readonly` field.

Because the value assigned to the targeted field is computable at compile time, change the declaration to a `const` field so that the value is computed at compile time instead of at run time.

How to fix violations

To fix a violation of this rule, replace the `static` and `readonly` modifiers with the `const` modifier.

NOTE

The use of the `const` modifier is not recommended for all scenarios.

When to suppress warnings

It is safe to suppress a warning from this rule, or disable the rule, if performance is not of concern.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1802
// The code that's violating the rule is on this line.
#pragma warning restore CA1802
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1802.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Performance.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

Use the following options to configure which parts of your codebase to run this rule on.

- [Include specific API surfaces](#)
- [Required modifiers](#)

You can configure these options for just this rule, for all rules, or for all rules in this category ([Performance](#)). For more information, see [Code quality rule configuration options](#).

Include specific API surfaces

You can configure which parts of your codebase to run this rule on, based on their accessibility. For example, to specify that the rule should run only against the non-public API surface, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXX.api_surface = private, internal
```

Required modifiers

You can configure this rule to override the required field modifiers. By default, `static` and `readonly` are both required modifiers for fields that are analyzed. You can override this to a comma separated list of one or more modifier values from the below table:

OPTION VALUE	SUMMARY
<code>none</code>	No modifier requirement.
<code>static</code> or <code>Shared</code>	Must be declared as 'static' ('Shared' in Visual Basic).
<code>const</code>	Must be declared as 'const'.
<code>readonly</code>	Must be declared as 'readonly'.

For example, to specify that the rule should run against both static and instance fields, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CA1802.required_modifiers = none
```

Example

The following example shows a type, `UseReadOnly`, that violates the rule and a type, `UseConstant`, that satisfies the rule.

```
Imports System

Namespace ca1802

    ' This class violates the rule.
    Public Class UseReadOnly

        Shared ReadOnly x As Integer = 3
        Shared ReadOnly y As Double = x + 2.1
        Shared ReadOnly s As String = "readonly"

    End Class

    ' This class satisfies the rule.
    Public Class UseConstant

        Const x As Integer = 3
        Const y As Double = x + 2.1
        Const s As String = "const"

    End Class

End Namespace
```

```
// This class violates the rule.
public class UseReadOnly
{
    static readonly int x = 3;
    static readonly double y = x + 2.1;
    static readonly string s = "readonly";

    public void Print()
    {
        Console.WriteLine(s);
    }
}

// This class satisfies the rule.
public class UseConstant
{
    const int x = 3;
    const double y = x + 2.1;
    const string s = "const";
}
```

CA1805: Do not initialize unnecessarily.

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA1805
Category	Performance
Fix is breaking or non-breaking	Non-breaking

Cause

A field of a class is explicitly initialized to the default value of that field's type.

Rule description

The .NET runtime initializes all fields of reference types to their default values before running the constructor. In most cases, explicitly initializing a field to its default value in a constructor is redundant, adding maintenance costs and potentially degrading performance (such as with increased assembly size), and the explicit initialization can be removed.

How to fix violations

In most cases, the proper fix is to delete the unnecessary initialization.

```
class C
{
    // Violation
    int _value1 = 0;

    // Fixed
    int _value1;
}
```

In some cases, deleting the initialization may result in subsequent [CS0649](#) warnings being issued due to the field retaining its default value forever. In such cases, a better fix may be to delete the field entirely or replace it with a property:

```
class C
{
    // Violation
    private static readonly int s_value = 0;

    // Fixed
    private static int Value => 0;
}
```

When to suppress warnings

It is always safe to suppress the warning, as the warning simply highlights potentially unnecessary code and

work that may be avoided.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1805
// The code that's violating the rule is on this line.
#pragma warning restore CA1805
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1805.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Performance.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Performance rules](#)

CA1806: Do not ignore method results

9/20/2022 • 4 minutes to read • [Edit Online](#)

	Value
Rule ID	CA1806
Category	Performance
Fix is breaking or non-breaking	Non-breaking

Cause

There are several possible reasons for this warning:

- A new object is created but never used.
- A method that creates and returns a new string is called and the new string is never used.
- A COM or P/Invoke method that returns a `HRESULT` or error code that's never used.
- A language-integrated query (LINQ) method that returns a result that's never used.

Rule description

Unnecessary object creation and the associated garbage collection of the unused object degrade performance.

Strings are immutable and methods such as `String.ToUpper` return a new instance of a string instead of modifying the instance of the string in the calling method.

Ignoring `HRESULT` or an error code can lead to low-resource conditions or unexpected behavior in error conditions.

LINQ methods are known to not have side effects, and the result should not be ignored.

How to fix violations

If a method creates a new instance of an object that's never used, pass the instance as an argument to another method or assign the instance to a variable. If the object creation is unnecessary, remove it.

-or-

If method A calls method B but does not use the new string instance that method B returns, pass the instance as an argument to another method or assign the instance to a variable. Or remove the call if it's unnecessary.

-or-

If method A calls method B but does not use the `HRESULT` or error code that the method returns, use the result in a conditional statement, assign the result to a variable, or pass it as an argument to another method.

-or-

If a LINQ method A calls method B but does not use the result, use the result in a conditional statement, assign the result to a variable, or pass it as an argument to another method.

When to suppress warnings

Do not suppress a warning from this rule unless the act of creating the object serves some purpose.

SUPPRESS A WARNING

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1806
// The code that's violating the rule is on this line.
#pragma warning restore CA1806
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1806.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Performance.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

CONFIGURE CODE TO ANALYZE

Use the following option to configure which parts of your codebase to run this rule on.

Additional methods to enforce

You can configure this rule to check that results from additional custom APIs are used. Specify one or more methods as the *value* of the `additional_use_results_methods` option. To specify multiple method names, separate them with `|`. The allowable formats for the method name are:

- Method name only (which will include all methods with that name, regardless of their containing type or namespace).
- Fully qualified name in the [documentation ID format](#), with an optional `M:` prefix.

For example, to specify that rule CA1806 should also check that the result from a method named `MyMethod1` is used, add the following key-value pair to an `.editorconfig` file in your project.

```
dotnet_code_quality.CA1806.additional_use_results_methods = MyMethod1
```

Or, use the fully qualified name to disambiguate or ensure that only a specific method with that name is included.

```
dotnet_code_quality.CA1806.additional_use_results_methods = M:MyNamespace.MyType.MyMethod1(ParamType)
```

EXAMPLE 1

The following example shows a class that ignores the result of calling `String.Trim`.

```

public class Book
{
    private readonly string _Title;

    public Book(string title)
    {
        if (title != null)
        {
            // Violates this rule
            title.Trim();
        }

        _Title = title;
    }

    public string Title
    {
        get { return _Title; }
    }
}

```

```

Public Class Book
    Public Sub New(ByVal title As String)

        If title IsNot Nothing Then
            ' Violates this rule
            title.Trim()
        End If

        Me.Title = title

    End Sub

    Public ReadOnly Property Title() As String

End Class

```

Example 2

The following example fixes the [Example 1](#) violation by assigning the result of `String.Trim` back to the variable it was called on.

```

public class Book
{
    private readonly string _Title;

    public Book(string title)
    {
        if (title != null)
        {
            title = title.Trim();
        }

        _Title = title;
    }

    public string Title
    {
        get { return _Title; }
    }
}

```

```

Public Class Book
    Public Sub New(ByVal title As String)

        If title IsNot Nothing Then
            title = title.Trim()
        End If

        Me.Title = title

    End Sub

    Public ReadOnly Property Title() As String

End Class

```

Example 3

The following example shows a method that does not use an object that it creates.

NOTE

This violation cannot be reproduced in Visual Basic.

```

public class Book
{
    public Book()
    {

    }

    public static Book CreateBook()
    {
        // Violates this rule
        new Book();
        return new Book();
    }
}

```

Example 4

The following example fixes the [Example 3](#) violation by removing the unnecessary creation of an object.

```

public class Book
{
    public Book()
    {

    }

    public static Book CreateBook()
    {
        return new Book();
    }
}

```

CA1810: Initialize reference type static fields inline

9/20/2022 • 3 minutes to read • [Edit Online](#)

	Value
Rule ID	CA1810
Category	Performance
Fix is breaking or non-breaking	Non-breaking

Cause

A reference type declares an explicit static constructor.

Rule description

When a type declares an explicit static constructor, the just-in-time (JIT) compiler adds a check to each static method and instance constructor of the type to make sure that the static constructor was previously called. Static initialization is triggered when any static member is accessed or when an instance of the type is created. However, static initialization is not triggered if you declare a variable of the type but do not use it, which can be important if the initialization changes global state.

When all static data is initialized inline and an explicit static constructor is not declared, Microsoft intermediate language (MSIL) compilers add the `beforefieldinit` flag and an implicit static constructor, which initializes the static data, to the MSIL type definition. When the JIT compiler encounters the `beforefieldinit` flag, most of the time the static constructor checks are not added. Static initialization is guaranteed to occur at some time before any static fields are accessed but not before a static method or instance constructor is invoked. Note that static initialization can occur at any time after a variable of the type is declared.

Static constructor checks can decrease performance. Often a static constructor is used only to initialize static fields, in which case you must only make sure that static initialization occurs before the first access of a static field. The `beforefieldinit` behavior is appropriate for these and most other types. It is only inappropriate when static initialization affects global state and one of the following is true:

- The effect on global state is expensive and is not required if the type is not used.
- The global state effects can be accessed without accessing any static fields of the type.

How to fix violations

To fix a violation of this rule, initialize all static data when it is declared and remove the static constructor.

When to suppress warnings

It is safe to suppress a warning from this rule if one of the following applies:

- Performance is not a concern.
- Global state changes that are caused by static initialization are expensive or must be guaranteed to occur before a static method of the type is called or an instance of the type is created.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1810
// The code that's violating the rule is on this line.
#pragma warning restore CA1810
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.cs,vb]
dotnet_diagnostic.CA1810.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.cs,vb]
dotnet_analyzer_diagnostic.category-Performance.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Example

The following example shows a type, `StaticConstructor`, that violates the rule and a type, `NoStaticConstructor`, that replaces the static constructor with inline initialization to satisfy the rule.

```
public class StaticConstructor
{
    static int someInteger;
    static string resourceString;

    static StaticConstructor()
    {
        someInteger = 3;
        ResourceManager stringManager =
            new ResourceManager("strings", Assembly.GetExecutingAssembly());
        resourceString = stringManager.GetString("string");
    }

    public void Print()
    {
        Console.WriteLine(someInteger);
    }
}

public class NoStaticConstructor
{
    static int someInteger = 3;
    static string resourceString = InitializeResourceString();

    static string InitializeResourceString()
    {
        ResourceManager stringManager =
            new ResourceManager("strings", Assembly.GetExecutingAssembly());
        return stringManager.GetString("string");
    }

    public void Print()
    {
        Console.WriteLine(someInteger);
    }
}
```

```

Imports System
Imports System.Resources

Namespace ca1810

    Public Class StaticConstructor

        Shared someInteger As Integer
        Shared resourceString As String

        Shared Sub New()

            someInteger = 3
            Dim stringManager As New ResourceManager("strings",
                System.Reflection.Assembly.GetExecutingAssembly())
            resourceString = stringManager.GetString("string")

        End Sub

    End Class

    Public Class NoStaticConstructor

        Shared someInteger As Integer = 3
        Shared resourceString As String = InitializeResourceString()

        Private Shared Function InitializeResourceString()

            Dim stringManager As New ResourceManager("strings",
                System.Reflection.Assembly.GetExecutingAssembly())
            Return stringManager.GetString("string")

        End Function

    End Class

End Namespace

```

Note the addition of the `beforefieldinit` flag on the MSIL definition for the `NoStaticConstructor` class.

```

.class public auto ansi StaticConstructor
extends [mscorlib]System.Object
{
} // end of class StaticConstructor

.class public auto ansi beforefieldinit NoStaticConstructor
extends [mscorlib]System.Object
{
} // end of class NoStaticConstructor

```

Related rules

- [CA2207: Initialize value type static fields inline](#)

CA1812: Avoid uninstantiated internal classes

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA1812
Category	Performance
Fix is breaking or non-breaking	Non-breaking

Cause

An internal (assembly-level) type is never instantiated.

Rule description

This rule tries to locate a call to one of the constructors of the type and reports a violation if no call is found.

The following types are not examined by this rule:

- Value types
- Abstract types
- Enumerations
- Delegates
- Compiler-emitted array types
- Types that cannot be instantiated and that only define `static` (`Shared` in Visual Basic) methods.

If you apply the `System.Runtime.CompilerServices.InternalsVisibleToAttribute` to the assembly that's being analyzed, this rule will not flag types that are marked as `internal` (`Friend` in Visual Basic) because a field may be used by a friend assembly.

How to fix violations

To fix a violation of this rule, remove the type or add code that uses it. If the type contains only `static` methods, add one of the following to the type to prevent the compiler from emitting a default public instance constructor:

- The `static` modifier for C# types that target .NET Framework 2.0 or later.
- A private constructor for types that target .NET Framework versions 1.0 and 1.1.

When to suppress warnings

It is safe to suppress a warning from this rule. We recommend that you suppress this warning in the following situations:

- The class is created through late-bound reflection methods such as `System.Activator.CreateInstance`.
- The class is registered in an inversion of control (IoC) container as part of the `dependency injection`

pattern.

- The class is created automatically by the runtime or ASP.NET. Some examples of automatically created classes are those that implement [System.Configuration.IConfigurationSectionHandler](#) or [System.Web.IHttpHandler](#).
- The class is used as a type parameter in a class definition and has a [new](#) constraint. The following example will be flagged by rule CA1812:

```
internal class MyClass
{
    public void DoSomething()
    {
    }
}
public class MyGeneric<T> where T : new()
{
    public T Create()
    {
        return new T();
    }
}

MyGeneric<MyClass> mc = new MyGeneric<MyClass>();
mc.Create();
```

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1812
// The code that's violating the rule is on this line.
#pragma warning restore CA1812
```

To disable the rule for a file, folder, or project, set its severity to [none](#) in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1812.severity = none
```

To disable this entire category of rules, set the severity for the category to [none](#) in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Performance.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Related rules

- [CA1801: Review unused parameters](#)

CA1813: Avoid unsealed attributes

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA1813
Category	Performance
Fix is breaking or non-breaking	Breaking

Cause

A public type inherits from [System.Attribute](#), is not abstract, and is not sealed (`NotInheritable` in Visual Basic).

Rule description

.NET provides methods for retrieving custom attributes. By default, these methods search the attribute inheritance hierarchy. For example, [System.Attribute.GetCustomAttribute](#) searches for the specified attribute type or any attribute type that extends the specified attribute type. Sealing the attribute eliminates the search through the inheritance hierarchy, and can improve performance.

How to fix violations

To fix a violation of this rule, seal the attribute type or make it abstract.

When to suppress warnings

It is safe to suppress a warning from this rule. Suppress only if you are defining an attribute hierarchy and cannot seal the attribute or make it abstract.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1813
// The code that's violating the rule is on this line.
#pragma warning restore CA1813
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.cs,vb]
dotnet_diagnostic.CA1813.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Performance.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Example

The following example shows a custom attribute that satisfies this rule.

```
// Satisfies rule: AvoidUnsealedAttributes.
[AttributeUsage(AttributeTargets.Class | AttributeTargets.Struct)]
public sealed class DeveloperAttribute : Attribute
{
    private string nameValue;
    public DeveloperAttribute(string name)
    {
        nameValue = name;
    }

    public string Name
    {
        get
        {
            return nameValue;
        }
    }
}
```

```
Imports System

Namespace ca1813

    ' Satisfies rule: AvoidUnsealedAttributes.
    <AttributeUsage(AttributeTargets.Class Or AttributeTargets.Struct)>
    Public NotInheritable Class DeveloperAttribute
        Inherits Attribute

        Public Sub New(name As String)
            Me.Name = name
        End Sub

        Public ReadOnly Property Name() As String
    End Class

End Namespace
```

Related rules

- [CA1019: Define accessors for attribute arguments](#)
- [CA1018: Mark attributes with AttributeUsageAttribute](#)

See also

- [Attributes](#)

CA1814: Prefer jagged arrays over multidimensional

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA1814
Category	Performance
Fix is breaking or non-breaking	Breaking

Cause

A member is declared as a multidimensional array, which can result in wasted space for some data sets.

Rule description

In a [multidimensional array](#), each element in each dimension has the same, fixed size as the other elements in that dimension. In a [jagged array](#), which is an array of arrays, each inner array can be of a different size. By only using the space that's needed for a given array, no space is wasted. This rule, CA1814, recommends switching to a jagged array to conserve memory.

How to fix violations

To fix a violation of this rule, change the multidimensional array to a jagged array.

When to suppress warnings

It's okay to suppress a warning from this rule if the multidimensional array does not waste space.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1814
// The code that's violating the rule is on this line.
#pragma warning restore CA1814
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1814.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Performance.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Example

The following example shows declarations for jagged and multidimensional arrays.

```
Imports System

Public Class ArrayHolder
    Private jaggedArray As Integer()() = {New Integer() {1, 2, 3, 4}, _
                                           New Integer() {5, 6, 7}, _
                                           New Integer() {8}, _
                                           New Integer() {9}}

    Private multiDimArray As Integer(,) = {{1, 2, 3, 4}, _
                                            {5, 6, 7, 0}, _
                                            {8, 0, 0, 0}, _
                                            {9, 0, 0, 0}}
End Class
```

```
public class ArrayHolder
{
    int[][] jaggedArray = { new int[] {1,2,3,4},
                           new int[] {5,6,7},
                           new int[] {8},
                           new int[] {9}
                         };

    int[,] multiDimArray = {{1,2,3,4},
                           {5,6,7,0},
                           {8,0,0,0},
                           {9,0,0,0}
                         };
}
```

CA1815: Override equals and operator equals on value types

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA1815
Category	Performance
Fix is breaking or non-breaking	Non-breaking

Cause

A value type does not override [System.Object.Equals](#) or does not implement the equality operator (==). This rule does not check enumerations.

By default, this rule only looks at externally visible types, but this is [configurable](#).

Rule description

For [non-blittable](#) value types, the inherited implementation of [Equals](#) uses the [System.Reflection](#) library to compare the contents of all fields. Reflection is computationally expensive, and comparing every field for equality might be unnecessary. If you expect users to compare or sort instances, or use them as hash table keys, your value type should implement [Equals](#). If your programming language supports operator overloading, you should also provide an implementation of the equality and inequality operators.

How to fix violations

To fix a violation of this rule, provide an implementation of [Equals](#). If you can, implement the equality operator.

When to suppress warnings

It's safe to suppress a warning from this rule if instances of the value type will not be compared to each other.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1815
// The code that's violating the rule is on this line.
#pragma warning restore CA1815
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1815.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Performance.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

Use the following option to configure which parts of your codebase to run this rule on.

- [Include specific API surfaces](#)

You can configure this option for just this rule, for all rules, or for all rules in this category ([Performance](#)). For more information, see [Code quality rule configuration options](#).

Include specific API surfaces

You can configure which parts of your codebase to run this rule on, based on their accessibility. For example, to specify that the rule should run only against the non-public API surface, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXX.api_surface = private, internal
```

Example

The following code shows a structure (value type) that violates this rule:

```
// Violates this rule
public struct Point
{
    public Point(int x, int y)
    {
        X = x;
        Y = y;
    }

    public int X { get; }

    public int Y { get; }
}
```

The following code fixes the previous violation by overriding `System.ValueType.Equals` and implementing the equality operators (`==` and `!=`):

```

public struct Point : IEquatable<Point>
{
    public Point(int x, int y)
    {
        X = x;
        Y = y;
    }

    public int X { get; }

    public int Y { get; }

    public override int GetHashCode()
    {
        return X ^ Y;
    }

    public override bool Equals(object obj)
    {
        if (!(obj is Point))
            return false;

        return Equals((Point)obj);
    }

    public bool Equals(Point other)
    {
        if (X != other.X)
            return false;

        return Y == other.Y;
    }

    public static bool operator ==(Point point1, Point point2)
    {
        return point1.Equals(point2);
    }

    public static bool operator !=(Point point1, Point point2)
    {
        return !point1.Equals(point2);
    }
}

```

Related rules

- [CA2231: Overload operator equals on overriding ValueType.Equals](#)
- [CA2226: Operators should have symmetrical overloads](#)

See also

- [System.Object.Equals](#)

CA1819: Properties should not return arrays

9/20/2022 • 4 minutes to read • [Edit Online](#)

	Value
Rule ID	CA1819
Category	Performance
Fix is breaking or non-breaking	Breaking

Cause

A property returns an array.

By default, this rule only looks at externally visible properties and types, but this is [configurable](#).

Rule description

Arrays returned by properties are not write-protected, even if the property is read-only. To keep the array tamper-proof, the property must return a copy of the array. Typically, users won't understand the adverse performance implications of calling such a property. Specifically, they might use the property as an indexed property.

How to fix violations

To fix a violation of this rule, either make the property a method or change the property to return a collection.

When to suppress warnings

You can suppress a warning that's raised for a property of an attribute that's derived from the [Attribute](#) class. Attributes can contain properties that return arrays, but can't contain properties that return collections.

You can suppress the warning if the property is part of a [Data Transfer Object \(DTO\)](#) class.

Otherwise, do not suppress a warning from this rule.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1819
// The code that's violating the rule is on this line.
#pragma warning restore CA1819
```

To disable the rule for a file, folder, or project, set its severity to [none](#) in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1819.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Performance.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

Use the following option to configure which parts of your codebase to run this rule on.

- [Include specific API surfaces](#)

You can configure this option for just this rule, for all rules, or for all rules in this category ([Performance](#)). For more information, see [Code quality rule configuration options](#).

Include specific API surfaces

You can configure which parts of your codebase to run this rule on, based on their accessibility. For example, to specify that the rule should run only against the non-public API surface, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXX.api_surface = private, internal
```

Example violation

The following example shows a property that violates this rule:

```
public class Book
{
    private string[] _Pages;

    public Book(string[] pages)
    {
        _Pages = pages;
    }

    public string[] Pages
    {
        get { return _Pages; }
    }
}
```

```
Public Class Book
    Public Sub New(ByVal pages As String())
        Me.Pages = pages
    End Sub

    Public ReadOnly Property Pages() As String()
End Class
```

To fix a violation of this rule, either make the property a method or change the property to return a collection instead of an array.

Change the property to a method

The following example fixes the violation by changing the property to a method:

```
Public Class Book

    Private _Pages As String()

    Public Sub New(ByVal pages As String())
        _Pages = pages
    End Sub

    Public Function GetPages() As String()
        ' Need to return a clone of the array so that consumers
        ' of this library cannot change its contents
        Return DirectCast(_Pages.Clone(), String())
    End Function

End Class
```

```
public class Book
{
    private string[] _Pages;

    public Book(string[] pages)
    {
        _Pages = pages;
    }

    public string[] GetPages()
    {
        // Need to return a clone of the array so that consumers
        // of this library cannot change its contents
        return (string[])_Pages.Clone();
    }
}
```

Change the property to return a collection

The following example fixes the violation by changing the property to return a [System.Collections.ObjectModel.ReadOnlyCollection<T>](#):

```
public class Book
{
    private ReadOnlyCollection<string> _Pages;
    public Book(string[] pages)
    {
        _Pages = new ReadOnlyCollection<string>(pages);
    }

    public ReadOnlyCollection<string> Pages
    {
        get { return _Pages; }
    }
}
```

```

Public Class Book
    Public Sub New(ByVal pages As String())
        Me.Pages = New ReadOnlyCollection(Of String)(pages)
    End Sub

    Public ReadOnly Property Pages() As ReadOnlyCollection(Of String)

End Class

```

Allow users to modify a property

You might want to allow the consumer of the class to modify a property. The following example shows a read/write property that violates this rule:

```

public class Book
{
    private string[] _Pages;

    public Book(string[] pages)
    {
        _Pages = pages;
    }

    public string[] Pages
    {
        get { return _Pages; }
        set { _Pages = value; }
    }
}

```

```

Public Class Book
    Public Sub New(ByVal pages As String())
        Me.Pages = pages
    End Sub

    Public Property Pages() As String()

End Class

```

The following example fixes the violation by changing the property to return a [System.Collections.ObjectModel.Collection<T>](#):

```

Public Class Book
    Public Sub New(ByVal pages As String())
        Me.Pages = New Collection(Of String)(pages)
    End Sub

    Public ReadOnly Property Pages() As Collection(Of String)
End Class

```

```
public class Book
{
    private Collection<string> _Pages;

    public Book(string[] pages)
    {
        _Pages = new Collection<string>(pages);
    }

    public Collection<string> Pages
    {
        get { return _Pages; }
    }
}
```

Related rules

- [CA1024: Use properties where appropriate](#)

CA1820: Test for empty strings using string length

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA1820
Category	Performance
Fix is breaking or non-breaking	Non-breaking

Cause

A string is compared to the empty string by using `Object.Equals`.

Rule description

Comparing strings using the `String.Length` property or the `String.IsNullOrEmpty` method is faster than using `Equals`. This is because `Equals` executes significantly more MSIL instructions than either `IsNullOrEmpty` or the number of instructions executed to retrieve the `Length` property value and compare it to zero.

For null strings, `Equals` and `<string>.Length == 0` behave differently. If you try to get the value of the `Length` property on a null string, the common language runtime throws a `System.NullReferenceException`. If you perform a comparison between a null string and the empty string, the common language runtime does not throw an exception and returns `false`. Testing for null does not significantly affect the relative performance of these two approaches. When targeting .NET Framework 2.0 or later, use the `IsNullOrEmpty` method. Otherwise, use the `Length == 0` comparison whenever possible.

How to fix violations

To fix a violation of this rule, change the comparison to use the `IsNullOrEmpty` method.

When to suppress warnings

It's safe to suppress a warning from this rule if performance is not an issue.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1820
// The code that's violating the rule is on this line.
#pragma warning restore CA1820
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1820.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Performance.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Example

The following example illustrates the different techniques that are used to look for an empty string.

```
public class StringTester
{
    string s1 = "test";

    public void EqualsTest()
    {
        // Violates rule: TestForEmptyStringsUsingStringLength.
        if (s1 == "")
        {
            Console.WriteLine("s1 equals empty string.");
        }
    }

    // Use for .NET Framework 1.0 and 1.1.
    public void LengthTest()
    {
        // Satisfies rule: TestForEmptyStringsUsingStringLength.
        if (s1 != null && s1.Length == 0)
        {
            Console.WriteLine("s1.Length == 0.");
        }
    }

    // Use for .NET Framework 2.0.
    public void NullOrEmptyTest()
    {
        // Satisfies rule: TestForEmptyStringsUsingStringLength.
        if (!String.IsNullOrEmpty(s1))
        {
            Console.WriteLine("s1 != null and s1.Length != 0.");
        }
    }
}
```

CA1821: Remove empty finalizers

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA1821
Category	Performance
Fix is breaking or non-breaking	Non-breaking

Cause

A type implements a finalizer that is empty, calls only the base type finalizer, or calls only conditionally emitted methods.

Rule description

Whenever you can, avoid finalizers because of the additional performance overhead that's involved in tracking object lifetime. The garbage collector runs the finalizer before it collects the object. This means that at least two collections are required to collect the object. An empty finalizer incurs this added overhead without any benefit.

How to fix violations

Remove the empty finalizer. If a finalizer is required for debugging, enclose the whole finalizer in

`#if DEBUG / #endif` directives.

When to suppress warnings

Do not suppress a message from this rule.

Example

The following example shows an empty finalizer that should be removed, a finalizer that should be enclosed in

`#if DEBUG / #endif` directives, and a finalizer that uses the `#if DEBUG / #endif` directives correctly.

```
public class Class1
{
    // Violation occurs because the finalizer is empty.
    ~Class1()
    {
    }
}

public class Class2
{
    // Violation occurs because Debug.Fail is a conditional method.
    // The finalizer will contain code only if the DEBUG directive
    // symbol is present at compile time. When the DEBUG
    // directive is not present, the finalizer will still exist, but
    // it will be empty.
    ~Class2()
    {
        Debug.Fail("Finalizer called!");
    }
}

public class Class3
{
#if DEBUG
    // Violation will not occur because the finalizer will exist and
    // contain code when the DEBUG directive is present. When the
    // DEBUG directive is not present, the finalizer will not exist,
    // and therefore not be empty.
    ~Class3()
    {
        Debug.Fail("Finalizer called!");
    }
#endif
}
```

CA1822: Mark members as static

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA1822
Category	Performance
Fix is breaking or non-breaking	<p>Non-breaking - If the member is not visible outside the assembly, regardless of the change you make.</p> <p>Non-breaking - If you just change the member to an instance member with the <code>this</code> keyword.</p> <p>Breaking - If you change the member from an instance member to a static member and it is visible outside the assembly.</p>

Cause

A member that does not access instance data is not marked as static (Shared in Visual Basic).

Rule description

Members that do not access instance data or call instance methods can be marked as static (Shared in Visual Basic). After you mark the methods as static, the compiler will emit nonvirtual call sites to these members. Emitting nonvirtual call sites will prevent a check at run time for each call that makes sure that the current object pointer is non-null. This can achieve a measurable performance gain for performance-sensitive code. In some cases, the failure to access the current object instance represents a correctness issue.

How to fix violations

Mark the member as static (or Shared in Visual Basic) or use 'this'/'Me' in the method body, if appropriate.

When to suppress warnings

It is safe to suppress a warning from this rule for previously shipped code for which the fix would be a breaking change.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1822
// The code that's violating the rule is on this line.
#pragma warning restore CA1822
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1822.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Performance.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

Use the following option to configure which parts of your codebase to run this rule on.

- [Include specific API surfaces](#)

You can configure this option for just this rule, for all rules, or for all rules in this category ([Performance](#)). For more information, see [Code quality rule configuration options](#).

Include specific API surfaces

You can configure which parts of your codebase to run this rule on, based on their accessibility. For example, to specify that the rule should run only against the non-public API surface, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXX.api_surface = private, internal
```

Related rules

- [CA1812: Avoid uninstantiated internal classes](#)

CA1823: Avoid unused private fields

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA1823
Category	Performance
Fix is breaking or non-breaking	Non-breaking

Cause

This rule is reported when a private field exists in your code but is not used by any code path.

Rule description

Private fields were detected that do not appear to be accessed in the assembly.

How to fix violations

To fix a violation of this rule, remove the field or add code that uses it.

When to suppress warnings

It is safe to suppress a warning from this rule.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1823
// The code that's violating the rule is on this line.
#pragma warning restore CA1823
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1823.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Performance.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Related rules

- [CA1812: Avoid uninstantiated internal classes](#)
- [CA1801: Review unused parameters](#)

CA1824: Mark assemblies with NeutralResourcesLanguageAttribute

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA1824
Category	Performance
Fix is breaking or non-breaking	Non-breaking

Cause

An assembly contains a ResX-based resource but does not have the [System.Resources.NeutralResourcesLanguageAttribute](#) applied to it.

Rule description

The [NeutralResourcesLanguageAttribute](#) attribute informs the resource manager of an app's default culture. If the default culture's resources are embedded in the app's main assembly, and [ResourceManager](#) has to retrieve resources that belong to the same culture as the default culture, the [ResourceManager](#) automatically uses the resources located in the main assembly instead of searching for a satellite assembly. This bypasses the usual assembly probe, improves lookup performance for the first resource you load, and can reduce your working set.

TIP

See [Package and deploy resources](#) for the process that [ResourceManager](#) uses to probe for resource files.

Fix violations

To fix a violation of this rule, add the attribute to the assembly, and specify the language of the resources of the neutral culture.

To specify the neutral language for resources

1. In [Solution Explorer](#), right-click your project, and then select [Properties](#).
2. Select the [Package](#) tab.

NOTE

If your project is a .NET Framework project, select the [Application](#) tab, and then select [Assembly Information](#).

3. Select the language from the [Neutral language](#) or [Assembly neutral language](#) drop-down list.
4. Select [OK](#).

When to suppress warnings

It's permissible to suppress a warning from this rule. However, startup performance might degrade. To suppress this warning, add `dotnet_diagnostic.CA1824.severity = none` to your `.globalconfig` or `.editorconfig` file.

SUPPRESS A WARNING

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1824
// The code that's violating the rule is on this line.
#pragma warning restore CA1824
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1824.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Performance.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

SEE ALSO

- [NeutralResourcesLanguageAttribute](#)
- [Resources in .NET apps](#)

CA1825: Avoid zero-length array allocations

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA1825
Category	Performance
Fix is breaking or non-breaking	Non-breaking

Cause

An empty [Array](#) with no elements is allocated.

Rule description

Initializing a zero-length array leads to an unnecessary memory allocation. Instead, use the statically allocated empty array instance by calling the [Array.Empty](#) method. The memory allocation is shared across all invocations of this method.

How to fix violations

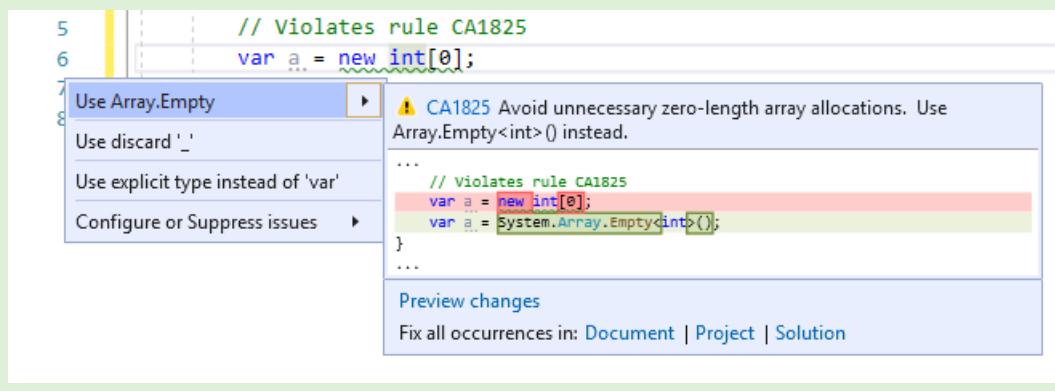
To fix a violation, replace the zero-length array allocation with a call to [Array.Empty](#). For example, the following two code snippets show a violation of the rule and how to fix it:

```
class C
{
    public void M1()
    {
        // Violates rule CA1825.
        var a = new int[0];
    }
}
```

```
class C
{
    public void M1()
    {
        // Resolves rule CA1825 violation.
        var a = System.Array.Empty<int>();
    }
}
```

TIP

A code fix is available for this rule in Visual Studio. To use it, position the cursor on the array allocation and press **Ctrl+.** (period). Choose **Use Array.Empty** from the list of options that's presented.



When to suppress warnings

It's safe to suppress a violation of this rule if you're not concerned about the additional memory allocation.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1825
// The code that's violating the rule is on this line.
#pragma warning restore CA1825
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1825.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Performance.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Related rules

- [CA1814: Prefer jagged arrays over multidimensional](#)

See also

- [Performance rules](#)

CA1826: Use property instead of Linq Enumerable method

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA1826
Category	Performance
Fix is breaking or non-breaking	Non-breaking

Cause

The [Enumerable](#) LINQ method was used on a type that supports an equivalent, more efficient property.

Rule description

This rule flags the [Enumerable](#) LINQ method calls on collections of types that have equivalent, but more efficient properties to fetch the same data.

This rule analyzes the following collection types:

- A type that implements [IReadOnlyList<T>](#), but not [IList<T>](#)

This rule flags calls to following methods on these collection types:

- [System.Linq.Enumerable.Count](#) method
- [System.Linq.Enumerable.First](#) method
- [System.Linq.Enumerable.FirstOrDefault](#) method
- [System.Linq.Enumerable.Last](#) method
- [System.Linq.Enumerable.LastOrDefault](#) method

The analyzed collection types and/or methods may be extended in future to cover more cases.

How to fix violations

To fix a violation, replace the [Enumerable](#) method calls with property access. For example, the following two code snippets show a violation of the rule and how to fix it:

```
using System;
using System.Collections.Generic;
using System.Linq;

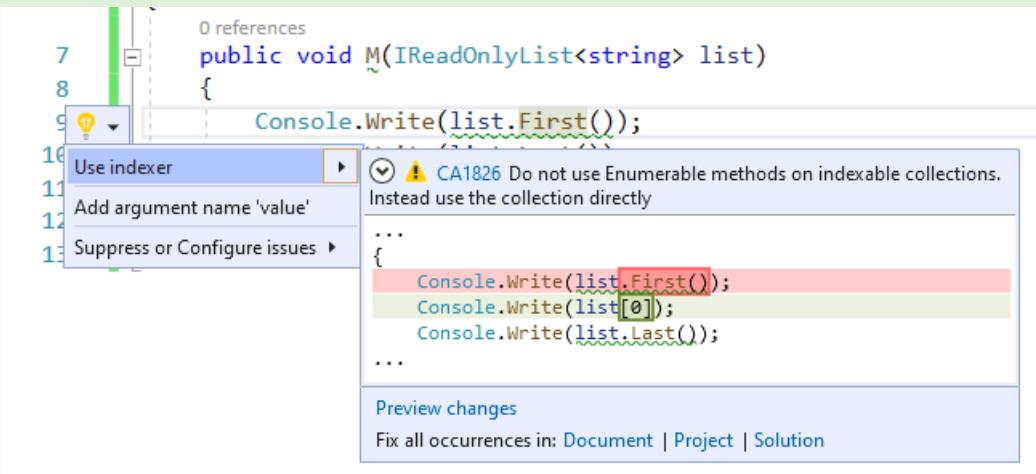
class C
{
    public void M(IReadOnlyList<string> list)
    {
        Console.WriteLine(list.First());
        Console.WriteLine(list.Last());
        Console.WriteLine(list.Count());
    }
}
```

```
using System;
using System.Collections.Generic;

class C
{
    public void M(IReadOnlyList<string> list)
    {
        Console.WriteLine(list[0]);
        Console.WriteLine(list[list.Count - 1]);
        Console.WriteLine(list.Count);
    }
}
```

TIP

A code fix is available for this rule in Visual Studio. To use it, position the cursor on the violation and press **Ctrl+.** (period). Choose **Use indexer** from the list of options that's presented.



When to suppress warnings

It's safe to suppress a violation of this rule if you're not concerned about the performance impact from specific [Enumerable](#) method calls.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1826
// The code that's violating the rule is on this line.
#pragma warning restore CA1826
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1826.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Performance.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Related rules

- [CA1827: Do not use Count/LongCount when Any can be used](#)
- [CA1828: Do not use CountAsync/LongCountAsync when AnyAsync can be used](#)
- [CA1829: Use Length/Count property instead of Enumerable.Count method](#)

See also

- [Performance rules](#)

CA1827: Do not use Count/LongCount when Any can be used

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA1827
Category	Performance
Fix is breaking or non-breaking	Non-breaking

Cause

The [Count](#) or [LongCount](#) method was used where [Any](#) method would be more efficient.

Rule description

This rule flags the [Count](#) and [LongCount](#) LINQ method calls used to check if the collection has at least one element. These method calls require enumerating the entire collection to compute the count. The same check is faster with the [Any](#) method as it avoids enumerating the collection.

How to fix violations

To fix a violation, replace the [Count](#) or [LongCount](#) method call with the [Any](#) method. For example, the following two code snippets show a violation of the rule and how to fix it:

```
using System.Collections.Generic;
using System.Linq;

class C
{
    public string M1(IEnumerable<string> list)
        => list.Count() != 0 ? "Not empty" : "Empty";

    public string M2(IEnumerable<string> list)
        => list.LongCount() > 0 ? "Not empty" : "Empty";
}
```

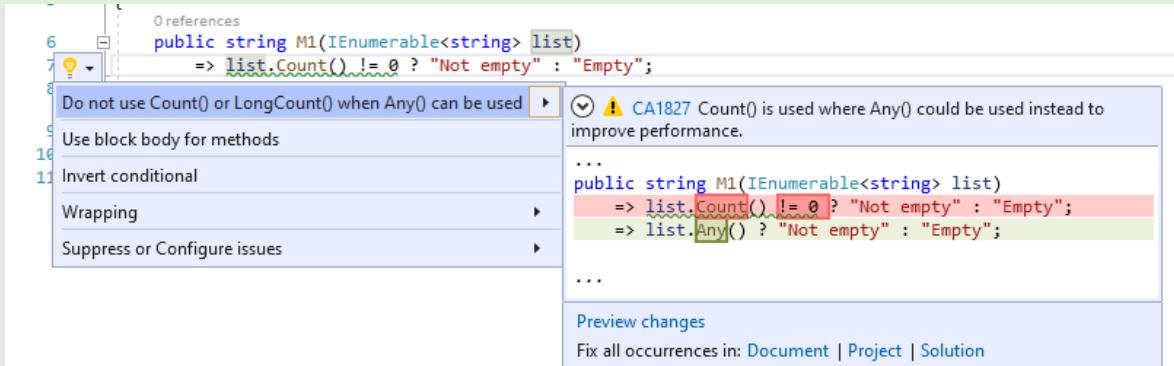
```
using System.Collections.Generic;
using System.Linq;

class C
{
    public string M1(IEnumerable<string> list)
        => list.Any() ? "Not empty" : "Empty";

    public string M2(IEnumerable<string> list)
        => list.Any() ? "Not empty" : "Empty";
}
```

TIP

A code fix is available for this rule in Visual Studio. To use it, position the cursor on the violation and press **Ctrl+.** (period). Choose **Do not use Count() or LongCount() when Any() can be used** from the list of options that's presented.



When to suppress warnings

It's safe to suppress a violation of this rule if you're not concerned about the performance impact from unnecessary collection enumeration to compute the count.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1827
// The code that's violating the rule is on this line.
#pragma warning restore CA1827
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1827.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Performance.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Related rules

- [CA1826: Use property instead of Linq Enumerable method](#)
- [CA1828: Do not use CountAsync/LongCountAsync when AnyAsync can be used](#)
- [CA1829: Use Length/Count property instead of Enumerable.Count method](#)

See also

- [Performance rules](#)

CA1828: Do not use CountAsync/LongCountAsync when AnyAsync can be used

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA1828
Category	Performance
Fix is breaking or non-breaking	Non-breaking

Cause

[CountAsync](#) or The [LongCountAsync](#) method was used where the [AnyAsync](#) method would be more efficient.

Rule description

This rule flags the [CountAsync](#) and [LongCountAsync](#) LINQ method calls used to check if the collection has at least one element. These method calls require enumerating the entire collection to compute the count. The same check is faster with the [AnyAsync](#) method as it avoids enumerating the collection.

How to fix violations

To fix a violation, replace the [CountAsync](#) or [LongCountAsync](#) method call with the [AnyAsync](#) method. For example, the following two code snippets show a violation of the rule and how to fix it:

```
using System.Linq;
using System.Threading.Tasks;
using static Microsoft.EntityFrameworkCore.EntityFrameworkQueryExtensions;

class C
{
    public async Task<string> M1Async(IQueryable<string> list)
        => await list.CountAsync() != 0 ? "Not empty" : "Empty";

    public async Task<string> M2Async(IQueryable<string> list)
        => await list.LongCountAsync() > 0 ? "Not empty" : "Empty";
}
```

```

using System.Linq;
using System.Threading.Tasks;
using static Microsoft.EntityFrameworkCore.EntityFrameworkQueryExtensions;

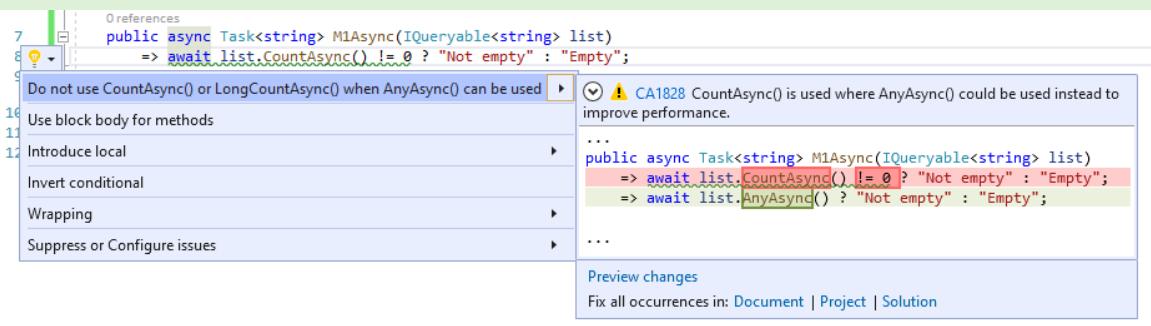
class C
{
    public async Task<string> M1Async(IQueryable<string> list)
        => await list.AnyAsync() ? "Not empty" : "Empty";

    public async Task<string> M2Async(IQueryable<string> list)
        => await list.AnyAsync() ? "Not empty" : "Empty";
}

```

TIP

A code fix is available for this rule in Visual Studio. To use it, position the cursor on the violation and press **Ctrl+.** (period). Choose **Do not use CountAsync() or LongCountAsync() when AnyAsync() can be used** from the list of options that's presented.



When to suppress warnings

It's safe to suppress a violation of this rule if you're not concerned about the performance impact from unnecessary collection enumeration to compute the count.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```

#pragma warning disable CA1828
// The code that's violating the rule is on this line.
#pragma warning restore CA1828

```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```

[*.{cs,vb}]
dotnet_diagnostic.CA1828.severity = none

```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```

[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Performance.severity = none

```

For more information, see [How to suppress code analysis warnings](#).

Related rules

- [CA1826: Use property instead of Linq Enumerable method](#)
- [CA1827: Do not use Count/LongCount when Any can be used](#)
- [CA1829: Use Length/Count property instead of Enumerable.Count method](#)

See also

- [Performance rules](#)

CA1829: Use Length/Count property instead of Enumerable.Count method

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA1829
Category	Performance
Fix is breaking or non-breaking	Non-breaking

Cause

The `Count` LINQ method was used on a type that supports an equivalent, more efficient `Length` or `Count` property.

Rule description

This rule flags the `Count` LINQ method calls on collections of types that have equivalent, but more efficient `Length` or `Count` property to fetch the same data. `Length` or `Count` property does not enumerate the collection, hence is more efficient.

This rule flags `Count` calls on the following collection types with `Length` property:

- `System.Array`
- `System.Collections.Immutable.ImmutableArray<T>`

This rule flags `Count` calls on the following collection types with the `Count` property:

- `System.Collections.ICollection`
- `System.Collections.Generic.ICollection<T>`
- `System.Collections.Generic.IReadOnlyCollection<T>`

The analyzed collection types may be extended in future to cover more cases.

How to fix violations

To fix a violation, replace the `Count` method call with use of the `Length` or `Count` property access. For example, the following two code snippets show a violation of the rule and how to fix it:

```

using System.Collections.Generic;
using System.Linq;

class C
{
    public int GetCount(int[] array)
        => array.Count();

    public int GetCount(ICollection<int> collection)
        => collection.Count();
}

```

```

using System.Collections.Generic;

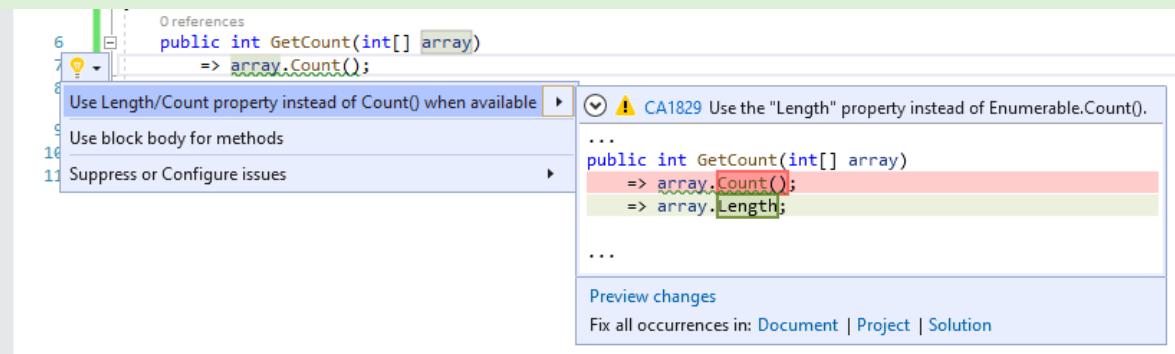
class C
{
    public int GetCount(int[] array)
        => array.Length;

    public int GetCount(ICollection<int> collection)
        => collection.Count();
}

```

TIP

A code fix is available for this rule in Visual Studio. To use it, position the cursor on the violation and press **Ctrl+.** (period). Choose **Use Length/Count property instead of Count() when available** from the list of options that's presented.



When to suppress warnings

It's safe to suppress a violation of this rule if you're not concerned about the performance impact from unnecessary collection enumeration to compute the count.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```

#pragma warning disable CA1829
// The code that's violating the rule is on this line.
#pragma warning restore CA1829

```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1829.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Performance.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Related rules

- [CA1826: Use property instead of Linq Enumerable method](#)
- [CA1827: Do not use Count/LongCount when Any can be used](#)
- [CA1828: Do not use CountAsync/LongCountAsync when AnyAsync can be used](#)

See also

- [Performance rules](#)

CA1830: Prefer strongly-typed Append and Insert method overloads on StringBuilder.

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA1830
Category	Performance
Fix is breaking or non-breaking	Non-breaking

Cause

An `StringBuilder` `Append` or `Insert` method was called with an argument that was the result of calling `ToString` on a type for which the `Append` or `Insert` method has a dedicated overload.

Rule description

`Append` and `Insert` provide overloads for multiple types beyond `String`. When possible, prefer the strongly-typed overloads over using `ToString()` and the string-based overload.

How to fix violations

Delete the unnecessary `ToString()` from the invocation.

```
using System.Text;

class C
{
    int _value;

    // Violation
    public void Log(StringBuilder destination)
    {
        destination.Append("Value: ").Append(_value.ToString()).AppendLine();
    }

    // Fixed
    public void Log(StringBuilder destination)
    {
        destination.Append("Value: ").Append(_value).AppendLine();
    }
}
```

When to suppress warnings

It's safe to suppress a violation of this rule if you're not concerned about the performance impact from unnecessary string allocations.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1830
// The code that's violating the rule is on this line.
#pragma warning restore CA1830
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.cs,vb]
dotnet_diagnostic.CA1830.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.cs,vb]
dotnet_analyzer_diagnostic.category-Performance.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Performance rules](#)

CA1831: Use AsSpan instead of Range-based indexers for string when appropriate

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA1831
Category	Performance
Fix is breaking or non-breaking	Non-breaking

Cause

A range-indexer is used on a string and the value is implicitly assigned to `ReadOnlySpan<char>`.

Rule description

This rule fires when you use a range-indexer on a string and assign it to a span type. The range indexer on a `Span<T>` is a non-copying `Slice` operation, but for the range indexer on a string, the method `Substring` will be used instead of `Slice`. This produces a copy of the requested portion of the string. This copy is usually unnecessary when it's implicitly used as a `ReadOnlySpan<T>` or `ReadOnlyMemory<T>` value. If a copy isn't intended, use the `AsSpan` method to avoid the unnecessary copy. If the copy is intended, either assign it to a local variable first or add an explicit cast. The analyzer only reports when an implicit cast is used on the result of the range indexer operation.

Detects

Implicit conversion:

```
ReadOnlySpan<char> slice = str[a..b];
```

Does not detect

Explicit conversion:

```
ReadOnlySpan<char> slice = (ReadOnlySpan<char>)str[a..b];
```

How to fix violations

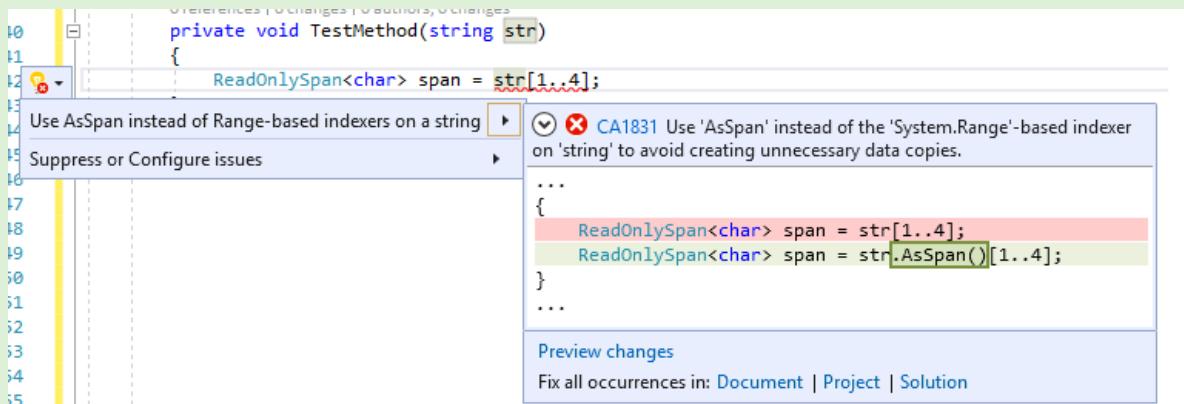
To fix a violation of this rule, use `AsSpan` instead of the `Range`-based indexer on the string to avoid creating unnecessary data copies.

```
public void TestMethod(string str)
{
    // The violation occurs
    ReadOnlySpan<char> slice = str[1..3];
    ...
}
```

```
public void TestMethod(string str)
{
    // The violation fixed with AsSpan extension method
    ReadOnlySpan<char> slice = str.AsSpan()[1..3];
    ...
}
```

TIP

A code fix is available for this rule in Visual Studio. To use it, position the cursor on the violation and press **Ctrl+.** (period). Choose **Use AsSpan instead of the Range-based indexer on a string** from the list of options that's presented.



You can also add an explicit cast to avoid this warning.

```
public void TestMethod(string str)
{
    // The violation occurs.
    ReadOnlySpan<char> slice = str[1..3];
    ...
}
```

```
public void TestMethod(string str)
{
    // The violation avoided with explicit casting.
    ReadOnlySpan<char> slice = (ReadOnlySpan<char>)str[1..3];
    ...
}
```

When to suppress warnings

It's safe to suppress a violation of this rule if creating a copy is intended.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1831
// The code that's violating the rule is on this line.
#pragma warning restore CA1831
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1831.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Performance.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Related rules

- [CA1832: Use AsSpan or AsMemory instead of Range-based indexers for getting ReadOnlySpan or ReadOnlyMemory portion of an array](#)
- [CA1833: Use AsSpan or AsMemory instead of Range-based indexers for getting Span or Memory portion of an array](#)

See also

- [Performance rules](#)

CA1832: Use AsSpan or AsMemory instead of Range-based indexers for getting ReadOnlySpan or ReadOnlyMemory portion of an array

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA1832
Category	Performance
Fix is breaking or non-breaking	Non-breaking

Cause

When using a range-indexer on an array and implicitly assigning the value to `ReadOnlySpan<T>` or `ReadOnlyMemory<T>`.

Rule description

Using a range-indexer on an array and assigning to a memory or span type: The range indexer on a `Span<T>` is a non-copying `Slice` operation, but for the range indexer on array the method `GetSubArray` will be used instead of `Slice`, which produces a copy of requested portion of the array. This copy is usually unnecessary when it is implicitly used as a `ReadOnlySpan<T>` or `ReadOnlyMemory<T>` value. If a copy isn't intended, use the `AsSpan` or `AsMemory` method to avoid the unnecessary copy. If the copy is intended, either assign it to a local variable first or add an explicit cast. The analyzer only reports when an implicit cast is used on the result of the range indexer operation.

Detects

Implicit conversions:

- `ReadOnlySpan<SomeT> slice = arr[a..b];`
- `ReadOnlyMemory<SomeT> slice = arr[a..b];`

Does not detect

Explicit conversions:

- `ReadOnlySpan<SomeT> slice = (ReadOnlySpan<SomeT>)arr[a..b];`
- `ReadOnlyMemory<SomeT> slice = (ReadOnlyMemory<SomeT>)arr[a..b];`

How to fix violations

To fix the violation of this rule: use the `AsSpan` or `AsMemory` extension method to avoid creating unnecessary data copies.

```

class C
{
    public void TestMethod(byte[] arr)
    {
        // The violation occurs for both statements below
        ReadOnlySpan<byte> tmp1 = arr[0..2];
        ReadOnlyMemory<byte> tmp3 = arr[5..8];
        ...
    }
}

```

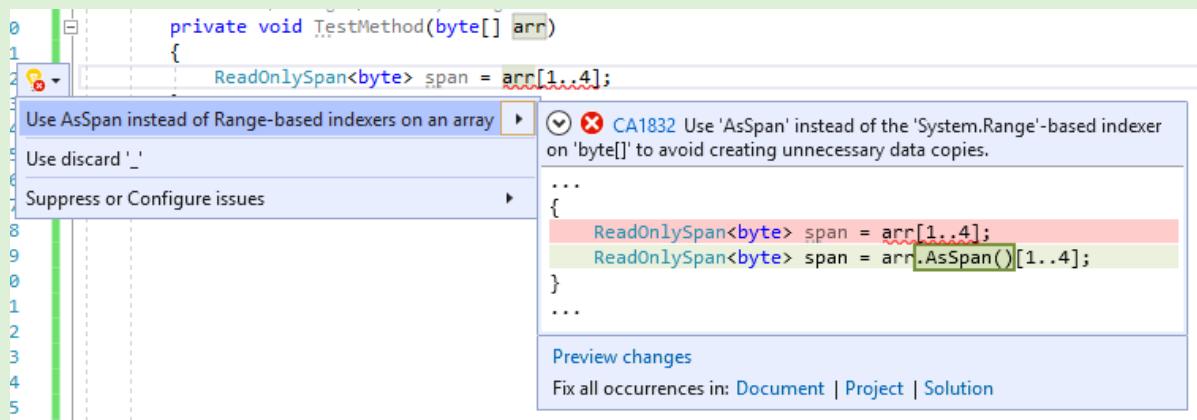
```

class C
{
    public void TestMethod(byte[] arr)
    {
        // The violations fixed with AsSpan or AsMemory accordingly
        ReadOnlySpan<byte> tmp1 = arr.AsSpan()[0..2];
        ReadOnlyMemory<byte> tmp3 = arr.AsMemory()[5..8];
        ...
    }
}

```

TIP

A code fix is available for this rule in Visual Studio. To use it, position the cursor on the violation and press **Ctrl+.** (period). Choose **Use AsSpan instead of the Range-based indexer on an array** from the list of options that is presented.



You can also avoid this warning by adding an explicit cast.

```

class C
{
    public void TestMethod(byte[] arr)
    {
        // The violation occurs
        ReadOnlySpan<byte> tmp1 = arr[0..2];
        ReadOnlyMemory<byte> tmp3 = arr[5..8];
        ...
    }
}

```

```
class C
{
    public void TestMethod(byte arr[])
    {
        // The violation fixed with explicit casting
        ReadOnlySpan<byte> tmp1 = (ReadOnlySpan<byte>)arr[0..2];
        ReadOnlyMemory<byte> tmp3 = (ReadOnlyMemory<byte>)arr[5..8];
        ...
    }
}
```

When to suppress warnings

It's safe to suppress a violation of this rule if creating a copy is intended.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1832
// The code that's violating the rule is on this line.
#pragma warning restore CA1832
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1832.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Performance.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Related rules

- [CA1831: Use AsSpan instead of Range-based indexers for string when appropriate](#)
- [CA1833: Use AsSpan or AsMemory instead of Range-based indexers for getting Span or Memory portion of an array](#)

See also

- [Performance rules](#)

CA1833: Use AsSpan or AsMemory instead of Range-based indexers for getting Span or Memory portion of an array

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA1833
Category	Performance
Fix is breaking or non-breaking	Non-breaking

Cause

When using a range-indexer on an array and implicitly assigning the value to `Span<T>` or `Memory<T>`.

Rule description

Using a range-indexer on array and assigning to a memory or span type: The range indexer on a `Span<T>` is a non-copying `Slice` operation, but for the range indexer on array the method `GetSubArray` will be used instead of `Slice`, which produces a copy of requested portion of the array. This copy is usually unnecessary when it is implicitly used as a `Span<T>` or `Memory<T>` value. If a copy isn't intended, use the `AsSpan` or `AsMemory` method to avoid the unnecessary copy. If the copy is intended, either assign it to a local variable first or add an explicit cast. The analyzer only reports when an implicit cast is used on the result of the range indexer operation.

Detects

Implicit conversions:

- `Span<SomeT> slice = arr[a..b];`
- `Memory<SomeT> slice = arr[a..b];`

Does not detect

Explicit conversions:

- `Span<SomeT> slice = (Span<SomeT>)arr[a..b];`
- `Memory<SomeT> slice = (Memory<SomeT>)arr[a..b];`

How to fix violations

To fix the violation of this rule: use the `AsSpan` or `AsMemory` extension method to avoid creating unnecessary data copies.

```

class C
{
    public void TestMethod(byte[] arr)
    {
        // The violation occurs for both statements below
        Span<byte> tmp2 = arr[0..5];
        Memory<byte> tmp4 = arr[5..10];
        ...
    }
}

```

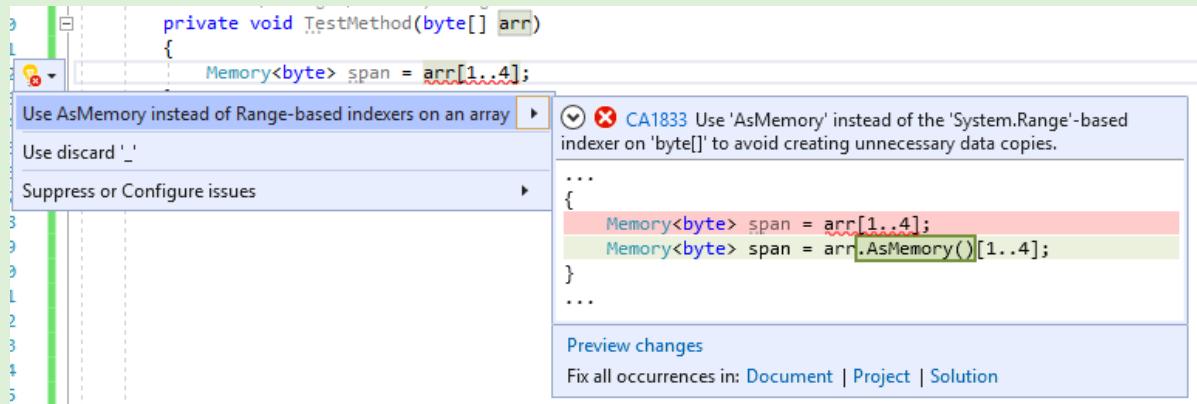
```

class C
{
    public void TestMethod(byte[] arr)
    {
        // The violations fixed with AsSpan or AsMemory accordingly
        Span<byte> tmp2 = arr.AsSpan()[0..5];
        Memory<byte> tmp4 = arr.AsMemory()[5..10];
        ...
    }
}

```

TIP

A code fix is available for this rule in Visual Studio. To use it, position the cursor on the violation and press **Ctrl+.** (period). Choose **Use AsMemory instead of the Range-based indexer on an array** from the list of options that's presented.



You can also avoid this warning by adding an explicit cast.

```

class C
{
    public void TestMethod(byte[] arr)
    {
        // The violation occurs
        Span<byte> tmp1 = arr[0..5];
        Memory<byte> tmp2 = arr[5..10];
        ...
    }
}

```

```
class C
{
    public void TestMethod(byte arr[])
    {
        // The violation fixed with explicit casting
        Span<byte> tmp1 = (Span<byte>)arr[0..5];
        Memory<byte> tmp2 = (Memory<byte>)arr[5..10];
        ...
    }
}
```

When to suppress warnings

It's safe to suppress a violation of this rule if creating a copy is intended.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1833
// The code that's violating the rule is on this line.
#pragma warning restore CA1833
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1833.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Performance.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Related rules

- [CA1831: Use AsSpan instead of Range-based indexers for string when appropriate](#)
- [CA1832: Use AsSpan or AsMemory instead of Range-based indexers for getting ReadOnlySpan or ReadOnlyMemory portion of an array](#)

See also

- [Performance rules](#)

CA1834: Use `StringBuilder.Append(char)` for single character strings

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA1834
Category	Performance
Fix is breaking or non-breaking	Non-breaking

Cause

This rule fires when a unit length string is passed to the `Append` method.

Rule description

When calling `StringBuilder.Append` with a unit length string, consider using a `const char` rather than a unit length `const string` to improve performance.

How to fix violations

The violation can either be fixed manually, or, in some cases, using Quick Actions to fix code in Visual Studio.

Examples:

Example 1

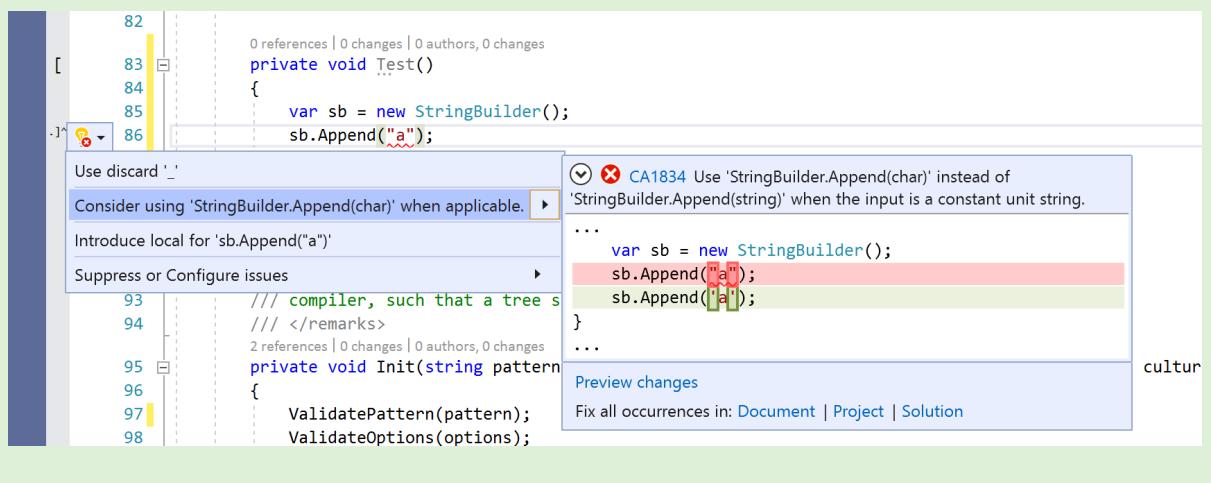
Invocations of `StringBuilder.Append` with a string literal of unit length:

```
using System;
using System.Text;

namespace TestNamespace
{
    class TestClass
    {
        private void TestMethod()
        {
            StringBuilder sb = new StringBuilder();
            sb.Append("a");
        }
    }
}
```

TIP

A code fix is available for this rule in Visual Studio. To use it, position the cursor on the violation and press **Ctrl+.**. Choose **Consider using 'StringBuilder.Append(char)' when applicable.** from the list of options that is presented.



Fix applied by Visual Studio:

```
using System;
using System.Text;

namespace TestNamespace
{
    class TestClass
    {
        private void TestMethod()
        {
            StringBuilder sb = new StringBuilder();
            sb.Append('a');
        }
    }
}
```

In some cases, for example when using a unit length `const string` class field, a code-fix is not suggested by Visual Studio (but the analyzer still fires). These instances require a manual fix.

Example 2

Invocations of `StringBuilder.Append` with a `const string` class field of unit length:

```
using System;
using System.Text;

namespace TestNamespace
{
    public class Program
    {
        public const string unitString = "a";

        static void Main(string[] args)
        {
            StringBuilder sb = new StringBuilder();
            sb.Append(unitString);
        }
    }
}
```

After careful analysis, `unitString` here can be changed to a `char` without causing any build errors.

```
using System;
using System.Text;

namespace TestNamespace
{
    public class Program
    {
        public const char unitString = 'a';

        static void Main(string[] args)
        {
            StringBuilder sb = new StringBuilder();
            sb.Append(unitString);
        }
    }
}
```

When to suppress warnings

It's safe to suppress a violation of this rule if you're not concerned about improving performance when using `StringBuilder`.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1834
// The code that's violating the rule is on this line.
#pragma warning restore CA1834
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1834.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Performance.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Performance rules](#)

CA1835: Prefer the memory-based overloads of ReadAsync/WriteAsync methods in stream-based classes

9/20/2022 • 4 minutes to read • [Edit Online](#)

	VALUE
Type name	PreferStreamAsyncMemoryOverloads
Rule ID	CA1835
Category	Performance
Fix is breaking or non-breaking	Non-breaking

Cause

This rule locates awaited invocations of the byte-array-based method overloads for `ReadAsync` and `WriteAsync`, and suggests using the memory-based method overloads instead, because they are more efficient.

Rule description

The memory-based method overloads have a more efficient memory usage than the byte array-based ones.

The rule works on `ReadAsync` and `WriteAsync` invocations of any class that inherits from `Stream`.

The rule only works when the method is preceded by the `await` keyword.

DETECTED METHOD	SUGGESTED METHOD
<code>ReadAsync(Byte[], Int32, Int32, CancellationToken)</code>	<code>ReadAsync(Memory<Byte>, CancellationToken)</code>
<code>ReadAsync(Byte[], Int32, Int32)</code>	<code>ReadAsync(Memory<Byte>, CancellationToken)</code> with <code>CancellationToken</code> set to <code>default</code> in C#, or <code>Nothing</code> in Visual Basic.
<code>WriteAsync(Byte[], Int32, Int32, CancellationToken)</code>	<code>WriteAsync(ReadOnlyMemory<Byte>, CancellationToken)</code>
<code>WriteAsync(Byte[], Int32, Int32)</code>	<code>WriteAsync(ReadOnlyMemory<Byte>, CancellationToken)</code> with <code>CancellationToken</code> set to <code>default</code> in C#, or <code>Nothing</code> in Visual Basic.

IMPORTANT

Make sure to pass the `offset` and `count` integer arguments to the created `Memory` or `ReadOnlyMemory` instances.

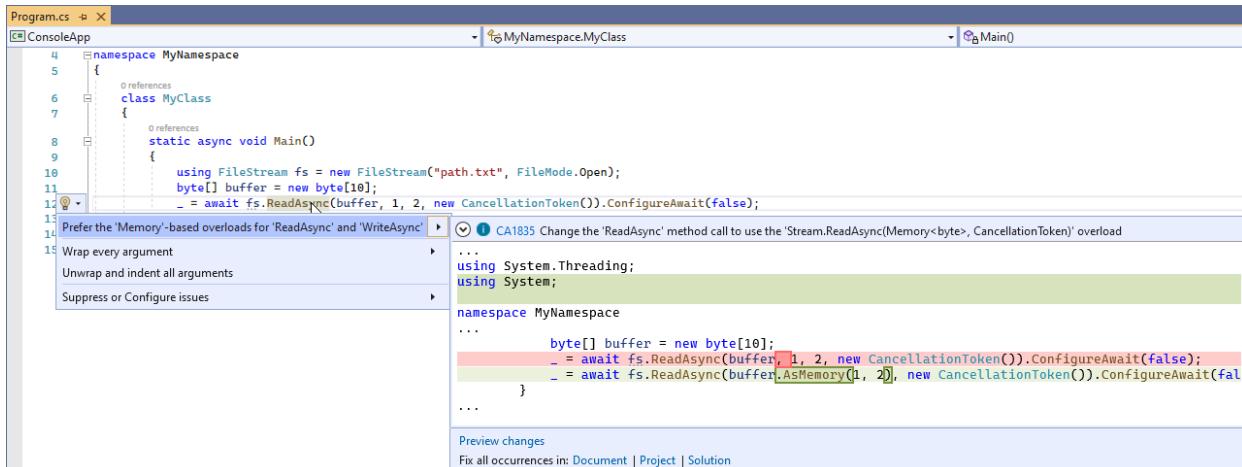
NOTE

Rule CA1835 is available in all .NET versions where the memory-based overloads are available:

- .NET Standard 2.1 and above.
- .NET Core 2.1 and above.

How to fix violations

You can either fix them manually, or you can opt to let Visual Studio do it for you, by hovering over the light bulb that shows up next to the method invocation, and selecting the suggested change. Example:



The rule can detect a variety of violations for the `ReadAsync` and `WriteAsync` methods. Here are examples of the cases that the rule can detect:

Example 1

Invocations of `ReadAsync`, without and with a `CancellationToken` argument:

```
using System;
using System.IO;
using System.Threading;

class MyClass
{
    public async void MyMethod(CancellationToken ct)
    {
        using (FileStream s = new FileStream("path.txt", FileMode.Create))
        {
            byte[] buffer = new byte[s.Length];
            await s.ReadAsync(buffer, 0, buffer.Length);
            await s.ReadAsync(buffer, 0, buffer.Length, ct);
        }
    }
}
```

Fix:

```

using System;
using System.IO;
using System.Threading;

class MyClass
{
    public async void MyMethod(CancellationToken ct)
    {
        using (FileStream s = new FileStream("path.txt", FileMode.Create))
        {
            byte[] buffer = new byte[s.Length];
            await s.ReadAsync(buffer.AsMemory(0, buffer.Length));
            await s.ReadAsync(buffer.AsMemory(0, buffer.Length), ct);
        }
    }
}

```

Example 2

Invocations of `WriteAsync`, without and with a `CancellationToken` argument:

```

using System;
using System.IO;
using System.Threading;

class MyClass
{
    public async void MyMethod(CancellationToken ct)
    {
        using (FileStream s = File.Open("path.txt", FileMode.Open))
        {
            byte[] buffer = { 0xBA, 0x5E, 0xBA, 0x11, 0xF0, 0x07, 0xBA, 0x11 };
            await s.WriteAsync(buffer, 0, buffer.Length);
            await s.WriteAsync(buffer, 0, buffer.Length, ct);
        }
    }
}

```

Fix:

```

using System;
using System.IO;
using System.Threading;

class MyClass
{
    public async void MyMethod()
    {
        using (FileStream s = File.Open("path.txt", FileMode.Open))
        {
            byte[] buffer = { 0xBA, 0x5E, 0xBA, 0x11, 0xF0, 0x07, 0xBA, 0x11 };
            await s.WriteAsync(buffer.AsMemory(0, buffer.Length));
            await s.WriteAsync(buffer.AsMemory(0, buffer.Length), ct);
        }
    }
}

```

Example 3

Invocations with `ConfigureAwait`:

```

using System;
using System.IO;
using System.Threading;

class MyClass
{
    public async void MyMethod()
    {
        using (FileStream s = File.Open("path.txt", FileMode.Open))
        {
            byte[] buffer1 = { 0xBA, 0x5E, 0xBA, 0x11, 0xF0, 0x07, 0xBA, 0x11 };
            await s.WriteAsync(buffer1, 0, buffer1.Length).ConfigureAwait(false);

            byte[] buffer2 = new byte[s.Length];
            await s.ReadAsync(buffer2, 0, buffer2.Length).ConfigureAwait(true);
        }
    }
}

```

Fix:

```

using System;
using System.IO;
using System.Threading;

class MyClass
{
    public async void MyMethod()
    {
        using (FileStream s = File.Open("path.txt", FileMode.Open))
        {
            byte[] buffer1 = { 0xBA, 0x5E, 0xBA, 0x11, 0xF0, 0x07, 0xBA, 0x11 };
            await s.WriteAsync(buffer1.AsMemory(0, buffer1.Length)).ConfigureAwait(false);

            byte[] buffer2 = new byte[s.Length];
            await s.ReadAsync(buffer2.AsMemory(0, buffer2.Length)).ConfigureAwait(true);
        }
    }
}

```

Non-violations

Following are some examples of invocations where the rule will **not** be fired.

The return value is saved in a `Task` variable instead of being awaited:

```

using System;
using System.IO;
using System.Threading;
using System.Threading.Tasks;

class MyClass
{
    public void MyMethod()
    {
        byte[] buffer = { 0xBA, 0x5E, 0xBA, 0x11, 0xF0, 0x07, 0xBA, 0x11 };
        using (FileStream s = new FileStream("path.txt", FileMode.Create))
        {
            Task t = s.WriteAsync(buffer, 0, buffer.Length);
        }
    }
}

```

The return value is returned by the wrapping method instead of being awaited:

```
using System;
using System.IO;
using System.Threading;
using System.Threading.Tasks;

class MyClass
{
    public Task MyMethod(FileStream s, byte[] buffer)
    {
        return s.WriteAsync(buffer, 0, buffer.Length);
    }
}
```

The return value is used to call `ContinueWith`, which is the method being awaited:

```
using System;
using System.IO;
using System.Threading;
using System.Threading.Tasks;

class MyClass
{
    public void MyMethod()
    {
        byte[] buffer = { 0xBA, 0x5E, 0xBA, 0x11, 0xF0, 0x07, 0xBA, 0x11 };
        using (FileStream s = new FileStream("path.txt", FileMode.Create))
        {
            await s.WriteAsync(buffer, 0, buffer.Length).ContinueWith(c => { /* ... */ });
        }
    }
}
```

When to suppress warnings

It's safe to suppress a violation of this rule if you're not concerned about improving performance when reading or writing buffers in stream-based classes.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1835
// The code that's violating the rule is on this line.
#pragma warning restore CA1835
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1835.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]\ndotnet_analyzer_diagnostic.category-Performance.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Performance rules](#)

CA1836: Prefer IsEmpty over Count when available

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA1836
Category	Performance
Fix is breaking or non-breaking	Non-breaking

Cause

The `Count` or `Length` property or the `Count<TSource>(IEnumerable<TSource>)` extension method was used to determine whether or not the object contains any items by comparing the value to `0` or `1`, and the object has a more efficient `IsEmpty` property that could be used instead.

Rule description

This rule flags the calls to the `Count` and `Length` properties or `Count<TSource>(IEnumerable<TSource>)` and `LongCount<TSource>(IEnumerable<TSource>)` LINQ methods when they are used to determine if the object contains any items and the object has a more efficient `IsEmpty` property.

The analysis of this rule originally overlapped with similar rules CA1827, CA1828, and CA1829; the analyzers of such rules were merged along with the one for CA1836 to report the best diagnosis in case of overlap.

How to fix violations

To fix a violation, replace the `Count<TSource>(IEnumerable<TSource>)` or `LongCount<TSource>(IEnumerable<TSource>)` method call or the `Length` or `Count` property access when it's used in an operation that determines if the object is empty with use of the `IsEmpty` property access. For example, the following two code snippets show a violation of the rule and how to fix it:

```
using System.Collections.Concurrent;

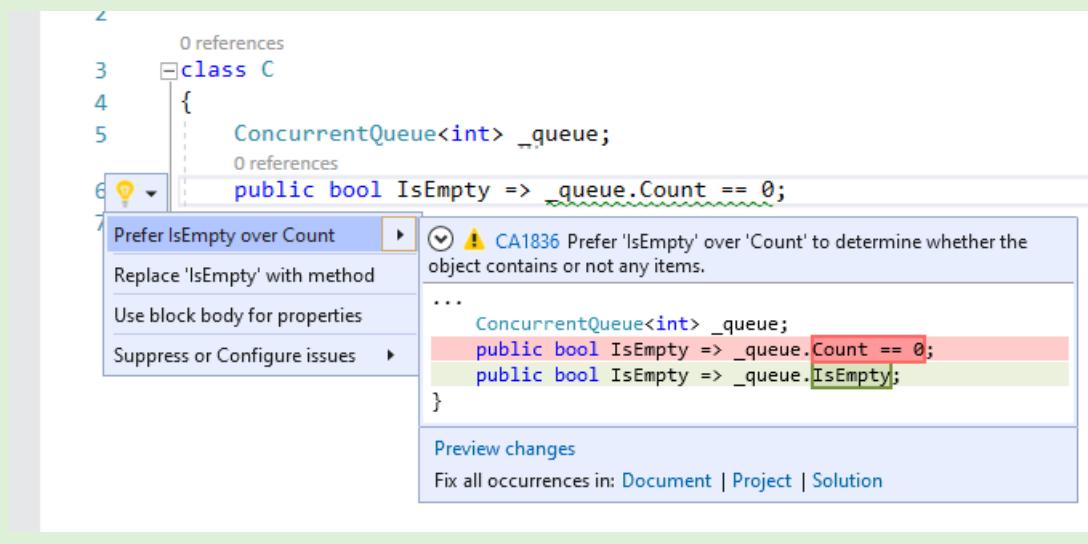
class C
{
    ConcurrentQueue<int> _queue;
    public bool IsEmpty => _queue.Count == 0;
}
```

```
using System.Collections.Concurrent;

class C
{
    ConcurrentQueue<int> _queue;
    public bool IsEmpty => _queue.IsEmpty;
}
```

TIP

A code fix is available for this rule in Visual Studio. To use it, position the cursor on the violation and press **Ctrl+.** (period). Choose **'Prefer 'IsEmpty' over 'Count'** to determine whether the object contains or not any items from the list of options that's presented.



When to suppress warnings

It's safe to suppress a violation of this rule if you're not concerned about the performance impact from unnecessary item enumeration to compute the count.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1836
// The code that's violating the rule is on this line.
#pragma warning restore CA1836
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1836.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Performance.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Related rules

- [CA1827: Do not use Count/LongCount when Any can be used](#)
- [CA1828: Do not use CountAsync/LongCountAsync when AnyAsync can be used](#)
- [CA1829: Use Length/Count property instead of Enumerable.Count method](#)

See also

- [Performance rules](#)

CA1837: Use Environment.ProcessId instead of Process.GetCurrentProcess().Id

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA1837
Category	Performance
Fix is breaking or non-breaking	Non-breaking

Cause

This rule locates calls to `System.Diagnostics.Process.GetCurrentProcess().Id` and suggests using `System.Environment.ProcessId` instead, because it is more efficient.

Rule description

`System.Diagnostics.Process.GetCurrentProcess().Id` is expensive:

- It allocates a `Process` instance, usually just to get the `Id`.
- The `Process` instance needs to be disposed, which has a performance impact.
- It's easy to forget to call `Dispose()` on the `Process` instance.
- If nothing else besides `Id` uses the `Process` instance, then the linked size grows unnecessarily by increasing the graph of types referenced.
- It is somewhat difficult to discover or find this API.

`System.Environment.ProcessId` avoids all the above.

Note

Rule CA1837 is available starting on .NET 5.0.

How to fix violations

The violation can either be fixed manually, or, in some cases, using Quick Actions to fix code in Visual Studio.

The following two code snippets show a violation of the rule and how to fix it:

```
using System.Diagnostics;

class MyClass
{
    void MyMethod()
    {
        int pid = Process.GetCurrentProcess().Id;
    }
}
```

```

Imports System.Diagnostics

Class MyClass
    Private Sub MyMethod()
        Dim pid As Integer = Process.GetCurrentProcess().Id
    End Function
End Class

```

```

using System.Diagnostics;

class MyClass
{
    void MyMethod()
    {
        int pid = System.Environment.ProcessId;
    }
}

```

```

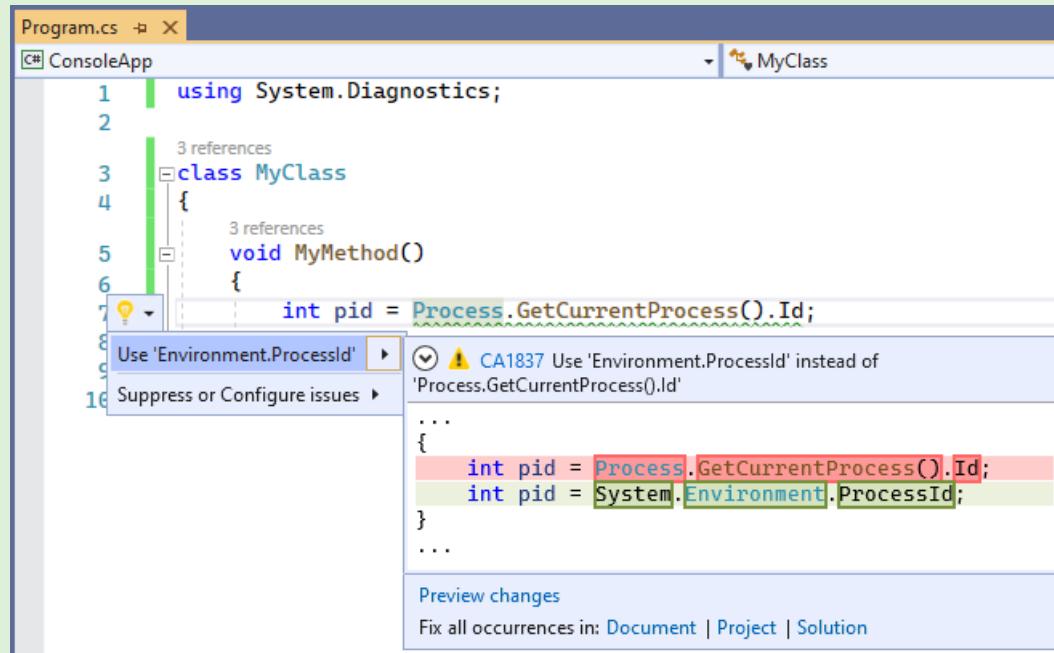
Imports System.Diagnostics

Class MyClass
    Private Sub MyMethod()
        Dim pid As Integer = System.Environment.ProcessId
    End Function
End Class

```

TIP

A code fix is available for this rule in Visual Studio. To use it, position the cursor on the violation and press **Ctrl+.** (period). Choose **Use 'Environment.ProcessId' instead of 'Process.GetCurrentProcess().Id'** from the list of options that's presented.



When to suppress warnings

It's safe to suppress a violation of this rule if you're not concerned about the performance impact from unnecessary allocation and eventual disposal of a `Process` instance.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1837
// The code that's violating the rule is on this line.
#pragma warning restore CA1837
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.cs,vb]
dotnet_diagnostic.CA1837.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.cs,vb]
dotnet_analyzer_diagnostic.category-Performance.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Performance rules](#)

CA1838: Avoid `StringBuilder` parameters for P/Invokes

9/20/2022 • 3 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA1838
Category	Performance
Fix is breaking or non-breaking	Non-breaking

Cause

A [P/Invoke](#) has a `StringBuilder` parameter.

Rule description

Marshalling of `StringBuilder` always creates a native buffer copy, resulting in multiple allocations for one P/Invoke call. To marshal a `StringBuilder` as a P/Invoke parameter, the runtime will:

- Allocate a native buffer.
- If it is an `In` parameter, copy the contents of the `StringBuilder` to the native buffer.
- If it is an `Out` parameter, copy the native buffer into a newly allocated managed array.

By default, `StringBuilder` is `In` and `Out`.

For more information about marshalling strings, see [Default marshalling for strings](#).

This rule is disabled by default, because it can require case-by-case analysis of whether the violation is of interest and potentially non-trivial refactoring to address the violation. Users can explicitly enable this rule by [configuring its severity](#).

How to fix violations

In general, addressing a violation involves reworking the P/Invoke and its callers to use a buffer instead of `StringBuilder`. The specifics would depend on the use cases for the P/Invoke.

Here is an example for the common scenario of using `StringBuilder` as an output buffer to be filled by the native function:

```

// Violation
[DllImport("MyLibrary", CharSet = CharSet.Unicode)]
private static extern void Foo(StringBuilder sb, ref int length);

public void Bar()
{
    int BufferSize = ...
    StringBuilder sb = new StringBuilder(BufferSize);
    int len = sb.Capacity;
    Foo(sb, ref len);
    string result = sb.ToString();
}

```

For use cases where the buffer is small and `unsafe` code is acceptable, `stackalloc` can be used to allocate the buffer on the stack:

```

[DllImport("MyLibrary", CharSet = CharSet.Unicode)]
private static extern unsafe void Foo(char* buffer, ref int length);

public void Bar()
{
    int BufferSize = ...
    unsafe
    {
        char* buffer = stackalloc char[BufferSize];
        int len = BufferSize;
        Foo(buffer, ref len);
        string result = new string(buffer);
    }
}

```

For larger buffers, a new array can be allocated as the buffer:

```

[DllImport("MyLibrary", CharSet = CharSet.Unicode)]
private static extern void Foo([Out] char[] buffer, ref int length);

public void Bar()
{
    int BufferSize = ...
    char[] buffer = new char[BufferSize];
    int len = buffer.Length;
    Foo(buffer, ref len);
    string result = new string(buffer);
}

```

When the P/Invoke is frequently called for larger buffers, `ArrayPool<T>` can be used to avoid the repeated allocations and memory pressure that comes with them:

```
[DllImport("MyLibrary", CharSet = CharSet.Unicode)]
private static extern unsafe void Foo([Out] char[] buffer, ref int length);

public void Bar()
{
    int BufferSize = ...
    char[] buffer = ArrayPool<char>.Shared.Rent(BufferSize);
    try
    {
        int len = buffer.Length;
        Foo(buffer, ref len);
        string result = new string(buffer);
    }
    finally
    {
        ArrayPool<char>.Shared.Return(buffer);
    }
}
```

If the buffer size is not known until runtime, the buffer may need to be created differently based on the size to avoid allocating large buffers with `stackalloc`.

The preceding examples use 2-byte wide characters (`CharSet.Unicode`). If the native function uses 1-byte characters (`CharSet.Ansi`), a `byte` buffer can be used instead of a `char` buffer. For example:

```
[DllImport("MyLibrary", CharSet = CharSet.Ansi)]
private static extern unsafe void Foo(byte* buffer, ref int length);

public void Bar()
{
    int BufferSize = ...
    unsafe
    {
        byte* buffer = stackalloc byte[BufferSize];
        int len = BufferSize;
        Foo(buffer, ref len);
        string result = Marshal.PtrToStringAnsi((IntPtr)buffer);
    }
}
```

If the parameter is also used as input, the buffers need to be populated with the string data with any null terminator explicitly added.

When to suppress warnings

Suppress a violation of this rule if you're not concerned about the performance impact of marshalling a `StringBuilder`.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1838
// The code that's violating the rule is on this line.
#pragma warning restore CA1838
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]\ndotnet_diagnostic.CA1838.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]\ndotnet_analyzer_diagnostic.category-Performance.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Default marshalling for strings](#)
- [Native interoperability best practices](#)
- [ArrayPool<T>](#)
- [stackalloc](#)
- [Charsets](#)
- [Performance rules](#)

CA1839: Use Environment.ProcessPath instead of Process.GetCurrentProcess().MainModule.FileName

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA1839
Category	Performance
Fix is breaking or non-breaking	Non-breaking

Cause

Using `Process.GetCurrentProcess().MainModule.FileName` for getting the path to the file that launched the process instead of `Environment.ProcessPath`.

Rule description

`System.Diagnostics.Process.GetCurrentProcess().MainModule.FileName` is expensive:

- It allocates a `Process` and `ProcessModule` instance, usually just to get the `FileName`.
- The `Process` instance needs to be disposed, which has a performance impact.
- It's easy to forget to call `Dispose()` on the `Process` instance.
- If nothing else besides `FileName` uses the `Process` instance, then the linked size grows unnecessarily by increasing the graph of types referenced.
- It is somewhat difficult to discover or find this API.

`System.Environment.ProcessPath` avoids all of these downsides and produces the same information.

NOTE

`System.Environment.ProcessPath` is available starting in .NET 6.

How to fix violations

The violation can either be fixed manually, or, in some cases, using Quick Actions to fix code in Visual Studio.

The following two code snippets show a violation of the rule and how to fix it:

```
using System.Diagnostics;

class MyClass
{
    void MyMethod()
    {
        string path = Process.GetCurrentProcess().MainModule.FileName; // Violation occurs
    }
}
```

```

Imports System.Diagnostics

Class MyClass
    Private Sub MyMethod()
        Dim path As String = Process.GetCurrentProcess().MainModule.FileName ' Violation occurs.
    End Function
End Class

```

```

using System.Diagnostics;

class MyClass
{
    void MyMethod()
    {
        string path = System.Environment.ProcessPath; // Violation fixed
    }
}

```

```

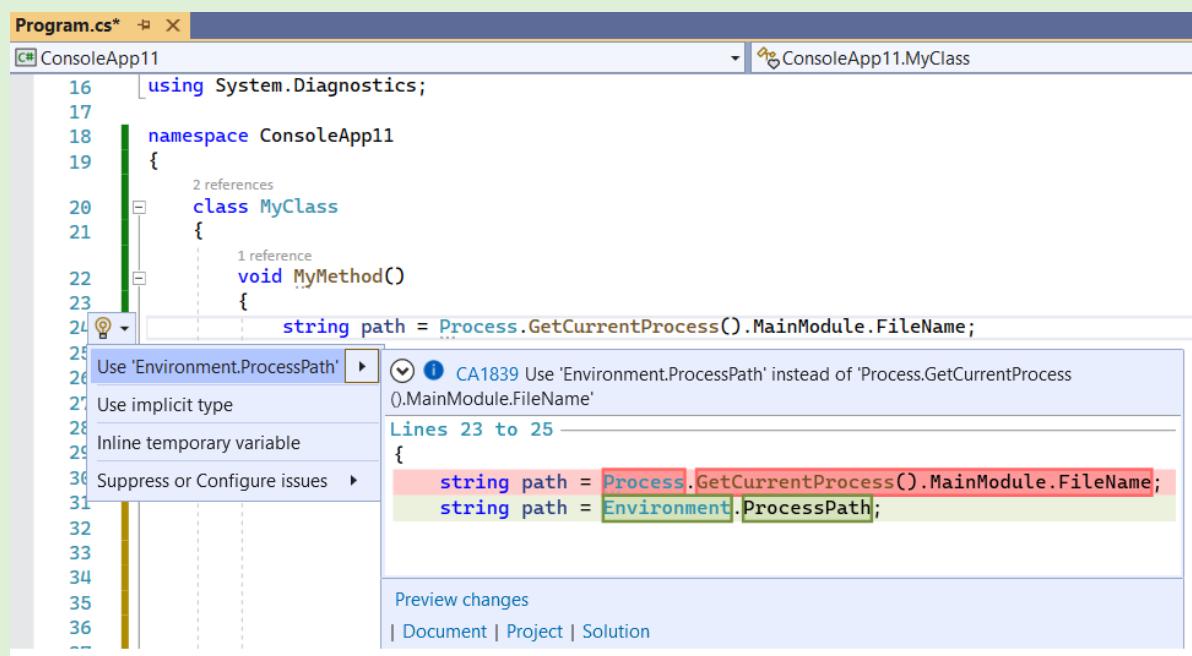
Imports System.Diagnostics

Class MyClass
    Private Sub MyMethod()
        Dim path As String = System.Environment.ProcessPath ' Violation fixed.
    End Function
End Class

```

TIP

A code fix is available for this rule in Visual Studio. To use it, position the cursor on the violation and press **Ctrl+.** (period). Choose **Use 'Environment.ProcessPath'** from the list of options that's presented.



When to suppress warnings

It's safe to suppress a violation of this rule if you're not concerned about the performance impact from unnecessary allocation and eventual disposal of the [Process](#) and [ProcessModule](#) instances.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1839
// The code that's violating the rule is on this line.
#pragma warning restore CA1839
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.cs,vb]
dotnet_diagnostic.CA1839.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.cs,vb]
dotnet_analyzer_diagnostic.category-Performance.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Performance rules](#)
- [Use 'Environment.ProcessId' instead of 'Process.GetCurrentProcess\(\).Id'](#)
- [Use 'Environment.CurrentManagedThreadId' instead of 'Thread.CurrentThread.ManagedThreadId'](#)

CA1840: Use Environment.CurrentManagedThreadId instead of Thread.CurrentThread.ManagedThreadId

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA1840
Category	Performance
Fix is breaking or non-breaking	Non-breaking

Cause

Using `Thread.CurrentThread.ManagedThreadId` for getting the current managed thread ID instead of `System.Environment.CurrentManagedThreadId`.

Rule description

`System.Environment.CurrentManagedThreadId` is a compact and efficient replacement of the `Thread.CurrentThread.ManagedThreadId` pattern.

How to fix violations

The violation can either be fixed manually, or, in some cases, using Quick Actions to fix code in Visual Studio.

The following two code snippets show a violation of the rule and how to fix it:

```
using System.Threading;

class MyClass
{
    void MyMethod()
    {
        int id = Thread.CurrentThread.ManagedThreadId; // Violation occurs
    }
}
```

```
Imports System.Threading

Class MyClass
    Private Sub MyMethod()
        Dim id As Integer = Thread.CurrentThread.ManagedThreadId ' Violation occurs.
    End Function
End Class
```

```

using System.Threading;

class MyClass
{
    void MyMethod()
    {
        int id = System.Environment.CurrentManagedThreadId; // Violation fixed
    }
}

```

```

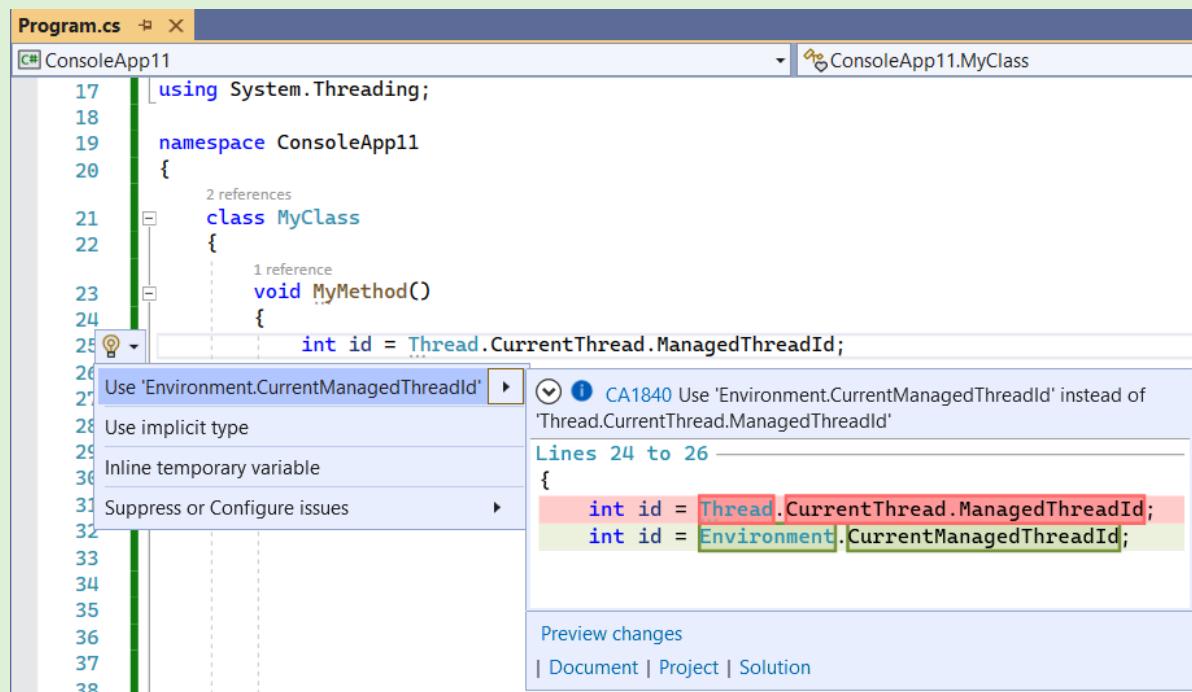
Imports System.Threading

Class MyClass
    Private Sub MyMethod()
        Dim id As Integer = System.Environment.CurrentManagedThreadId ' Violation fixed.
    End Function
End Class

```

TIP

A code fix is available for this rule in Visual Studio. To use it, position the cursor on the violation and press **Ctrl+.** (period). Choose **Use 'Environment.CurrentManagedThreadId'** from the list of options that's presented.



When to suppress warnings

It's safe to suppress a violation of this rule if you're not concerned about the performance impact from using `Thread.CurrentThread.ManagedThreadId`.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1840
// The code that's violating the rule is on this line.
#pragma warning restore CA1840
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1840.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Performance.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Performance rules](#)
- [Use 'Environment.ProcessId' instead of 'Process.GetCurrentProcess\(\).Id'](#)
- [Use 'Environment.ProcessPath' instead of 'Process.GetCurrentProcess\(\).MainModule.FileName'](#)

CA1841: Prefer Dictionary Contains methods

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA1841
Category	Performance
Fix is breaking or non-breaking	Non-breaking

Cause

This rule locates calls to a `Contains` method on the `Keys` or `Values` collection of an `IDictionary< TKey, TValue >` that could be replaced with a call to a `ContainsKey` or `ContainsValue` method on the dictionary itself.

Rule description

Calling `Contains` on the `Keys` or `Values` collection can often be more expensive than calling `ContainsKey` or `ContainsValue` on the dictionary itself:

- Many dictionary implementations lazily instantiate the key and value collections, which means that accessing the `Keys` or `Values` collection may result in extra allocations.
- You may end up calling an extension method on `IEnumerable<T>` if the keys or values collection uses explicit interface implementation to hide methods on `ICollection<T>`. This can lead to reduced performance, especially when accessing the key collection. Most dictionary implementations are able to provide a fast O(1) containment check for keys, while the `Contains` extension method on `IEnumerable<T>` usually does a slow O(n) containment check.

How to fix violations

To fix violations, replace calls to `dictionary.Keys.Contains` or `dictionary.Values.Contains` with calls to `dictionary.ContainsKey` or `dictionary.ContainsValue`, respectively.

The following code snippet shows examples of violations, and how to fix them.

```

using System.Collections.Generic;
// Importing this namespace brings extension methods for IEnumarable<T> into scope.
using System.Linq;

class Example
{
    void Method()
    {
        var dictionary = new Dictionary<string, int>();

        // Violation
        dictionary.Keys.Contains("hello world");

        // Fixed
        dictionary.ContainsKey("hello world");

        // Violation
        dictionary.Values.Contains(17);

        // Fixed
        dictionary.ContainsValue(17);
    }
}

```

```

Imports System.Collection.Generic
' Importing this namespace brings extension methods for IEnumarable(Of T) into scope.
' Note that in Visual Basic, this namespace is often imported automatically throughout the project.
Imports System.Linq

Class Example
    Private Sub Method()
        Dim dictionary = New Dictionary(Of String, Of Integer)

        ' Violation
        dictionary.Keys.Contains("hello world")

        ' Fixed
        dictionary.ContainsKey("hello world")

        ' Violation
        dictionary.Values.Contains(17)

        ' Fixed
        dictionary.ContainsValue(17)
    End Sub
End Class

```

When to suppress warnings

It is safe to suppress warnings from this rule if the code in question is not performance-critical.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```

#pragma warning disable CA1841
// The code that's violating the rule is on this line.
#pragma warning restore CA1841

```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1841.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Performance.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Performance rules](#)

CA1842: Do not use 'WhenAll' with a single task

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA1842
Category	Performance
Fix is breaking or non-breaking	Non-breaking

Cause

[WhenAll](#) is called with a single task.

Rule description

Using `WhenAll` with a single task may result in performance loss.

How to fix violations

You should await or return the task instead.

When to suppress warnings

Do not suppress a warning from this rule.

See also

- [CA1843: Do not use 'WaitAll' with a single task](#)
- [Performance rules](#)

CA1843: Do not use 'WaitAll' with a single task

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA1843
Category	Performance
Fix is breaking or non-breaking	Non-breaking

Cause

`WaitAll` is called with a single task.

Rule description

Using `WaitAll` with a single task may result in performance loss.

How to fix violations

You should await or return the task instead.

When to suppress warnings

Do not suppress a warning from this rule.

See also

- [CA1842: Do not use 'WhenAll' with a single task](#)
- [Performance rules](#)

CA1844: Provide memory-based overrides of async methods when subclassing 'Stream'

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA1844
Category	Performance
Fix is breaking or non-breaking	Non-breaking

Cause

A type derived from `Stream` overrides `ReadAsync(Byte[], Int32, Int32, CancellationToken)` but does not override `ReadAsync(Memory<Byte>, CancellationToken)`. Or, a type derived from `Stream` overrides `WriteAsync(Byte[], Int32, Int32, CancellationToken)` but does not override `WriteAsync(ReadOnlyMemory<Byte>, CancellationToken)`.

Rule description

The memory-based `ReadAsync` and `WriteAsync` methods were added to improve performance, which they accomplish in multiple ways:

- They return `ValueTask` and `ValueTask<int>` instead of `Task` and `Task<int>`, respectively.
- They allow any type of buffer to be passed in without having to perform an extra copy to an array.

In order to realize these performance benefits, types that derive from `Stream` must provide their own memory-based implementation. Otherwise, the default implementation will be forced to copy the memory into an array in order to call the array-based implementation, resulting in reduced performance. When the caller passes in a `Memory<T>` or `ReadOnlyMemory<T>` instance that's not backed by an array, performance is affected more.

How to fix violations

The easiest way to fix violations is to rewrite your array-based implementation as a memory-based implementation, and then implement the array-based methods in terms of the memory-based methods.

When to suppress warnings

It's safe to suppress a warning from this rule if any of the following situations apply:

- The performance hit is not a concern.
- You know that your `Stream` subclass will only ever use the array-based methods.
- Your `Stream` subclass has dependencies that don't support memory-based buffers.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1844
// The code that's violating the rule is on this line.
#pragma warning restore CA1844
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1844.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Performance.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Performance rules](#)

CA1845: Use span-based 'string.Concat'

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA1845
Category	Performance
Fix is breaking or non-breaking	Non-breaking

Cause

This rule locates string-concatenation expressions that contain `Substring` calls and suggests replacing `Substring` with `AsSpan` and using the span-based overload of `String.Concat`.

Rule description

Calling `Substring` produces a copy of the extracted substring. By using `AsSpan` instead of `Substring` and calling the overload of `string.Concat` that accepts spans, you can eliminate the unnecessary string allocation.

How to fix violations

To fix violations:

1. Replace the string concatenation with a call to `string.Concat`, and
2. Replace calls to `Substring` with calls to `AsSpan`.

The following code snippet shows examples of violations, and how to fix them.

```
using System;

class Example
{
    public void Method()
    {
        string text = "fwobz the fwutzle";

        // Violation: allocations by Substring are wasteful.
        string s1 = text.Substring(10) + "___" + text.Substring(0, 5);

        // Fixed: using AsSpan avoids allocations of temporary strings.
        string s2 = string.Concat(text.Aspans(10), "___", text.Aspans(0, 5));
    }
}
```

When to suppress warnings

Do not suppress warnings from this rule. There is no reason to use `Substring` over `AsSpan` when the extracted substring is only being passed to a method with a span-based equivalent.

See also

- [Performance rules](#)

CA1846: Prefer `AsSpan` over `Substring`

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA1846
Category	Performance
Fix is breaking or non-breaking	Non-breaking

Cause

The result of a call to one of the `String.Substring` overloads is passed to a method with an available overload that accepts `ReadOnlySpan<Char>`.

Rule description

`Substring` allocates a new `string` object on the heap and performs a full copy of the extracted text. String manipulation is a performance bottleneck for many programs. Allocating many small, short-lived strings on a hot path can create enough collection pressure to impact performance. The O(n) copies created by `Substring` become relevant when the substrings get large. The `Span<T>` and `ReadOnlySpan<T>` types were created to solve these performance problems.

Many APIs that accept strings also have overloads that accept a `ReadOnlySpan<System.Char>` argument. When such overloads are available, you can improve performance by calling `AsSpan` instead of `Substring`.

How to fix violations

To fix a violation of this rule, replace the call to `string.Substring` with a call to one of the `MemoryExtensions.AsSpan` extension methods.

```
using System;

public void MyMethod(string iniFileLine)
{
    // Violation
    int.TryParse(iniFileLine.Substring(7), out int x);
    int.TryParse(iniFileLine.Substring(2, 5), out int y);

    // Fix
    int.TryParse(iniFileLine.AsSpan(7), out int x);
    int.TryParse(iniFileLine.AsSpan(2, 5), out int y);
}
```

```
Imports System

Public Sub MyMethod(iniFileLine As String)
    Dim x As Integer
    Dim y As Integer

    ' Violation
    Integer.TryParse(iniFileLine.Substring(7), x)
    Integer.TryParse(iniFileLine.Substring(2, 5), y)

    ' Fix
    Integer.TryParse(iniFileLine.AsSpan(7), x)
    Integer.TryParse(iniFileLine.AsSpan(2, 5), y)
End Sub
```

When to suppress warnings

It is safe to suppress warnings from this rule if performance is not a concern.

See also

- [Performance warnings](#)

CA1847: Use string.Contains(char) instead of string.Contains(string) with single characters

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA1847
Category	Performance
Fix is breaking or non-breaking	Non-breaking

Cause

`string.Contains(string)` is used when `string.Contains(char)` was available.

Rule description

When searching for a single character, using `string.Contains(char)` offers better performance than `string.Contains(string)`.

How to fix violations

In general, the rule is fixed simply by using a char literal instead of a string literal.

```
public bool ContainsLetterI()
{
    var testString = "I am a test string.";
    return testString.Contains("I");
}
```

```
Public Function ContainsLetterI() As Boolean
    Dim testString As String = "I am a test string."
    Return testString.Contains("I")
End Function
```

This code can be changed to use a char literal instead.

```
public bool ContainsLetterI()
{
    var testString = "I am a test string.";
    return testString.Contains('I');
}
```

```
Public Function ContainsLetterI() As Boolean
    Dim testString As String = "I am a test string."
    Return testString.Contains("I"c)
End Function
```

When to suppress warnings

Suppress a violation of this rule if you're not concerned about the performance impact of the search invocation in question.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1847
// The code that's violating the rule is on this line.
#pragma warning restore CA1847
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1847.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Performance.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Performance rules](#)

CA1848: Use the LoggerMessage delegates

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA1848
Category	Performance
Fix is breaking or non-breaking	Non-breaking

Cause

Use of [logger extension methods](#), such as [LogInformation](#) and [LogDebug](#).

Rule description

For high-performance logging scenarios, use the [LoggerMessage](#) pattern.

How to fix violations

Use [LoggerMessage](#) to fix violations of this rule.

[LoggerMessage](#) provides the following performance advantages over Logger extension methods:

- Logger extension methods require "boxing" (converting) value types, such as `int`, into `object`. The [LoggerMessage](#) pattern avoids boxing by using static `Action` fields and extension methods with strongly typed parameters.
- Logger extension methods must parse the message template (named format string) every time a log message is written. [LoggerMessage](#) only requires parsing a template once when the message is defined.

When to suppress warnings

Do not suppress a warning from this rule.

See also

- [High-performance logging with LoggerMessage in ASP.NET Core](#)
- [Performance rules](#)

CA1849: Call async methods when in an async method

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA1849
Category	Performance
Fix is breaking or non-breaking	Non-breaking

Cause

All methods where an Async-suffixed equivalent exists will produce this warning when called from a Task-returning method. In addition, calling `Task.Wait()`, `Task<T>.Result`, or `Task.GetAwaiter().GetResult()` will produce this warning.

Rule description

In a method which is already asynchronous, calls to other methods should be to their async versions, where they exist.

How to fix violations

Violation:

```
Task DoAsync()
{
    file.Read(buffer, 0, 10);
}
```

Fix:

Await the async version of the method:

```
async Task DoAsync()
{
    await file.ReadAsync(buffer, 0, 10);
}
```

When to suppress warnings

It's safe to suppress a warning from this rule in the case where there are two separate code paths for sync and async code, using an if condition. Also if there is a check for whether the Task has resolved, it is safe to use sync methods and properties.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1849
// The code that's violating the rule is on this line.
#pragma warning restore CA1849
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1849.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Performance.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Performance rules](#)

CA1850: Prefer static `HashData` method over

`ComputeHash`

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA1850
Category	Performance
Fix is breaking or non-breaking	Non-breaking

Cause

An instance of a type that derives from `HashAlgorithm` is created to call its `computeHash` method, and that type has a static `HashData` method.

Rule description

Static `HashData` methods were introduced in .NET 5 on the following types:

- [MD5](#)
- [SHA1](#)
- [SHA256](#)
- [SHA384](#)
- [SHA512](#)

These methods help simplify code in cases where you just want to hash some data.

It's more efficient to use these static `HashData` methods than to create and manage a `HashAlgorithm` instance to call `ComputeHash`.

How to fix violations

In general, you can fix the rule by changing your code to call `HashData` and remove use of the `HashAlgorithm` instance.

```
public bool CheckHash(byte[] buffer)
{
    using (var sha256 = SHA256.Create())
    {
        byte[] digest = sha256.ComputeHash(buffer);
        return DoesHashExist(digest);
    }
}
```

```
Public Function CheckHash(buffer As Byte()) As Boolean
    Using sha256 As SHA256 = SHA256.Create()
        Dim digest As Byte() = sha256.ComputeHash(buffer)
        Return DoesHashExist(digest)
    End Using
End Function
```

The previous code can be changed to call the static [HashData\(Byte\[\]\)](#) method directly.

```
public bool CheckHash(byte[] buffer)
{
    byte[] digest = SHA256.HashData(buffer);
    return DoesHashExist(digest);
}
```

```
Public Function CheckHash(buffer As Byte()) As Boolean
    Dim digest As Byte() = SHA256.HashData(buffer)
    Return DoesHashExist(digest)
End Function
```

When to suppress warnings

It is safe to suppress a warning from this rule.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1850
// The code that's violating the rule is on this line.
#pragma warning restore CA1850
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1850.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Performance.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Performance rules](#)

CA1851: Possible multiple enumerations of `IEnumerable` collection

9/20/2022 • 3 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA1851
Category	Performance
Fix is breaking or non-breaking	Non-breaking
Introduced version	.NET 7

Cause

Detected multiple enumerations of an `IEnumerable` collection.

Rule description

A collection of type `IEnumerable` or `IEnumerable<T>` has the ability to defer enumeration when it's generated. Many LINQ methods, such as `Select`, use `deferred execution`. Enumeration starts when the collection is passed into a LINQ enumeration method, like `ElementAt`, or used in a `for each statement`. The enumeration result is not calculated once and cached, like `Lazy`.

If the enumeration operation itself is expensive, for example, a query into a database, multiple enumerations would hurt the performance of the program.

If the enumeration operation has side effects, multiple enumerations might result in bugs.

How to fix violations

If the underlying type of your `IEnumerable` collection is some other type, such as `List` or `Array`, it's safe to convert the collection to its underlying type to fix the diagnostic.

Violation:

```
public void MyMethod(IEnumerable<int> input)
{
    var count = input.Count();
    foreach (var i in input) { }
}
```

```
Public Sub MyMethod(input As IEnumerable(Of Integer))
    Dim count = input.Count()
    For Each i In input
        Next
    End Sub
```

Fix:

```
public void MyMethod(IEnumerable<int> input)
{
    // If the underlying type of 'input' is List<int>
    var inputList = (List<int>)input;
    var count = inputList.Count();
    foreach (var i in inputList) { }
}
```

```
Public Sub MyMethod(input As IEnumerable(Of Integer))
    ' If the underlying type of 'input' is array
    Dim inputArray = CType(input, Integer())
    Dim count = inputArray.Count()
    For Each i In inputArray
        Next
End Sub
```

If the underlying type of the `IEnumerable` collection uses an iterator-based implementation (for example, if it's generated by LINQ methods like `Select` or by using the `yield` keyword), you can fix the violation by materializing the collection. However, this allocates extra memory.

For example:

Violation:

```
public void MyMethod()
{
    var someStrings = GetStrings().Select(i => string.Concat("Hello"));

    // It takes 2 * O(n) to complete 'Count()' and 'Last()' calls, where 'n' is the length of 'someStrings'.
    var count = someStrings.Count();
    var lastElement = someStrings.Last();
}
```

```
Public Sub MyMethod()
    Dim someStrings = GetStrings().Select(Function(i) String.Concat(i, "Hello"))

    ' It takes 2 * O(n) to complete 'Count()' and 'Last()' calls, where 'n' is the length of 'someStrings'.
    Dim count = someStrings.Count()
    Dim lastElement = someStrings.Last()
End Sub
```

Fix:

```
public void MyMethod()
{
    var someStrings = GetStrings().Select(i => string.Concat("Hello"));
    // Materialize it into an array.
    // Note: This operation would allocate O(n) memory,
    // and it takes O(n) time to finish, where 'n' is the length of 'someStrings'.
    var someStringsArray = someStrings.ToArray()

    // It takes 2 * O(1) to complete 'Count()' and 'Last()' calls.
    var count = someStringsArray.Count();
    var lastElement = someStringsArray.Last();
}
```

```

Public Sub MyMethod()
    Dim someStrings = GetStrings().Select(Function(i) String.Concat(i, "Hello"))
    ' Materialize it into an array.
    ' Note: This operation would allocate O(n) memory,
    ' and it takes O(n) time to finish, where 'n' is the length of 'someStrings'.
    Dim someStringsArray = someStrings.ToArray()

    ' It takes 2 * O(1) to complete 'Count()' and 'Last()' calls.
    Dim count = someStrings.Count()
    Dim lastElement = someStrings.Last()
End Sub

```

Configure customized enumeration methods and LINQ chain methods

By default, all the methods in the `System.Linq` namespace are included in the analysis scope. You can add custom methods that enumerate an `IEnumerable` argument into the scope by setting the `enumeration_methods` option in an `.editorconfig` file.

You can also add customized LINQ chain methods (that is, methods take an `IEnumerable` argument and return a new `IEnumerable` instance) to the analysis scope by setting the `linq_chain_methods` option in an `.editorconfig` file.

Configure the default assumption of methods take `IEnumerable` parameters

By default, all customized methods that accept an `IEnumerable` argument are assumed not to enumerate the argument. You can change this by setting the `assume_method_enumerates_parameters` option in an `.editorconfig` file.

When to suppress warnings

It's safe to suppress this warning if the underlying type of the `IEnumerable` collection is some other type like `List` or `Array`, or if you're sure that methods that take an `IEnumerable` collection don't enumerate it.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```

#pragma warning disable CA1851
// The code that's violating the rule is on this line.
#pragma warning restore CA1851

```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```

[*.{cs,vb}]
dotnet_diagnostic.CA1851.severity = none

```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]\ndotnet_analyzer_diagnostic.category-Performance.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Deferred execution example \(LINQ to XML\)](#)

CA1854: Prefer the

`IDictionary.TryGetValue(TKey, out TValue)`

method

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA1854
Category	Performance
Fix is breaking or non-breaking	Non-breaking

Cause

An `IDictionary` element access that's guarded by an `IDictionary.ContainsKey` check.

Rule description

When an element of an `IDictionary` is accessed, the indexer implementation checks for a null value by calling the `IDictionary.ContainsKey` method. If you also call `IDictionary.ContainsKey` in an `if` clause to guard a value lookup, two lookups are performed when only one is needed.

How to fix violations

Replace the `IDictionary.ContainsKey` invocation and element access with a call to the `IDictionary.TryGetValue` method.

Violation:

```
public string? GetValue(string key)
{
    if (_dictionary.ContainsKey(key))
    {
        return _dictionary[key];
    }

    return null;
}
```

```
Public Function GetValue(key As String) As String
    If _dictionary.ContainsKey(key) Then
        Return _dictionary(key)
    End If

    Return Nothing
End Function
```

Fix:

```
public string? GetValue(string key)
{
    if (_dictionary.TryGetValue(key, out string? value))
    {
        return value;
    }

    return null;
}
```

```
Public Function GetValue(key As String) As String
    Dim value as String

    If _dictionary.TryGetValue(key, value) Then
        Return value
    End If

    Return Nothing
End Function
```

When to suppress warnings

It's safe to suppress this warning if you're using a custom implementation of `IDictionary` that avoids a value lookup when performing the `IDictionary.ContainsKey` check.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1854
// The code that's violating the rule is on this line.
#pragma warning restore CA1854
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1854.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Performance.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

SingleFile rules

9/20/2022 • 2 minutes to read • [Edit Online](#)

SingleFile rules for .NET.

NOTE

Prior to .NET 6, this category was named `Publish`.

In this section

RULE	DESCRIPTION
IL3000 Avoid accessing Assembly file path when publishing as a single file	Avoid accessing Assembly file path when publishing as a single file
IL3001 Avoid accessing Assembly file path when publishing as a single file	Avoid accessing Assembly file path when publishing as a single file
IL3002 Avoid calling members annotated with 'RequiresAssemblyFilesAttribute' when publishing as a single file	Avoid calling members annotated with 'RequiresAssemblyFilesAttribute' when publishing as a single file
IL3003 'RequiresAssemblyFilesAttribute' annotations must match across all interface implementations or overrides.	'RequiresAssemblyFilesAttribute' annotations must match across all interface implementations or overrides.

IL3000: Avoid accessing Assembly file path when publishing as a single file

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	IL3000
Category	SingleFile
Fix is breaking or non-breaking	Non-breaking

Cause

When publishing as a single-file (for example, by setting the PublishSingleFile property in a project to true), calling the `Assembly.Location` property for assemblies embedded inside the single-file bundle always returns an empty string.

How to fix violations

If the app only needs the containing directory for the single-file bundle, consider using the `AppContext.BaseDirectory` property instead. Otherwise, consider removing the call entirely.

When to suppress warnings

It's appropriate to silence this warning if the assembly being accessed is definitely not in the single-file bundle. This may be the case if the assembly is being loaded dynamically from a file path.

IL3001: Avoid accessing Assembly file path when publishing as a single file

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	IL3001
Category	SingleFile
Fix is breaking or non-breaking	Non-breaking

Cause

When publishing as a single file (for example, by setting the PublishSingleFile property in a project to true), calling the `Assembly.GetFile(s)` methods for assemblies embedded inside the single-file bundle always throws an exception, as these methods are not single-file compatible.

How to fix violations

To embed files in assemblies in single-file bundles, consider using embedded resources and the `Assembly.GetManifestResourceStream` method.

When to suppress warnings

It's appropriate to silence this warning if the assembly being accessed is definitely not in the single-file bundle. This may be the case if the assembly is being loaded dynamically from a file path.

IL3002: Avoid calling members annotated with 'RequiresAssemblyFilesAttribute' when publishing as a single file.

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	IL3002
Category	SingleFile
Fix is breaking or non-breaking	Non-breaking

Cause

When publishing an app as a single file (for example, by setting the `PublishSingleFile` property in a project to `true`), calling members annotated with the `RequiresAssemblyFilesAttribute` attribute is not single-file compatible. These calls are not compatible because members annotated with this attribute require assembly files to be on disk, and the assemblies embedded in a single-file app are memory loaded.

Example:

```
[RequiresAssemblyFiles(Message="Use 'MethodFriendlyToSingleFile' instead", Url="http://help/assemblyfiles")]
void MethodWithAssemblyFilesUsage()
{
}
void TestMethod()
{
    // IL3002: Using member 'MethodWithAssemblyFilesUsage' which has 'RequiresAssemblyFilesAttribute'
    // can break functionality when embedded in a single-file app. Use 'MethodFriendlyToSingleFile' instead.
    http://help/assemblyfiles
    MethodWithAssemblyFilesUsage();
}
```

How to fix violations

Members annotated with the 'RequiresAssemblyFilesAttribute' attribute have a message intended to give useful information to users who are publishing as a single file. Consider adapting existing code to the attribute's message or removing the violating call.

When to suppress warnings

It's appropriate to suppress the warning when the existing code has been adapted to the recommendation outlined in the 'RequiresAssemblyFilesAttribute' attribute's message.

IL3003: 'RequiresAssemblyFilesAttribute' annotations must match across all interface implementations or overrides.

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	IL3003
Category	SingleFile
Fix is breaking or non-breaking	Non-breaking

Cause

When publishing as a single file (for example, by setting the `PublishSingleFile` property in a project to `true`), interface implementations or derived classes with members that don't have matching annotations of `[RequiresAssemblyFiles]` can lead to calling a member with the attribute, which is not single-file compatible.

There are four possible scenarios where the warning can be generated:

A member of a base class has the attribute but the overriding member of the derived class does not have the attribute:

```
public class Base
{
    [RequiresAssemblyFiles]
    public virtual void TestMethod() {}

    public class Derived : Base
    {
        // IL3003: Base member 'Base.TestMethod' with 'RequiresAssemblyFilesAttribute' has a derived member
        // 'Derived.TestMethod()' without 'RequiresAssemblyFilesAttribute'. For all interfaces and overrides the
        // implementation attribute must match the definition attribute.
        public override void TestMethod() {}
    }
}
```

A member of a base class does not have the attribute but the overriding member of the derived class does have the attribute:

```
public class Base
{
    public virtual void TestMethod() {}

    public class Derived : Base
    {
        // IL3003: Member 'Derived.TestMethod()' with 'RequiresAssemblyFilesAttribute' overrides base member
        // 'Base.TestMethod()' without 'RequiresAssemblyFilesAttribute'. For all interfaces and overrides the
        // implementation attribute must match the definition attribute.
        [RequiresAssemblyFiles]
        public override void TestMethod() {}
    }
}
```

An interface member has the attribute but its implementation does not have the attribute:

```
interface IRAF
{
    [RequiresAssemblyFiles]
    void TestMethod();
}

class Implementation : IRAF
{
    // IL3003: Interface member 'IRAF.TestMethod()' with 'RequiresAssemblyFilesAttribute' has an
    // implementation member 'Implementation.TestMethod()' without 'RequiresAssemblyFilesAttribute'. For all
    // interfaces and overrides the implementation attribute must match the definition attribute.
    public void TestMethod () { }
}
```

An interface member does not have the attribute but its implementation does have the attribute:

```
interface INoRAF
{
    void TestMethod();
}

class Implementation : INoRAF
{
    [RequiresAssemblyFiles]
    // IL3003: Member 'Implementation.TestMethod()' with 'RequiresAssemblyFilesAttribute' implements
    // interface member 'INoRAF.TestMethod()' without 'RequiresAssemblyFilesAttribute'. For all interfaces and
    // overrides the implementation attribute must match the definition attribute.
    public void TestMethod () { }
}
```

How to fix violations

For all interfaces and overrides, the implementation must match the definition in terms of the presence or absence of the 'RequiresAssemblyFilesAttribute' attribute. Either both the members contain the attribute or neither of them. Remove or add the attribute on the interface/base class member or implementing/deriving class member so that the attribute is on both or neither.

When to suppress warnings

This warning should not be suppressed.

Reliability rules

9/20/2022 • 2 minutes to read • [Edit Online](#)

Reliability rules support library and application reliability, such as correct memory and thread usage. The reliability rules include:

RULE	DESCRIPTION
CA2000: Dispose objects before losing scope	Because an exceptional event might occur that will prevent the finalizer of an object from running, the object should be explicitly disposed before all references to it are out of scope.
CA2002: Do not lock on objects with weak identity	An object is said to have a weak identity when it can be directly accessed across application domain boundaries. A thread that tries to acquire a lock on an object that has a weak identity can be blocked by a second thread in a different application domain that has a lock on the same object.
CA2007: Do not directly await a Task	An asynchronous method <code>awaits</code> a <code>Task</code> directly.
CA2008: Do not create tasks without passing a TaskScheduler	A task creation or continuation operation uses a method overload that does not specify a <code>TaskScheduler</code> parameter.
CA2009: Do not call ToImmutableCollection on an ImmutableCollection value	<code>ToImmutable</code> method was unnecessarily called on an immutable collection from <code>System.Collections.Immutable</code> namespace.
CA2011: Do not assign property within its setter	A property was accidentally assigned a value within its own <code>set accessor</code> .
CA2012: Use ValueTasks correctly	ValueTasks returned from member invocations are intended to be directly awaited. Attempts to consume a ValueTask multiple times or to directly access one's result before it's known to be completed may result in an exception or corruption. Ignoring such a ValueTask is likely an indication of a functional bug and may degrade performance.
CA2013: Do not use ReferenceEquals with value types	When comparing values using <code>System.Object.ReferenceEquals</code> , if objA and objB are value types, they are boxed before they are passed to the <code>ReferenceEquals</code> method. This means that even if both objA and objB represent the same instance of a value type, the <code>ReferenceEquals</code> method nevertheless returns false.
CA2014: Do not use stackalloc in loops.	Stack space allocated by a <code>stackalloc</code> is only released at the end of the current method's invocation. Using it in a loop can result in unbounded stack growth and eventual stack overflow conditions.
CA2015: Do not define finalizers for types derived from MemoryManager<T>	Adding a finalizer to a type derived from <code>MemoryManager<T></code> may permit memory to be freed while it is still in use by a <code>Span<T></code> .

RULE	DESCRIPTION
CA2016: Forward the <code>CancellationToken</code> parameter to methods that take one	Forward the <code>CancellationToken</code> parameter to methods that take one to ensure the operation cancellation notifications gets properly propagated, or pass in <code>CancellationToken.None</code> explicitly to indicate intentionally not propagating the token.
CA2017: Parameter count mismatch	Number of parameters supplied in the logging message template do not match the number of named placeholders.
CA2018: The <code>count</code> argument to <code>Buffer.BlockCopy</code> should specify the number of bytes to copy	When using <code>Buffer.BlockCopy</code> , the <code>count</code> argument specifies the number of bytes to copy. You should only use <code>Array.Length</code> for the <code>count</code> argument on arrays whose elements are exactly one byte in size. <code>byte</code> , <code>sbyte</code> , and <code>bool</code> arrays have elements that are one byte in size.

CA2000: Dispose objects before losing scope

9/20/2022 • 7 minutes to read • [Edit Online](#)

	Value
Rule ID	CA2000
Category	Reliability
Fix is breaking or non-breaking	Non-breaking

Cause

A local object of an [IDisposable](#) type is created, but the object is not disposed before all references to the object are out of scope.

By default, this rule analyzes the entire codebase, but this is [configurable](#).

Rule description

If a disposable object is not explicitly disposed before all references to it are out of scope, the object will be disposed at some indeterminate time when the garbage collector runs the finalizer of the object. Because an exceptional event might occur that will prevent the finalizer of the object from running, the object should be explicitly disposed instead.

Special cases

Rule CA2000 does not fire for local objects of the following types even if the object is not disposed:

- [System.IO.Stream](#)
- [System.IO.StringReader](#)
- [System.IO.TextReader](#)
- [System.IO.TextWriter](#)
- [System.Resources.IResourceReader](#)

Passing an object of one of these types to a constructor and then assigning it to a field indicates a *dispose ownership transfer* to the newly constructed type. That is, the newly constructed type is now responsible for disposing of the object. If your code passes an object of one of these types to a constructor, no violation of rule CA2000 occurs even if the object is not disposed before all references to it are out of scope.

How to fix violations

To fix a violation of this rule, call [Dispose](#) on the object before all references to it are out of scope.

You can use the [using statement](#) ([Using](#) in Visual Basic) to wrap objects that implement [IDisposable](#). Objects that are wrapped in this manner are automatically disposed at the end of the [using](#) block. However, the following situations should not or cannot be handled with a [using](#) statement:

- To return a disposable object, the object must be constructed in a [try/finally](#) block outside of a [using](#) block.

- Do not initialize members of a disposable object in the constructor of a `using` statement.
- When constructors that are protected by only one exception handler are nested in the [acquisition part of a `using` statement](#), a failure in the outer constructor can result in the object created by the nested constructor never being closed. In the following example, a failure in the `StreamReader` constructor can result in the `FileStream` object never being closed. CA2000 flags a violation of the rule in this case.

```
using (StreamReader sr = new StreamReader(new FileStream("C:/myfile.txt", FileMode.Create)))
{ ... }
```

- Dynamic objects should use a shadow object to implement the dispose pattern of [IDisposable](#) objects.

When to suppress warnings

Do not suppress a warning from this rule unless:

- You've called a method on your object that calls `Dispose`, such as `Close`.
- The method that raised the warning returns an [IDisposable](#) object that wraps your object.
- The allocating method does not have dispose ownership; that is, the responsibility to dispose the object is transferred to another object or wrapper that's created in the method and returned to the caller.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA2000
// The code that's violating the rule is on this line.
#pragma warning restore CA2000
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA2000.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Reliability.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

Use the following options to configure which parts of your codebase to run this rule on.

- [Exclude specific symbols](#)
- [Exclude specific types and their derived types](#)

You can configure these options for just this rule, for all rules, or for all rules in this category ([Reliability](#)). For more information, see [Code quality rule configuration options](#).

Exclude specific symbols

You can exclude specific symbols, such as types and methods, from analysis. For example, to specify that the rule should not run on any code within types named `MyType`, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Symbol name only (includes all symbols with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#). Each symbol name requires a symbol-kind prefix, such as `M:` for methods, `T:` for types, and `N:` for namespaces.
- `.ctor` for constructors and `.cctor` for static constructors.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType</code>	Matches all symbols named <code>MyType</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType1 MyType2</code>	Matches all symbols named either <code>MyType1</code> or <code>MyType2</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS.MyType.MyMethod(ParamType)</code>	Matches specific method <code>MyMethod</code> with the specified fully qualified signature.
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS1.MyType1.MyMethod1(ParamType) M:NS2.MyType2.MyMethod2(ParamType)</code>	Matches specific methods <code>MyMethod1</code> and <code>MyMethod2</code> with the respective fully qualified signatures.

Exclude specific types and their derived types

You can exclude specific types and their derived types from analysis. For example, to specify that the rule should not run on any methods within types named `MyType` and their derived types, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Type name only (includes all types with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#), with an optional `T:` prefix.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType</code>	Matches all types named <code>MyType</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType1 MyType2</code>	Matches all types named either <code>MyType1</code> or <code>MyType2</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS.MyType</code>	Matches specific type <code>MyType</code> with given fully qualified name and all of its derived types.

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS1.MyType1 M:NS2.MyType2</code>	Matches specific types <code>MyType1</code> and <code>MyType2</code> with the respective fully qualified names, and all of their derived types.

Related rules

- [CA2213: Disposable fields should be disposed](#)

Example 1

If you're implementing a method that returns a disposable object, use a try/finally block without a catch block to make sure that the object is disposed. By using a try/finally block, you allow exceptions to be raised at the fault point and make sure that object is disposed.

In the OpenPort1 method, the call to open the ISerializable object SerialPort or the call to SomeMethod can fail. A CA2000 warning is raised on this implementation.

In the OpenPort2 method, two SerialPort objects are declared and set to null:

- `tempPort`, which is used to test that the method operations succeed.
- `port`, which is used for the return value of the method.

The `tempPort` is constructed and opened in a `try` block, and any other required work is performed in the same `try` block. At the end of the `try` block, the opened port is assigned to the `port` object that will be returned and the `tempPort` object is set to `null`.

The `finally` block checks the value of `tempPort`. If it is not null, an operation in the method has failed, and `tempPort` is closed to make sure that any resources are released. The returned port object will contain the opened SerialPort object if the operations of the method succeeded, or it will be null if an operation failed.

```

public SerialPort OpenPort1(string portName)
{
    SerialPort port = new SerialPort(portName);
    port.Open(); //CA2000 fires because this might throw
    SomeMethod(); //Other method operations can fail
    return port;
}

public SerialPort OpenPort2(string portName)
{
    SerialPort tempPort = null;
    SerialPort port = null;
    try
    {
        tempPort = new SerialPort(portName);
        tempPort.Open();
        SomeMethod();
        //Add any other methods above this line
        port = tempPort;
        tempPort = null;
    }
    finally
    {
        if (tempPort != null)
        {
            tempPort.Close();
        }
    }
    return port;
}

```

```

Public Function OpenPort1(ByVal PortName As String) As SerialPort

    Dim port As New SerialPort(PortName)
    port.Open()      'CA2000 fires because this might throw
    SomeMethod()    'Other method operations can fail
    Return port

End Function

Public Function OpenPort2(ByVal PortName As String) As SerialPort

    Dim tempPort As SerialPort = Nothing
    Dim port As SerialPort = Nothing

    Try
        tempPort = New SerialPort(PortName)
        tempPort.Open()
        SomeMethod()
        'Add any other methods above this line
        port = tempPort
        tempPort = Nothing

    Finally
        If Not tempPort Is Nothing Then
            tempPort.Close()
        End If

    End Try

    Return port

End Function

```

Example 2

By default, the Visual Basic compiler has all arithmetic operators check for overflow. Therefore, any Visual Basic arithmetic operation might throw an [OverflowException](#). This could lead to unexpected violations in rules such as CA2000. For example, the following CreateReader1 function will produce a CA2000 violation because the Visual Basic compiler is emitting an overflow checking instruction for the addition that could throw an exception that would cause the StreamReader not to be disposed.

To fix this, you can disable the emitting of overflow checks by the Visual Basic compiler in your project or you can modify your code as in the following CreateReader2 function.

To disable the emitting of overflow checks, right-click the project name in Solution Explorer and then click **Properties**. Click **Compile**, click **Advanced Compile Options**, and then check **Remove integer overflow checks**.

```
Imports System.IO

Class CA2000
    Public Function CreateReader1(ByVal x As Integer) As StreamReader
        Dim local As New StreamReader("C:\Temp.txt")
        x += 1
        Return local
    End Function

    Public Function CreateReader2(ByVal x As Integer) As StreamReader
        Dim local As StreamReader = Nothing
        Dim localTemp As StreamReader = Nothing
        Try
            localTemp = New StreamReader("C:\Temp.txt")
            x += 1
            local = localTemp
            localTemp = Nothing
        Finally
            If (Not (localTemp Is Nothing)) Then
                localTemp.Dispose()
            End If
        End Try
        Return local
    End Function
End Class
```

See also

- [IDisposable](#)
- [Dispose Pattern](#)

CA2002: Do not lock on objects with weak identity

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA2002
Category	Reliability
Fix is breaking or non-breaking	Non-breaking

Cause

A thread attempts to acquire a lock on an object that has a weak identity.

Rule description

An object is said to have a weak identity when it can be directly accessed across application domain boundaries. A thread that tries to acquire a lock on an object that has a weak identity can be blocked by a second thread in a different application domain that has a lock on the same object.

The following types have a weak identity and are flagged by the rule:

- [String](#)
- Arrays of value types, including [integral types](#), [floating-point types](#), and [Boolean](#).
- [MarshalByRefObject](#)
- [ExecutionEngineException](#)
- [OutOfMemoryException](#)
- [StackOverflowException](#)
- [MemberInfo](#)
- [ParameterInfo](#)
- [Thread](#)
- [this](#) or [Me](#) object

How to fix violations

To fix a violation of this rule, use an object from a type that is not in the list in the Description section.

When to suppress warnings

It is safe to suppress the warning if the locked object is `this` or `Me` and the visibility of the self object type is private or internal, and the instance is not accessible using any public reference.

Otherwise, do not suppress a warning from this rule.

SUPPRESS A WARNING

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA2002
// The code that's violating the rule is on this line.
#pragma warning restore CA2002
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.cs,vb]
dotnet_diagnostic.CA2002.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.cs,vb]
dotnet_analyzer_diagnostic.category-Reliability.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

RELATED RULES

[CA2213: Disposable fields should be disposed](#)

EXAMPLE

The following example shows some object locks that violate the rule.

```
Imports System
Imports System.IO
Imports System.Reflection
Imports System.Threading

Namespace ca2002

    Class WeakIdentities

        Sub SyncLockOnWeakId1()

            SyncLock GetType(WeakIdentities)
            End SyncLock

        End Sub

        Sub SyncLockOnWeakId2()

            Dim stream As New MemoryStream()
            SyncLock stream
            End SyncLock

        End Sub

        Sub SyncLockOnWeakId3()

            SyncLock "string"
            End SyncLock

        End Sub

        Sub SyncLockOnWeakId4()

            Dim member As MethodInfo =
                Me.GetType().GetMember("SyncLockOnWeakId1")(0)
            SyncLock member
            End SyncLock

        End Sub

        Sub SyncLockOnWeakId5()

            Dim outOfMemory As New OutOfMemoryException()
            SyncLock outOfMemory
            End SyncLock

        End Sub

    End Class

End Namespace
```

```
class WeakIdentities
{
    void LockOnWeakId1()
    {
        lock (typeof(WeakIdentities)) { }
    }

    void LockOnWeakId2()
    {
        MemoryStream stream = new MemoryStream();
        lock (stream) { }
    }

    void LockOnWeakId3()
    {
        lock ("string") { }
    }

    void LockOnWeakId4()
    {
        MemberInfo member = this.GetType().GetMember("LockOnWeakId1")[0];
        lock (member) { }
    }
    void LockOnWeakId5()
    {
        OutOfMemoryException outOfMemory = new OutOfMemoryException();
        lock (outOfMemory) { }
    }
}
```

See also

- [Monitor](#)
- [AppDomain](#)
- [lock Statement \(C#\)](#)
- [SyncLock Statement \(Visual Basic\)](#)

CA2007: Do not directly await a Task

9/20/2022 • 3 minutes to read • [Edit Online](#)

	Value
Rule ID	CA2007
Category	Reliability
Fix is breaking or non-breaking	Non-breaking

Cause

An asynchronous method `awaits` a `Task` directly.

Rule description

When an asynchronous method awaits a `Task` directly, continuation usually occurs in the same thread that created the task, depending on the async context. This behavior can be costly in terms of performance and can result in a deadlock on the UI thread. Consider calling `Task.ConfigureAwait(Boolean)` to signal your intention for continuation.

How to fix violations

To fix violations, call `ConfigureAwait` on the awaited `Task`. You can pass either `true` or `false` for the `continueOnCapturedContext` parameter.

- Calling `ConfigureAwait(true)` on the task has the same behavior as not explicitly calling `ConfigureAwait`. By explicitly calling this method, you're letting readers know you intentionally want to perform the continuation on the original synchronization context.
- Call `ConfigureAwait(false)` on the task to schedule continuations to the thread pool, thereby avoiding a deadlock on the UI thread. Passing `false` is a good option for app-independent libraries.

Example

The following code snippet generates the warning:

```
public async Task Execute()
{
    Task task = null;
    await task;
}
```

To fix the violation, call `ConfigureAwait` on the awaited `Task`:

```
public async Task Execute()
{
    Task task = null;
    await task.ConfigureAwait(false);
}
```

When to suppress warnings

This warning is intended for libraries, where the code may be executed in arbitrary environments and where code shouldn't make assumptions about the environment or how the caller of the method may be invoking or waiting on it. It is generally appropriate to suppress the warning entirely for projects that represent application code rather than library code; in fact, running this analyzer on application code (for example, button click event handlers in a WinForms or WPF project) is likely to lead to the wrong actions being taken.

You can suppress this warning in any situation where either the continuation should be scheduled back to the original context or where there is no such context in place. For example, when writing code in a button click event handler in a WinForms or WPF application, in general the continuation from an await should run on the UI thread, and thus the default behavior of scheduling the continuation back to the originating context is desirable. As another example, when writing code in an ASP.NET Core application, by default there is no [SynchronizationContext](#) or [TaskScheduler](#), for which reason a `ConfigureAwait` wouldn't actually change any behavior.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA2007
// The code that's violating the rule is on this line.
#pragma warning restore CA2007
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA2007.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Reliability.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

Use the following options to configure which parts of your codebase to run this rule on.

- [Exclude async void methods](#)
- [Output kind](#)

You can configure all of these options for just this rule, for all rules, or for all rules in this category ([Reliability](#)). For more information, see [Code quality rule configuration options](#).

Exclude async void methods

You can configure whether you want to exclude asynchronous methods that don't return a value from this rule.

To exclude these kinds of methods, add the following key-value pair to an `.editorconfig` file in your project:

```
# Package version 2.9.0 and later  
dotnet_code_quality.CA2007.exclude_async_void_methods = true  
  
# Package version 2.6.3 and earlier  
dotnet_code_quality.CA2007.skip_async_void_methods = true
```

Output kind

You can also configure which output assembly kinds to apply this rule to. For example, to only apply this rule to code that produces a console application or a dynamically linked library (that is, not a UI app), add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CA2007.output_kind = ConsoleApplication, DynamicallyLinkedLibrary
```

See also

- [ConfigureAwait FAQ](#)
- [Should I await a task with ConfigureAwait\(false\)?](#)
- [CA2008: Do not create tasks without passing a TaskScheduler](#)
- [Reliability rules](#)

CA2008: Do not create tasks without passing a TaskScheduler

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA2008
Category	Reliability
Fix is breaking or non-breaking	Non-breaking

Cause

A task creation or continuation operation uses a method overload that does not specify a [TaskScheduler](#) parameter.

Rule description

The following .NET task creation and continuation methods have overloads that allow specifying or omitting a [TaskScheduler](#) instance:

- [System.Threading.Tasks.TaskFactory.StartNew](#) methods
- [System.Threading.Tasks.Task.ContinueWith](#) methods

Always specify an explicit [TaskScheduler](#) argument to avoid the default [Current](#) value, whose behavior is defined by the caller and may vary at run time. [Current](#) returns the scheduler associated with whatever [Task](#) is currently running on that thread. If there is no such task, it returns [Default](#), which represents the thread pool. Using [Current](#) could lead to deadlocks or UI responsiveness issues in some situations, when it was intended to create the task on the thread pool, but instead it waits to get back onto the UI thread.

For further information and detailed examples, see [New TaskCreationOptions and TaskContinuationOptions in .NET Framework 4.5](#).

NOTE

[VSTHRD105 - Avoid method overloads that assume TaskScheduler.Current](#) is a similar rule implemented in [Microsoft.VisualStudio.Threading.Analyzers](#) package.

How to fix violations

To fix violations, call the method overload that takes a [TaskScheduler](#) and explicitly pass in [Default](#) or [Current](#) to make the intent clear.

When to suppress warnings

This warning is intended primarily for libraries, where the code may be executed in arbitrary environments and where code shouldn't make assumptions about the environment or how the caller of the method may be

invoking or waiting on it. It may be appropriate to suppress the warning for projects that represent application code rather than library code.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA2008
// The code that's violating the rule is on this line.
#pragma warning restore CA2008
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA2008.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Reliability.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [New TaskCreationOptions and TaskContinuationOptions in .NET Framework 4.5](#)
- [VSTHRD105 - Avoid method overloads that assume TaskScheduler.Current](#)
- [CA2007: Do not directly await a Task](#)
- [Reliability rules](#)

CA2009: Do not call `ToImmutableCollection` on an `ImmutableCollection` value

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA2009
Category	Reliability
Fix is breaking or non-breaking	Non-breaking

Cause

`ToImmutable` method was unnecessarily called on an immutable collection from `System.Collections.Immutable` namespace.

Rule description

`System.Collections.Immutable` namespace contains types that define immutable collections. This rule analyzes the following immutable collection types:

- `System.Collections.Immutable.ImmutableArray<T>`
- `System.Collections.Immutable.ImmutableList<T>`
- `System.Collections.Immutable.ImmutableHashSet<T>`
- `System.Collections.Immutable.ImmutableSortedSet<T>`
- `System.Collections.Immutable.ImmutableDictionary<TKey,TValue>`
- `System.Collections.Immutable.ImmutableSortedDictionary<TKey,TValue>`

These types define extension methods that create a new immutable collection from an existing `IEnumerable<T>` collection.

- `ImmutableArray<T>` defines `ToImmutableArray`.
- `ImmutableList<T>` defines `ToImmutableList`.
- `ImmutableHashSet<T>` defines `ToImmutableHashSet`.
- `ImmutableSortedSet<T>` defines `ToImmutableSortedSet`.
- `ImmutableDictionary<TKey,TValue>` defines `ToImmutableDictionary`.
- `ImmutableSortedDictionary<TKey,TValue>` defines `ToImmutableSortedDictionary`.

These extension methods are designed to convert a mutable collection to an immutable collection. However, the caller might accidentally pass in an immutable collection as input to these methods. This can represent a performance and/or a functional issue.

- Performance issue: Unnecessary boxing, unboxing, and/or runtime type checks on an immutable collection.
- Potential functional issue: Caller assumed to be operating on a mutable collection, when it actually had an immutable collection.

How to fix violations

To fix violations, remove the redundant `ToImmutable` call on an immutable collection. For example, the following two code snippets show a violation of the rule and how to fix them:

```
using System;
using System.Collections.Generic;
using System.Collections.Immutable;

public class C
{
    public void M(IEnumerable<int> collection, ImmutableList<int> immutableArray)
    {
        // This is fine.
        M2(collection.ToImmutableArray());

        // This leads to CA2009.
        M2(immutableArray.ToImmutableArray());
    }

    private void M2(ImmutableList<int> immutableArray)
    {
        Console.WriteLine(immutableArray.Length);
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Collections.Immutable;

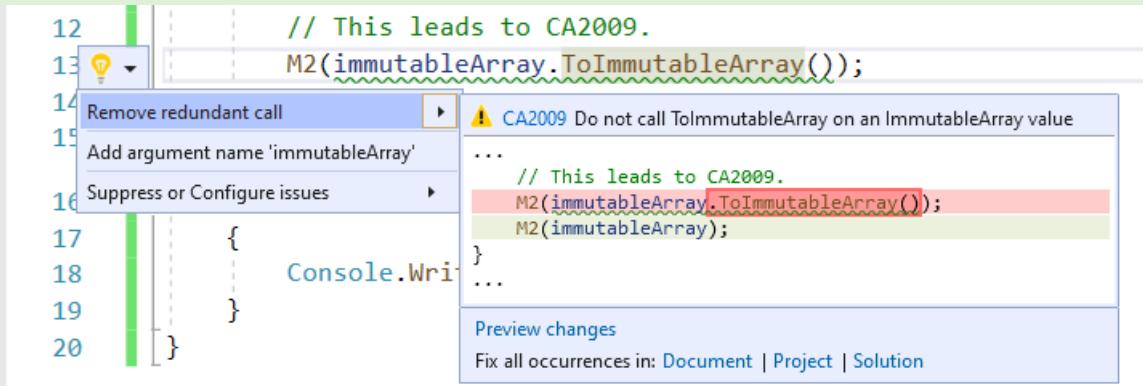
public class C
{
    public void M(IEnumerable<int> collection, ImmutableList<int> immutableArray)
    {
        // This is fine.
        M2(collection.ToImmutableArray());

        // This is now fine.
        M2(immutableArray);
    }

    private void M2(ImmutableList<int> immutableArray)
    {
        Console.WriteLine(immutableArray.Length);
    }
}
```

TIP

A code fix is available for this rule in Visual Studio. To use it, position the cursor on the violation and press **Ctrl+.** (period). Choose **Remove redundant call** from the list of options that's presented.



When to suppress warnings

Do not suppress violations from this rule, unless you're not concerned about the performance impact from unnecessary allocations of immutable collections.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA2009
// The code that's violating the rule is on this line.
#pragma warning restore CA2009
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA2009.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Reliability.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Reliability rules](#)
- [Performance rules](#)

CA2011: Do not assign property within its setter

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA2011
Category	Reliability
Fix is breaking or non-breaking	Non-breaking

Cause

A property was accidentally assigned a value within its own [set accessor](#).

Rule description

Assigning a property to itself in its [set accessor](#) leads to an infinite chain of recursive calls to the set accessor. This results in a [StackOverflowException](#) at run time. Such a mistake is common when the property and the backing field to store the property value have similar names. Instead of assigning the value to the backing field, it was accidentally assigned to the property itself.

How to fix violations

To fix violations, replace the violating assignment to the property with either an assignment to the backing field or switch to using an [auto-property](#). For example, the following code snippet shows a violation of the rule and a couple of ways to fix it:

```
public class C
{
    // Backing field for property 'P'
    private int p;

    public int P
    {
        get
        {
            return p;
        }
        set
        {
            // CA2011: Accidentally assigned to property, instead of the backing field.
            P = value;
        }
    }
}
```

```
public class C
{
    // Backing field for property 'P'
    private int _p;

    public int P
    {
        get
        {
            return _p;
        }
        set
        {
            // Option 1: Assign to backing field and rename the backing field for clarity.
            _p = value;
        }
    }
}
```

```
public class C
{
    // Option 2: Use auto-property.
    public int P { get; set; }
}
```

When to suppress warnings

It is fine to suppress violations from this rule if you are sure that the recursive calls to the set accessor are conditionally guarded to prevent infinite recursion.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA2011
// The code that's violating the rule is on this line.
#pragma warning restore CA2011
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA2011.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Reliability.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Related rules

- [CA2245: Do not assign a property to itself](#)

See also

- [Reliability rules](#)

CA2012: Use ValueTasks correctly

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA2012
Category	Reliability
Fix is breaking or non-breaking	Non-breaking

Cause

A [ValueTask](#) instance returned from a member invocation is used in a way that could lead to exceptions, corruption, or poor performance.

Rule description

[ValueTask](#) instances returned from member invocations are intended to be directly awaited. Attempts to consume a ValueTask multiple times or to directly access one's result before it's known to be completed may result in an exception or corruption. Ignoring such a ValueTask is likely an indication of a functional bug and may degrade performance.

How to fix violations

In general, ValueTasks should be directly awaited rather than discarded or stored into other locations like local variables or fields.

When to suppress warnings

For `valueTask` objects returned from arbitrary member calls, the caller needs to assume that the `ValueTask` must be consumed (for example, awaited) once and only once. However, if the developer also controls the member being invoked and has complete knowledge of its implementation, the developer may know it's safe to suppress the warning, for example, if the return `ValueTask` always wraps a [Task](#) object.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA2012
// The code that's violating the rule is on this line.
#pragma warning restore CA2012
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA2012.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]\ndotnet_analyzer_diagnostic.category-Reliability.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Reliability rules](#)

CA2013: Do not use ReferenceEquals with value types

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA2013
Category	Reliability
Fix is breaking or non-breaking	Non-breaking

Cause

Using `System.Object.ReferenceEquals` method to test one or more value types for equality.

Rule description

When comparing values using `ReferenceEquals`, if `objA` and `objB` are value types, they are boxed before they are passed to the `ReferenceEquals` method. This means that even if both `objA` and `objB` represent the same instance of a value type, the `ReferenceEquals` method nevertheless returns false, as the following example shows.

How to fix violations

To fix the violation, replace it with a more appropriate equality check such as `==`.

```
int int1 = 1, int2 = 1;

// Violation occurs, returns false.
Console.WriteLine(Object.ReferenceEquals(int1, int2)); // false

// Use appropriate equality operator or method instead
Console.WriteLine(int1 == int2); // true
Console.WriteLine(object.Equals(int1, int2)); // true
```

When to suppress warnings

It is not safe to suppress a warning from this rule. We recommend using the more appropriate equality operator, such as `==`.

Related rules

- [CA2231: Overload operator equals on overriding ValueType.Equals](#)

See also

- [Reliability rules](#)

CA2014: Do not use stackalloc in loops

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA2014
Category	Reliability
Fix is breaking or non-breaking	Non-breaking

Cause

Using the C# `stackalloc` expression inside of a loop.

Rule description

The C# `stackalloc` expression allocates memory from the current stack frame, and that memory may not be released until the current method call returns. If `stackalloc` is used in a loop, it can lead to stack overflows due to exhausting the stack memory.

How to fix violations

Move the `stackalloc` expression outside of all loops in the method.

When to suppress warnings

It may be safe to suppress the warning when the containing loop or loops are invoked only a finite number of times, such that the overall amount of memory allocated across all `stackalloc` operations is known to be relatively small.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA2014
// The code that's violating the rule is on this line.
#pragma warning restore CA2014
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.cs,vb]
dotnet_diagnostic.CA2014.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]\ndotnet_analyzer_diagnostic.category-Reliability.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Reliability rules](#)

CA2015: Do not define finalizers for types derived from MemoryManager<T>

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA2015
Category	Reliability
Fix is breaking or non-breaking	Non-breaking

Cause

Defining finalizers for types derived from [MemoryManager<T>](#)

Rule description

Adding a finalizer to a type derived from [MemoryManager<T>](#) is likely an indication of a bug, as it suggests a native resource that could have been handed out in a [Span<T>](#) is getting cleaned up and potentially while it is still in use by the [Span<T>](#).

NOTE

The [MemoryManager<T>](#) class is intended for advanced scenarios. Most developers do not need to use it.

How to fix violations

To fix the violation, remove the finalizer definition.

```
class DerivedClass <T> : MemoryManager<T>
{
    public override bool Dispose(bool disposing)
    {
        if (disposing)
        {
            _handle.Dispose();
        }
    }

    ...

    // Violation occurs, remove the finalizer to fix the warning.
    ~DerivedClass() => Dispose(false);
}
```

When to suppress warnings

It is safe to suppress a violation of this rule if the intent is to create a finalizer for debugging or validation purposes.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA2015
// The code that's violating the rule is on this line.
#pragma warning restore CA2015
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.cs,vb]
dotnet_diagnostic.CA2015.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.cs,vb]
dotnet_analyzer_diagnostic.category-Reliability.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Related rules

- [CA1821: Remove empty finalizers](#)

See also

- [Reliability rules](#)

CA2016: Forward the CancellationToken parameter to methods that take one

9/20/2022 • 4 minutes to read • [Edit Online](#)

	VALUE
Type name	ForwardCancellationTokenToInvocations
Rule ID	CA2016
Category	Reliability
Fix is breaking or non-breaking	Non-breaking

Cause

This rule locates method invocations that could accept a [CancellationToken](#) parameter, but are not passing any, and suggests to forward the parent method's `CancellationToken` to them.

Rule description

This rule analyzes method definitions that take a `CancellationToken` as their last parameter, then analyzes all methods invoked in its body. If any of the method invocations can either accept a `CancellationToken` as the last parameter, or have an overload that takes a `CancellationToken` as the last parameter, then the rule suggests using that option instead to ensure that the cancellation notification gets propagated to all operations that can listen to it.

NOTE

Rule CA2016 is available in all .NET versions where the `CancellationToken` type is available. For the applicable versions, see the [CancellationToken "Applies to"](#) section.

How to fix violations

You can either fix violations manually, or use the code fix available in Visual Studio. Hover the light bulb that appears next to the method invocation and select the suggested change.

The following example shows two suggested changes:

The screenshot shows the Visual Studio IDE. The top part displays the code for `Program.cs`. The code defines a class `MyTestClass` with three static methods: `MyMethodWithDefault`, `MyMethodWithOverload`, and `MyMethod`. The `MyMethodWithDefault` method takes a `CancellationToken` parameter. The `MyMethodWithOverload` method also takes a `CancellationToken` parameter. The `MyMethod` method takes a `CancellationToken` parameter and calls the other two methods. The bottom part shows the `Error List - Current Document (Program.cs)` window, which contains two warning entries for rule CA2016:

Line	Code	Description
22	CA2016	Forward the 'c' parameter to the 'MyMethodWithDefault' method or pass in 'CancellationToken.None' explicitly to indicate intentionally not propagating the token.
23	CA2016	Forward the 'c' parameter to the 'MyMethodWithOverload' method or pass in 'CancellationToken.None' explicitly to indicate intentionally not propagating the token.

It's safe to suppress a violation of this rule if you're not concerned about forwarding the canceled operation notification to lower method invocations. You can also explicitly pass `default` in C# (`Nothing` in Visual Basic) or `None` to suppress the rule violation.

The rule can detect a variety of violations. The following examples show cases that the rule can detect:

Example 1

The rule will suggest forwarding the `c` parameter from `MyMethod` to the `MyMethodWithDefault` invocation, because the method defines an optional token parameter:

```

using System.Threading;

namespace ConsoleApp
{
    public static class MyTestClass
    {
        public static void MyMethodWithDefault(CancellationToken ct = default)
        {
        }

        public static void MyMethod(CancellationToken c)
        {
            MyMethodWithDefault();
        }
    }
}

```

Fix:

Forward the `c` parameter:

```

public static void MyMethod(CancellationToken c)
{
    MyMethodWithDefault(c);
}

```

If you are not concerned about forwarding cancellation notifications to lower invocations, you can either:

Explicitly pass `default`:

```

public static void MyMethod(CancellationToken c)
{
    MyMethodWithDefault(default);
}

```

Or explicitly pass `CancellationToken.None`:

```

public static void MyMethod(CancellationToken c)
{
    MyMethodWithDefault(CancellationToken.None);
}

```

Example 2

The rule will suggest forwarding the `c` parameter from `MyMethod` to the `MyMethodWithOverload` invocation, because the method has an overload that takes a `CancellationToken` parameter:

```
using System.Threading;

namespace ConsoleApp
{
    public static class MyTestClass
    {
        public static void MyMethodWithOverload()
        {
        }

        public static void MyMethodWithOverload(CancellationToken ct = default)
        {
        }

        public static void MyMethod(CancellationToken c)
        {
            MyMethodWithOverload();
        }
    }
}
```

Fix:

Forward the `c` parameter:

```
public static void MyMethod(CancellationToken c)
{
    MyMethodWithOverload(c);
}
```

If you are not concerned about forwarding cancellation notifications to lower invocations, you can either:

Explicitly pass `default`:

```
public static void MyMethod(CancellationToken c)
{
    MyMethodWithOverload(default);
}
```

Or explicitly pass `CancellationToken.None`:

```
public static void MyMethod(CancellationToken c)
{
    MyMethodWithOverload(CancellationToken.None);
}
```

Non-violation examples

The `cancellationToken` parameter in the parent method is not in the last position:

```

using System.Threading;

namespace ConsoleApp
{
    public static class MyTestClass
    {
        public static void MyMethodWithDefault(CancellationToken ct = default)
        {
        }

        public static void MyMethod(CancellationToken c, int lastParameter)
        {
            MyMethodWithDefault();
        }
    }
}

```

The `cancellationToken` parameter in the default method is not in the last position:

```

using System.Threading;

namespace ConsoleApp
{
    public static class MyTestClass
    {
        public static void MyMethodWithDefault(CancellationToken ct = default, int lastParameter = 0)
        {
        }

        public static void MyMethod(CancellationToken c)
        {
            MyMethodWithDefault();
        }
    }
}

```

The `cancellationToken` parameter in the overload method is not in the last position:

```

using System.Threading;

namespace ConsoleApp
{
    public static class MyTestClass
    {
        public static void MyMethodWithOverload(int lastParameter)
        {
        }

        public static void MyMethodWithOverload(CancellationToken ct, int lastParameter)
        {
        }

        public static void MyMethod(CancellationToken c)
        {
            MyMethodWithOverload();
        }
    }
}

```

The parent method defines more than one `CancellationToken` parameter:

```
using System.Threading;

namespace ConsoleApp
{
    public static class MyTestClass
    {
        public static void MyMethodWithDefault(CancellationToken ct = default)
        {
        }

        public static void MyMethod(CancellationToken c1, CancellationToken c2)
        {
            MyMethodWithDefault();
        }
    }
}
```

The method with defaults defines more than one `CancellationToken` parameter:

```
using System.Threading;

namespace ConsoleApp
{
    public static class MyTestClass
    {
        public static void MyMethodWithDefault(CancellationToken c1 = default, CancellationToken c2 = default)
        {
        }

        public static void MyMethod(CancellationToken c)
        {
            MyMethodWithDefault();
        }
    }
}
```

The method overload defines more than one `CancellationToken` parameter:

```
using System.Threading;

namespace ConsoleApp
{
    public static class MyTestClass
    {
        public static void MyMethodWithOverload(CancellationToken c1, CancellationToken c2)
        {
        }

        public static void MyMethodWithOverload()
        {
        }

        public static void MyMethod(CancellationToken c)
        {
            MyMethodWithOverload();
        }
    }
}
```

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA2016
// The code that's violating the rule is on this line.
#pragma warning restore CA2016
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA2016.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Reliability.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

CA2017: Parameter count mismatch

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA2017
Category	Reliability
Fix is breaking or non-breaking	Non-breaking

Cause

The number of parameters supplied in the logging message template does not match the number of named placeholders.

Rule description

This rule flags logger calls that have an incorrect number of message arguments.

How to fix violations

Match the number of placeholders in the template format with the number of passed arguments.

When to suppress warnings

Do not suppress a warning from this rule.

See also

- [Reliability rules](#)

CA2018: The `count` argument to `Buffer.BlockCopy` should specify the number of bytes to copy

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA2018
Category	Reliability
Fix is breaking or non-breaking	Non-breaking

Cause

This rule fires when `Array.Length` is used for the `count` argument of `Buffer.BlockCopy` on arrays whose elements are larger than one byte in size.

Rule description

When using `Buffer.BlockCopy`, the `count` argument specifies the number of bytes to copy. You should only use `Array.Length` for the `count` argument on arrays whose elements are exactly one byte in size. `byte`, `sbyte`, and `bool` arrays have elements that are one byte in size.

How to fix violations

Specify the number of bytes that you intend to copy for the `count` argument.

Example

Violation:

```
using System;
class Program
{
    static void Main()
    {
        int[] src = new int[] {1, 2, 3, 4};
        int[] dst = new int[] {0, 0, 0, 0};

        Buffer.BlockCopy(src, 0, dst, 0, src.Length);
    }
}
```

Fix:

If your array's elements are larger than one byte in size, you can multiply the length of the array by the element size to get the number of bytes.

```
using System;
class Program
{
    static void Main()
    {
        int[] src = new int[] {1, 2, 3, 4};
        int[] dst = new int[] {0, 0, 0, 0};

        Buffer.BlockCopy(src, 0, dst, 0, src.Length * sizeof(int));
    }
}
```

When to suppress warnings

It is generally NOT safe to suppress a warning from this rule.

See also

- [Reliability rules](#)

Security rules

9/20/2022 • 18 minutes to read • [Edit Online](#)

Security rules support safer libraries and applications. These rules help prevent security flaws in your program. If you disable any of these rules, you should clearly mark the reason in code and also inform the designated security officer for your development project.

In this section

RULE	DESCRIPTION
CA2100: Review SQL queries for security vulnerabilities	A method sets the System.Data.IDbCommand.CommandText property by using a string that is built from a string argument to the method. This rule assumes that the string argument contains user input. A SQL command string built from user input is vulnerable to SQL injection attacks.
CA2109: Review visible event handlers	A public or protected event-handling method was detected. Event-handling methods should not be exposed unless absolutely necessary.
CA2119: Seal methods that satisfy private interfaces	An inheritable public type provides an overridable method implementation of an internal (Friend in Visual Basic) interface. To fix a violation of this rule, prevent the method from being overridden outside the assembly.
CA2153: Avoid Handling Corrupted State Exceptions	Corrupted State Exceptions (CSE) indicate that memory corruption exists in your process. Catching these rather than allowing the process to crash can lead to security vulnerabilities if an attacker can place an exploit into the corrupted memory region.
CA2300: Do not use insecure deserializer BinaryFormatter	Insecure deserializers are vulnerable when deserializing untrusted data. An attacker could modify the serialized data to include unexpected types to inject objects with malicious side effects.
CA2301: Do not call BinaryFormatter.Deserialize without first setting BinaryFormatter.Binder	Insecure deserializers are vulnerable when deserializing untrusted data. An attacker could modify the serialized data to include unexpected types to inject objects with malicious side effects.
CA2302: Ensure BinaryFormatter.Binder is set before calling BinaryFormatter.Deserialize	Insecure deserializers are vulnerable when deserializing untrusted data. An attacker could modify the serialized data to include unexpected types to inject objects with malicious side effects.
CA2305: Do not use insecure deserializer LosFormatter	Insecure deserializers are vulnerable when deserializing untrusted data. An attacker could modify the serialized data to include unexpected types to inject objects with malicious side effects.

Rule	Description
CA2310: Do not use insecure deserializer NetDataContractSerializer	Insecure deserializers are vulnerable when deserializing untrusted data. An attacker could modify the serialized data to include unexpected types to inject objects with malicious side effects.
CA2311: Do not deserialize without first setting NetDataContractSerializer.Binder	Insecure deserializers are vulnerable when deserializing untrusted data. An attacker could modify the serialized data to include unexpected types to inject objects with malicious side effects.
CA2312: Ensure NetDataContractSerializer.Binder is set before deserializing	Insecure deserializers are vulnerable when deserializing untrusted data. An attacker could modify the serialized data to include unexpected types to inject objects with malicious side effects.
CA2315: Do not use insecure deserializer ObjectStateFormatter	Insecure deserializers are vulnerable when deserializing untrusted data. An attacker could modify the serialized data to include unexpected types to inject objects with malicious side effects.
CA2321: Do not deserialize with JavaScriptSerializer using a SimpleTypeResolver	Insecure deserializers are vulnerable when deserializing untrusted data. An attacker could modify the serialized data to include unexpected types to inject objects with malicious side effects.
CA2322: Ensure JavaScriptSerializer is not initialized with SimpleTypeResolver before deserializing	Insecure deserializers are vulnerable when deserializing untrusted data. An attacker could modify the serialized data to include unexpected types to inject objects with malicious side effects.
CA2326: Do not use TypeNameHandling values other than None	Insecure deserializers are vulnerable when deserializing untrusted data. An attacker could modify the serialized data to include unexpected types to inject objects with malicious side effects.
CA2327: Do not use insecure JsonSerializerSettings	Insecure deserializers are vulnerable when deserializing untrusted data. An attacker could modify the serialized data to include unexpected types to inject objects with malicious side effects.
CA2328: Ensure that JsonSerializerSettings are secure	Insecure deserializers are vulnerable when deserializing untrusted data. An attacker could modify the serialized data to include unexpected types to inject objects with malicious side effects.
CA2329: Do not deserialize with JsonSerializer using an insecure configuration	Insecure deserializers are vulnerable when deserializing untrusted data. An attacker could modify the serialized data to include unexpected types to inject objects with malicious side effects.
CA2330: Ensure that JsonSerializer has a secure configuration when deserializing	Insecure deserializers are vulnerable when deserializing untrusted data. An attacker could modify the serialized data to include unexpected types to inject objects with malicious side effects.

Rule	Description
CA2350: Ensure <code>DataTable.ReadXml()</code> 's input is trusted	When deserializing a <code>DataTable</code> with untrusted input, an attacker can craft malicious input to perform a denial of service attack. There may be unknown remote code execution vulnerabilities.
CA2351: Ensure <code>DataSet.ReadXml()</code> 's input is trusted	When deserializing a <code>DataSet</code> with untrusted input, an attacker can craft malicious input to perform a denial of service attack. There may be unknown remote code execution vulnerabilities.
CA2352: Unsafe <code>DataSet</code> or <code>DataTable</code> in serializable type can be vulnerable to remote code execution attacks	A class or struct marked with <code>SerializableAttribute</code> contains a <code>DataSet</code> or <code>DataTable</code> field or property, and doesn't have a <code>GeneratedCodeAttribute</code> .
CA2353: Unsafe <code>DataSet</code> or <code>DataTable</code> in serializable type	A class or struct marked with an XML serialization attribute or a data contract attribute contains a <code>DataSet</code> or <code>DataTable</code> field or property.
CA2354: Unsafe <code>DataSet</code> or <code>DataTable</code> in deserialized object graph can be vulnerable to remote code execution attack	Deserializing with an <code>System.Runtime.Serialization.IFormatter</code> serialized, and the casted type's object graph can include a <code>DataSet</code> or <code>DataTable</code> .
CA2355: Unsafe <code>DataSet</code> or <code>DataTable</code> in deserialized object graph	Deserializing when the casted or specified type's object graph can include a <code>DataSet</code> or <code>DataTable</code> .
CA2356: Unsafe <code>DataSet</code> or <code>DataTable</code> in web deserialized object graph	A method with a <code>System.Web.Services.WebMethodAttribute</code> or <code>System.ServiceModel.OperationContractAttribute</code> has a parameter that may reference a <code>DataSet</code> or <code>DataTable</code> .
CA2361: Ensure autogenerated class containing <code>DataSet.ReadXml()</code> is not used with untrusted data	When deserializing a <code>DataSet</code> with untrusted input, an attacker can craft malicious input to perform a denial of service attack. There may be unknown remote code execution vulnerabilities.
CA2362: Unsafe <code>DataSet</code> or <code>DataTable</code> in autogenerated serializable type can be vulnerable to remote code execution attacks	When deserializing untrusted input with <code>BinaryFormatter</code> and the deserialized object graph contains a <code>DataSet</code> or <code>DataTable</code> , an attacker can craft a malicious payload to perform a remote code execution attack.
CA3001: Review code for SQL injection vulnerabilities	When working with untrusted input and SQL commands, be mindful of SQL injection attacks. An SQL injection attack can execute malicious SQL commands, compromising the security and integrity of your application.
CA3002: Review code for XSS vulnerabilities	When working with untrusted input from web requests, be mindful of cross-site scripting (XSS) attacks. An XSS attack injects untrusted input into raw HTML output, allowing the attacker to execute malicious scripts or maliciously modify content in your web page.
CA3003: Review code for file path injection vulnerabilities	When working with untrusted input from web requests, be mindful of using user-controlled input when specifying paths to files.

Rule	Description
CA3004: Review code for information disclosure vulnerabilities	Disclosing exception information gives attackers insight into the internals of your application, which can help attackers find other vulnerabilities to exploit.
CA3006: Review code for process command injection vulnerabilities	When working with untrusted input, be mindful of command injection attacks. A command injection attack can execute malicious commands on the underlying operating system, compromising the security and integrity of your server.
CA3007: Review code for open redirect vulnerabilities	When working with untrusted input, be mindful of open redirect vulnerabilities. An attacker can exploit an open redirect vulnerability to use your website to give the appearance of a legitimate URL, but redirect an unsuspecting visitor to a phishing or other malicious webpage.
CA3008: Review code for XPath injection vulnerabilities	When working with untrusted input, be mindful of XPath injection attacks. Constructing XPath queries using untrusted input may allow an attacker to maliciously manipulate the query to return an unintended result, and possibly disclose the contents of the queried XML.
CA3009: Review code for XML injection vulnerabilities	When working with untrusted input, be mindful of XML injection attacks.
CA3010: Review code for XAML injection vulnerabilities	When working with untrusted input, be mindful of XAML injection attacks. XAML is a markup language that directly represents object instantiation and execution. That means elements created in XAML can interact with system resources (for example, network access and file system IO).
CA3011: Review code for DLL injection vulnerabilities	When working with untrusted input, be mindful of loading untrusted code. If your web application loads untrusted code, an attacker may be able to inject malicious DLLs into your process and execute malicious code.
CA3012: Review code for regex injection vulnerabilities	When working with untrusted input, be mindful of regex injection attacks. An attacker can use regex injection to maliciously modify a regular expression, to make the regex match unintended results, or to make the regex consume excessive CPU resulting in a Denial of Service attack.
CA3061: Do not add schema by URL	Do not use the unsafe overload of the Add method because it may cause dangerous external references.
CA3075: Insecure DTD Processing	If you use insecure DTDProcessing instances or reference external entity sources, the parser may accept untrusted input and disclose sensitive information to attackers.
CA3076: Insecure XSLT Script Execution	If you execute Extensible StyleSheet Language Transformations (XSLT) in .NET applications insecurely, the processor may resolve untrusted URI references that could disclose sensitive information to attackers, leading to Denial of Service and Cross-Site attacks.

RULE	DESCRIPTION
CA3077: Insecure Processing in API Design, XML Document and XML Text Reader	When designing an API derived from <code>XMLDocument</code> and <code>XMLTextReader</code> , be mindful of <code>DtdProcessing</code> . Using insecure <code>DTDProcessing</code> instances when referencing or resolving external entity sources or setting insecure values in the XML may lead to information disclosure.
CA3147: Mark verb handlers with ValidateAntiForgeryToken	When designing an ASP.NET MVC controller, be mindful of cross-site request forgery attacks. A cross-site request forgery attack can send malicious requests from an authenticated user to your ASP.NET MVC controller.
CA5350: Do Not Use Weak Cryptographic Algorithms	Weak encryption algorithms and hashing functions are used today for a number of reasons, but they should not be used to guarantee the confidentiality or integrity of the data they protect. This rule triggers when it finds <code>TripleDES</code> , <code>SHA1</code> , or <code>RIPEMD160</code> algorithms in the code.
CA5351: Do Not Use Broken Cryptographic Algorithms	Broken cryptographic algorithms are not considered secure and their use should be strongly discouraged. This rule triggers when it finds the <code>MD5</code> hash algorithm or either the <code>DES</code> or <code>RC2</code> encryption algorithms in code.
CA5358: Do Not Use Unsafe Cipher Modes	Do Not Use Unsafe Cipher Modes
CA5359: Do not disable certificate validation	A certificate can help authenticate the identity of the server. Clients should validate the server certificate to ensure requests are sent to the intended server. If the <code>ServerCertificateValidationCallback</code> always returns <code>true</code> , any certificate will pass validation.
CA5360: Do not call dangerous methods in deserialization	Insecure deserialization is a vulnerability that occurs when untrusted data is used to abuse the logic of an application, inflict a Denial-of-Service (DoS) attack, or even execute arbitrary code upon it being deserialized. It's frequently possible for malicious users to abuse these deserialization features when the application is deserializing untrusted data that is under their control. Specifically, invoke dangerous methods in the process of deserialization. Successful insecure deserialization attacks could allow an attacker to carry out attacks such as DoS attacks, authentication bypasses, and remote code execution.
CA5361: Do not disable SChannel use of strong crypto	<p>Setting <code>Switch.System.Net.DontEnableSchUseStrongCrypto</code> to <code>true</code> weakens the cryptography used in outgoing Transport Layer Security (TLS) connections. Weaker cryptography can compromise the confidentiality of communication between your application and the server, making it easier for attackers to eavesdrop sensitive data.</p>
CA5362: Potential reference cycle in deserialized object graph	If deserializing untrusted data, then any code processing the deserialized object graph needs to handle reference cycles without going into infinite loops. This includes both code that's part of a deserialization callback and code that processes the object graph after deserialization completed. Otherwise, an attacker could perform a Denial-of-Service attack with malicious data containing a reference cycle.

Rule	Description
CA5363: Do not disable request validation	Request validation is a feature in ASP.NET that examines HTTP requests and determines whether they contain potentially dangerous content that can lead to injection attacks, including cross-site-scripting.
CA5364: Do not use deprecated security protocols	Transport Layer Security (TLS) secures communication between computers, most commonly with Hypertext Transfer Protocol Secure (HTTPS). Older protocol versions of TLS are less secure than TLS 1.2 and TLS 1.3 and are more likely to have new vulnerabilities. Avoid older protocol versions to minimize risk.
CA5365: Do Not Disable HTTP Header Checking	HTTP header checking enables encoding of the carriage return and newline characters, \r and \n, that are found in response headers. This encoding can help to avoid injection attacks that exploit an application that echoes untrusted data contained by the header.
CA5366: Use XmlReader For DataSet Read XML	Using a DataSet to read XML with untrusted data may load dangerous external references, which should be restricted by using an XmlReader with a secure resolver or with DTD processing disabled.
CA5367: Do Not Serialize Types With Pointer Fields	This rule checks whether there's a serializable class with a pointer field or property. Members that can't be serialized can be a pointer, such as static members or fields marked with NonSerializedAttribute .
CA5368: Set ViewStateUserKey For Classes Derived From Page	Setting the ViewStateUserKey property can help you prevent attacks on your application by allowing you to assign an identifier to the view-state variable for individual users so that attackers cannot use the variable to generate an attack. Otherwise, there will be vulnerabilities to cross-site request forgery.
CA5369: Use XmlReader for Deserialize	Processing untrusted DTD and XML schemas may enable loading dangerous external references, which should be restricted by using an XmlReader with a secure resolver or with DTD and XML inline schema processing disabled.
CA5370: Use XmlReader for validating reader	Processing untrusted DTD and XML schemas may enable loading dangerous external references. This dangerous loading can be restricted by using an XmlReader with a secure resolver or with DTD and XML inline schema processing disabled.
CA5371: Use XmlReader for schema read	Processing untrusted DTD and XML schemas may enable loading dangerous external references. Using an XmlReader with a secure resolver or with DTD and XML inline schema processing disabled restricts this.
CA5372: Use XmlReader for XPathDocument	Processing XML from untrusted data may load dangerous external references, which can be restricted by using an XmlReader with a secure resolver or with DTD processing disabled.

RULE	DESCRIPTION
CA5373: Do not use obsolete key derivation function	<p>This rule detects the invocation of weak key derivation methods <code>System.Security.Cryptography.PasswordDeriveBytes</code> and <code>Rfc2898DeriveBytes.CryptDeriveKey</code>. <code>System.Security.Cryptography.PasswordDeriveBytes</code> used a weak algorithm PBKDF1.</p>
CA5374: Do Not Use XslTransform	<p>This rule checks if <code>System.Xml.Xsl.XslTransform</code> is instantiated in the code. <code>System.Xml.Xsl.XslTransform</code> is now obsolete and shouldn't be used.</p>
CA5375: Do not use account shared access signature	<p>An account SAS can delegate access to read, write, and delete operations on blob containers, tables, queues, and file shares that are not permitted with a service SAS. However, it doesn't support container-level policies and has less flexibility and control over the permissions that are granted. Once malicious users get it, your storage account will be compromised easily.</p>
CA5376: Use SharedAccessProtocol HttpsOnly	<p>SAS is sensitive data that can't be transported in plain text on HTTP.</p>
CA5377: Use container level access policy	<p>A container-level access policy can be modified or revoked at any time. It provides greater flexibility and control over the permissions that are granted.</p>
CA5378: Do not disable ServicePointManagerSecurityProtocols	<p>Setting <code>DisableUsingServicePointManagerSecurityProtocols</code> to <code>true</code> limits Windows Communication Framework's (WCF) Transport Layer Security (TLS) connections to using TLS 1.0. That version of TLS will be deprecated.</p>
CA5379: Ensure key derivation function algorithm is sufficiently strong	<p>The <code>Rfc2898DeriveBytes</code> class defaults to using the <code>SHA1</code> algorithm. You should specify the hash algorithm to use in some overloads of the constructor with <code>SHA256</code> or higher. Note, <code>HashAlgorithm</code> property only has a <code>get</code> accessor and doesn't have a <code>overridden</code> modifier.</p>
CA5380: Do not add certificates to root store	<p>This rule detects code that adds a certificate into the Trusted Root Certification Authorities certificate store. By default, the Trusted Root Certification Authorities certificate store is configured with a set of public CAs that has met the requirements of the Microsoft Root Certificate Program.</p>
CA5381: Ensure certificates are not added to root store	<p>This rule detects code that potentially adds a certificate into the Trusted Root Certification Authorities certificate store. By default, the Trusted Root Certification Authorities certificate store is configured with a set of public certification authorities (CAs) that has met the requirements of the Microsoft Root Certificate Program.</p>
CA5382: Use secure cookies in ASP.NET Core	<p>Applications available over HTTPS must use secure cookies, which indicate to the browser that the cookie should only be transmitted using Transport Layer Security (TLS).</p>

Rule	Description
CA5383: Ensure use secure cookies in ASP.NET Core	Applications available over HTTPS must use secure cookies, which indicate to the browser that the cookie should only be transmitted using Transport Layer Security (TLS).
CA5384: Do not use digital signature algorithm (DSA)	DSA is a weak asymmetric encryption algorithm.
CA5385: Use Rivest–Shamir–Adleman (RSA) algorithm with sufficient key size	An RSA key smaller than 2048 bits is more vulnerable to brute force attacks.
CA5386: Avoid hardcoding SecurityProtocolType value	Transport Layer Security (TLS) secures communication between computers, most commonly with Hypertext Transfer Protocol Secure (HTTPS). Protocol versions TLS 1.0 and TLS 1.1 are deprecated, while TLS 1.2 and TLS 1.3 are current. In the future, TLS 1.2 and TLS 1.3 may be deprecated. To ensure that your application remains secure, avoid hardcoding a protocol version and target at least .NET Framework v4.7.1.
CA5387: Do not use weak key derivation function with insufficient iteration count	This rule checks if a cryptographic key was generated by Rfc2898DeriveBytes with an iteration count of less than 100,000. A higher iteration count can help mitigate against dictionary attacks that try to guess the generated cryptographic key.
CA5388: Ensure sufficient iteration count when using weak key derivation function	This rule checks if a cryptographic key was generated by Rfc2898DeriveBytes with an iteration count that may be less than 100,000. A higher iteration count can help mitigate against dictionary attacks that try to guess the generated cryptographic key.
CA5389: Do not add archive item's path to the target file system path	File path can be relative and can lead to file system access outside of the expected file system target path, leading to malicious config changes and remote code execution via lay-and-wait technique.
CA5390: Do not hard-code encryption key	For a symmetric algorithm to be successful, the secret key must be known only to the sender and the receiver. When a key is hard-coded, it is easily discovered. Even with compiled binaries, it is easy for malicious users to extract it. Once the private key is compromised, the cipher text can be decrypted directly and is not protected anymore.
CA5391: Use antiforgery tokens in ASP.NET Core MVC controllers	Handling a <code>POST</code> , <code>PUT</code> , <code>PATCH</code> , or <code>DELETE</code> request without validating an antiforgery token may be vulnerable to cross-site request forgery attacks. A cross-site request forgery attack can send malicious requests from an authenticated user to your ASP.NET Core MVC controller.
CA5392: Use DefaultDllImportSearchPaths attribute for P/Invokes	By default, P/Invoke functions using DllImportAttribute probe a number of directories, including the current working directory for the library to load. This can be a security issue for certain applications, leading to DLL hijacking.
CA5393: Do not use unsafe DllImportSearchPath value	There could be a malicious DLL in the default DLL search directories and assembly directories. Or, depending on where your application is run from, there could be a malicious DLL in the application's directory.

Rule	Description
CA5394: Do not use insecure randomness	Using a cryptographically weak pseudo-random number generator may allow an attacker to predict what security-sensitive value will be generated.
CA5395: Miss <code>HttpVerb</code> attribute for action methods	All the action methods that create, edit, delete, or otherwise modify data needs to be protected with the <code>antiforgery</code> attribute from cross-site request forgery attacks. Performing a GET operation should be a safe operation that has no side effects and doesn't modify your persisted data.
CA5396: Set <code>HttpOnly</code> to true for <code>HttpCookie</code>	As a defense in depth measure, ensure security sensitive HTTP cookies are marked as <code>HttpOnly</code> . This indicates web browsers should disallow scripts from accessing the cookies. Injected malicious scripts are a common way of stealing cookies.
CA5397: Do not use deprecated <code>SslProtocols</code> values	Transport Layer Security (TLS) secures communication between computers, most commonly with Hypertext Transfer Protocol Secure (HTTPS). Older protocol versions of TLS are less secure than TLS 1.2 and TLS 1.3 and are more likely to have new vulnerabilities. Avoid older protocol versions to minimize risk.
CA5398: Avoid hardcoded <code>SslProtocols</code> values	Transport Layer Security (TLS) secures communication between computers, most commonly with Hypertext Transfer Protocol Secure (HTTPS). Protocol versions TLS 1.0 and TLS 1.1 are deprecated, while TLS 1.2 and TLS 1.3 are current. In the future, TLS 1.2 and TLS 1.3 may be deprecated. To ensure that your application remains secure, avoid hardcoding a protocol version.
CA5399: Definitely disable <code>HttpClient</code> certificate revocation list check	A revoked certificate isn't trusted anymore. It could be used by attackers passing some malicious data or stealing sensitive data in HTTPS communication.
CA5400: Ensure <code>HttpClient</code> certificate revocation list check is not disabled	A revoked certificate isn't trusted anymore. It could be used by attackers passing some malicious data or stealing sensitive data in HTTPS communication.
CA5401: Do not use <code>CreateEncryptor</code> with non-default IV	Symmetric encryption should always use a non-repeatable initialization vector to prevent dictionary attacks.
CA5402: Use <code>CreateEncryptor</code> with the default IV	Symmetric encryption should always use a non-repeatable initialization vector to prevent dictionary attacks.
CA5403: Do not hard-code certificate	The <code>data</code> or <code>rawData</code> parameter of a <code>X509Certificate</code> or <code>X509Certificate2</code> constructor is hard-coded.
CA5404: Do not disable token validation checks	<code>TokenValidationParameters</code> properties that control token validation should not be set to <code>false</code> .
CA5405: Do not always skip token validation in delegates	The callback assigned to <code>AudienceValidator</code> or <code>LifetimeValidator</code> always returns <code>true</code> .

CA2100: Review SQL queries for security vulnerabilities

9/20/2022 • 6 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA2100
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

A method sets the [System.Data.IDbCommand.CommandText](#) property by using a string that is built from a string argument to the method.

By default, this rule analyzes the entire codebase, but this is [configurable](#).

Rule description

This rule assumes that any string, whose value can't be determined at compile time, may contain user input. A SQL command string that is built from user input is vulnerable to SQL injection attacks. In a SQL injection attack, a malicious user supplies input that alters the design of a query in an attempt to damage or gain unauthorized access to the underlying database. Typical techniques include injection of a single quotation mark or apostrophe, which is the SQL literal string delimiter; two dashes, which signifies a SQL comment; and a semicolon, which indicates that a new command follows. If user input must be part of the query, use one of the following, listed in order of effectiveness, to reduce the risk of attack.

- Use a stored procedure.
- Use a parameterized command string.
- Validate the user input for both type and content before you build the command string.

The following .NET types implement the [CommandText](#) property or provide constructors that set the property by using a string argument.

- [System.Data.Odbc.OdbcCommand](#) and [System.Data.Odbc.OdbcDataAdapter](#)
- [System.Data.OleDb.OleDbCommand](#) and [System.Data.OleDb.OleDbDataAdapter](#)
- [System.Data.OracleClient.OracleCommand](#) and [System.Data.OracleClient.OracleDataAdapter](#)
- [System.Data.SqlClient.SqlCommand](#) and [System.Data.SqlClient.SqlDataAdapter](#)

In some cases, this rule might not determine a string's value at compile time, even though you can. In those cases, this rule produces false positives when using those strings as SQL commands. The following is an example of such a string.

```
int x = 10;
string query = "SELECT TOP " + x.ToString() + " FROM Table";
```

The same applies when using `ToString()` implicitly.

```
int x = 10;
string query = String.Format("SELECT TOP {0} FROM Table", x);
```

How to fix violations

To fix a violation of this rule, use a parameterized query.

When to suppress warnings

It is safe to suppress a warning from this rule if the command text does not contain any user input.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA2100
// The code that's violating the rule is on this line.
#pragma warning restore CA2100
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA2100.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

Use the following options to configure which parts of your codebase to run this rule on.

- [Exclude specific symbols](#)
- [Exclude specific types and their derived types](#)

You can configure these options for just this rule, for all rules, or for all rules in this category ([Security](#)). For more information, see [Code quality rule configuration options](#).

Exclude specific symbols

You can exclude specific symbols, such as types and methods, from analysis. For example, to specify that the rule should not run on any code within types named `MyType`, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Symbol name only (includes all symbols with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#). Each symbol name requires a symbol-kind prefix, such as `M:` for methods, `T:` for types, and `N:` for namespaces.
- `.ctor` for constructors and `.cctor` for static constructors.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType</code>	Matches all symbols named <code>MyType</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType1 MyType2</code>	Matches all symbols named either <code>MyType1</code> or <code>MyType2</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS.MyType.MyMethod(ParamType)</code>	Matches specific method <code>MyMethod</code> with the specified fully qualified signature.
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS1.MyType1.MyMethod1(ParamType) M:NS2.MyType2.MyMethod2(ParamType)</code>	Matches specific methods <code>MyMethod1</code> and <code>MyMethod2</code> with the respective fully qualified signatures.

Exclude specific types and their derived types

You can exclude specific types and their derived types from analysis. For example, to specify that the rule should not run on any methods within types named `MyType` and their derived types, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Type name only (includes all types with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#), with an optional `T:` prefix.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType</code>	Matches all types named <code>MyType</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType1 MyType2</code>	Matches all types named either <code>MyType1</code> or <code>MyType2</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS.MyType</code>	Matches specific type <code>MyType</code> with given fully qualified name and all of its derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS1.MyType1 M:NS2.MyType2</code>	Matches specific types <code>MyType1</code> and <code>MyType2</code> with the respective fully qualified names, and all of their derived types.

Example

The following example shows a method, `UnsafeQuery`, that violates the rule and a method, `SaferQuery`, that satisfies the rule by using a parameterized command string.

```

Imports System
Imports System.Data
Imports System.Data.SqlClient

Namespace ca2100

    Public Class SqlQueries

        Function UnsafeQuery(connection As String,
                             name As String, password As String) As Object

            Dim someConnection As New SqlConnection(connection)
            Dim someCommand As New SqlCommand()
            someCommand.Connection = someConnection

            someCommand.CommandText = "SELECT AccountNumber FROM Users " &
            "WHERE Username=''' & name & "' AND Password=''' & password & ''"

            someConnection.Open()
            Dim accountNumber As Object = someCommand.ExecuteScalar()
            someConnection.Close()
            Return accountNumber

        End Function

        Function SaferQuery(connection As String,
                             name As String, password As String) As Object

            Dim someConnection As New SqlConnection(connection)
            Dim someCommand As New SqlCommand()
            someCommand.Connection = someConnection

            someCommand.Parameters.Add(
                "@username", SqlDbType.NChar).Value = name
            someCommand.Parameters.Add(
                "@password", SqlDbType.NChar).Value = password
            someCommand.CommandText = "SELECT AccountNumber FROM Users " &
            "WHERE Username=@username AND Password=@password"

            someConnection.Open()
            Dim accountNumber As Object = someCommand.ExecuteScalar()
            someConnection.Close()
            Return accountNumber

        End Function

    End Class

    Class MaliciousCode

        Shared Sub Main2100(args As String())

            Dim queries As New SqlQueries()
            queries.UnsafeQuery(args(0), "' OR 1=1 --", "[PLACEHOLDER]")
            ' Resultant query (which is always true):
            ' SELECT AccountNumber FROM Users WHERE Username='' OR 1=1

            queries.SaferQuery(args(0), "' OR 1=1 --", "[PLACEHOLDER]")
            ' Resultant query (notice the additional single quote character):
            ' SELECT AccountNumber FROM Users WHERE Username=''' OR 1=1 --
            '                                         AND Password='[PLACEHOLDER]'

        End Sub

    End Class

End Namespace

```

```

public class SqlQueries
{
    public object UnsafeQuery(
        string connection, string name, string password)
    {
        SqlConnection someConnection = new SqlConnection(connection);
        SqlCommand someCommand = new SqlCommand();
        someCommand.Connection = someConnection;

        someCommand.CommandText = "SELECT AccountNumber FROM Users " +
            "WHERE Username=' " + name +
            "' AND Password=' " + password + "'";

        someConnection.Open();
        object accountNumber = someCommand.ExecuteScalar();
        someConnection.Close();
        return accountNumber;
    }

    public object SaferQuery(
        string connection, string name, string password)
    {
        SqlConnection someConnection = new SqlConnection(connection);
        SqlCommand someCommand = new SqlCommand();
        someCommand.Connection = someConnection;

        someCommand.Parameters.Add(
            "@username", SqlDbType.NChar).Value = name;
        someCommand.Parameters.Add(
            "@password", SqlDbType.NChar).Value = password;
        someCommand.CommandText = "SELECT AccountNumber FROM Users " +
            "WHERE Username=@username AND Password=@password";

        someConnection.Open();
        object accountNumber = someCommand.ExecuteScalar();
        someConnection.Close();
        return accountNumber;
    }
}

class MaliciousCode
{
    static void Main2100(string[] args)
    {
        SqlQueries queries = new SqlQueries();
        queries.UnsafeQuery(args[0], "' OR 1=1 --", "[PLACEHOLDER]");
        // Resultant query (which is always true):
        // SELECT AccountNumber FROM Users WHERE Username='' OR 1=1

        queries.SaferQuery(args[0], "' OR 1=1 --", "[PLACEHOLDER]");
        // Resultant query (notice the additional single quote character):
        // SELECT AccountNumber FROM Users WHERE Username=''' OR 1=1 --
        //                                         AND Password='[PLACEHOLDER]'
    }
}

```

See also

- [Security Overview](#)

CA2109: Review visible event handlers

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA2109
Category	Security
Fix is breaking or non-breaking	Breaking

Cause

A public or protected event-handling method was detected.

Rule description

An externally visible event-handling method presents a security issue that requires review.

Do not expose event-handling methods unless absolutely necessary. An event handler, a delegate type, that invokes the exposed method can be added to any event as long as the handler and event signatures match. Events can potentially be raised by any code, and are frequently raised by highly trusted system code in response to user actions such as clicking a button. Adding a security check to an event-handling method does not prevent code from registering an event handler that invokes the method.

A demand cannot reliably protect a method invoked by an event handler. Security demands help protect code from untrusted callers by examining the callers on the call stack. Code that adds an event handler to an event is not necessarily present on the call stack when the event handler's methods run. Therefore, the call stack might have only highly trusted callers when the event handler method is invoked. This causes demands made by the event handler method to succeed. Also, the demanded permission might be asserted when the method is invoked. For these reasons, the risk of not fixing a violation of this rule can only be assessed after reviewing the event-handling method. When you review your code, consider the following issues:

- Does your event handler perform any operations that are dangerous or exploitable, such as asserting permissions or suppressing unmanaged code permission?
- What are the security threats to and from your code because it can run at any time with only highly trusted callers on the stack?

How to fix violations

To fix a violation of this rule, review the method and evaluate the following:

- Can you make the event-handling method non-public?
- Can you move all dangerous functionality out of the event handler?
- If a security demand is imposed, can this be accomplished in some other manner?

When to suppress warnings

SUPPRESS A WARNING FROM THIS RULE ONLY AFTER A CAREFUL SECURITY REVIEW TO MAKE SURE THAT YOUR CODE DOES NOT POSE

a security threat.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA2109
// The code that's violating the rule is on this line.
#pragma warning restore CA2109
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA2109.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Example

The following code shows an event-handling method that can be misused by malicious code.

```
public class HandleEvents
{
    // Due to the access level and signature, a malicious caller could
    // add this method to system-triggered events where all code in the call
    // stack has the demanded permission.

    // Also, the demand might be canceled by an asserted permission.

    [SecurityPermissionAttribute(SecurityAction.Demand, UnmanagedCode = true)]

    // Violates rule: ReviewVisibleEventHandlers.
    public static void SomeActionHappened(Object sender, EventArgs e)
    {
        Console.WriteLine("Do something dangerous from unmanaged code.");
    }
}
```

See also

- [System.Security.CodeAccessPermission.Demand](#)
- [System.EventArgs](#)

CA2119: Seal methods that satisfy private interfaces

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA2119
Category	Security
Fix is breaking or non-breaking	Breaking

Cause

An inheritable public type provides an overridable method implementation of an `internal` (`Friend` in Visual Basic) interface.

Rule description

Interface methods have public accessibility, which cannot be changed by the implementing type. An internal interface creates a contract that is not intended to be implemented outside the assembly that defines the interface. A public type that implements a method of an internal interface using the `virtual` (`Overridable` in Visual Basic) modifier allows the method to be overridden by a derived type that is outside the assembly. If a second type in the defining assembly calls the method and expects an internal-only contract, behavior might be compromised when, instead, the overridden method in the outside assembly is executed. This creates a security vulnerability.

How to fix violations

To fix a violation of this rule, prevent the method from being overridden outside the assembly by using one of the following:

- Make the declaring type `sealed` (`NotInheritable` in Visual Basic).
- Change the accessibility of the declaring type to `internal` (`Friend` in Visual Basic).
- Remove all public constructors from the declaring type.
- Implement the method without using the `virtual` modifier.
- Implement the method explicitly.

When to suppress warnings

It is safe to suppress a warning from this rule if, after careful review, no security issues exist that might be exploitable if the method is overridden outside the assembly.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA2119
// The code that's violating the rule is on this line.
#pragma warning restore CA2119
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA2119.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Example 1

The following example shows a type, `BaseImplementation`, that violates this rule.

```
// Internal by default.
interface IValidate
{
    bool UserIsValidated();
}

public class BaseImplementation : IValidate
{
    public virtual bool UserIsValidated()
    {
        return false;
    }
}

public class UseBaseImplementation
{
    public void SecurityDecision(BaseImplementation someImplementation)
    {
        if (someImplementation.UserIsValidated() == true)
        {
            Console.WriteLine("Account number & balance.");
        }
        else
        {
            Console.WriteLine("Please login.");
        }
    }
}
```

```

Interface IValidate
    Function UserIsValidated() As Boolean
End Interface

Public Class BaseImplementation
    Implements IValidate

        Overridable Function UserIsValidated() As Boolean _
            Implements IValidate.UserIsValidated
            Return False
        End Function

    End Class

    Public Class UseBaseImplementation

        Sub SecurityDecision(someImplementation As BaseImplementation)

            If (someImplementation.UserIsValidated() = True) Then
                Console.WriteLine("Account number & balance.")
            Else
                Console.WriteLine("Please login.")
            End If

        End Sub

    End Class

```

Example 2

The following example exploits the virtual method implementation of the previous example.

```

public class BaseImplementation
{
    public virtual bool UserIsValidated()
    {
        return false;
    }
}

public class UseBaseImplementation
{
    public void SecurityDecision(BaseImplementation someImplementation)
    {
        if (someImplementation.UserIsValidated() == true)
        {
            Console.WriteLine("Account number & balance.");
        }
        else
        {
            Console.WriteLine("Please login.");
        }
    }
}

```

```
Public Class BaseImplementation

    Overridable Function UserIsValidated() As Boolean
        Return False
    End Function

End Class

Public Class UseBaseImplementation

    Sub SecurityDecision(someImplementation As BaseImplementation)

        If (someImplementation.UserIsValidated() = True) Then
            Console.WriteLine("Account number & balance.")
        Else
            Console.WriteLine("Please login.")
        End If

    End Sub

End Class
```

See also

- [Interfaces \(C#\)](#)
- [Interfaces \(Visual Basic\)](#)

CA2153: Avoid handling Corrupted State Exceptions

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA2153
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

Corrupted State Exceptions (CSEs) indicate that memory corruption exists in your process. Catching these rather than allowing the process to crash can lead to security vulnerabilities if an attacker can place an exploit into the corrupted memory region.

Rule description

CSE indicates that the state of a process has been corrupted and not caught by the system. In the corrupted state scenario, a general handler only catches the exception if you mark your method with the [System.Runtime.ExceptionServices.HandleProcessCorruptedStateExceptionsAttribute](#) attribute. By default, the [Common Language Runtime \(CLR\)](#) does not invoke catch handlers for CSEs.

The safest option is to allow the process to crash without catching these kinds of exceptions. Even logging code can allow attackers to exploit memory corruption bugs.

This warning triggers when catching CSEs with a general handler that catches all exceptions, for example,

`catch (System.Exception e)` or `catch` with no exception parameter.

How to fix violations

To resolve this warning, do one of the following:

- Remove the [HandleProcessCorruptedStateExceptionsAttribute](#) attribute. This reverts to the default runtime behavior where CSEs are not passed to catch handlers.
- Remove the general catch handler in preference of handlers that catch specific exception types. This may include CSEs, assuming the handler code can safely handle them (rare).
- Rethrow the CSE in the catch handler, which passes the exception to the caller and should result in ending the running process.

When to suppress warnings

Do not suppress a warning from this rule.

Pseudo-code example

Violation

The following pseudo-code illustrates the pattern detected by this rule.

```
[HandleProcessCorruptedStateExceptions]
// Method that handles CSE exceptions.
void TestMethod1()
{
    try
    {
        FileStream fileStream = new FileStream("name", FileMode.Create);
    }
    catch (Exception e)
    {
        // Handle exception.
    }
}
```

Solution 1 - remove the attribute

Removing the [HandleProcessCorruptedStateExceptionsAttribute](#) attribute ensures that Corrupted State Exceptions are not handled by your method.

```
void TestMethod1()
{
    try
    {
        FileStream fileStream = new FileStream("name", FileMode.Create);
    }
    catch (Exception e)
    {
        // Handle exception.
    }
}
```

Solution 2 - catch specific exceptions

Remove the general catch handler and catch only specific exception types.

```
void TestMethod1()
{
    try
    {
        FileStream fileStream = new FileStream("name", FileMode.Create);
    }
    catch (IOException e)
    {
        // Handle IOException.
    }
    catch (UnauthorizedAccessException e)
    {
        // Handle UnauthorizedAccessException.
    }
}
```

Solution 3 - rethrow

Rethrow the exception.

```
[HandleProcessCorruptedStateExceptions]
void TestMethod1()
{
    try
    {
        FileStream fileStream = new FileStream("name", FileMode.Create);
    }
    catch (Exception e)
    {
        // Rethrow the exception.
        throw;
    }
}
```

CA2300: Do not use insecure deserializer BinaryFormatter

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA2300
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

A `System.Runtime.Serialization.Formatters.Binary.BinaryFormatter` deserialization method was called or referenced.

Rule description

Insecure deserializers are vulnerable when deserializing untrusted data. An attacker could modify the serialized data to include unexpected types to inject objects with malicious side effects. An attack against an insecure deserializer could, for example, execute commands on the underlying operating system, communicate over the network, or delete files.

This rule finds `System.Runtime.Serialization.Formatters.Binary.BinaryFormatter` deserialization method calls or references. If you want to deserialize only when the `Binder` property is set to restrict types, disable this rule and enable rules [CA2301](#) and [CA2302](#) instead. Limiting which types can be deserialized can help mitigate against known remote code execution attacks, but your deserialization will still be vulnerable to denial of service attacks.

`BinaryFormatter` is insecure and can't be made secure. For more information, see the [BinaryFormatter security guide](#).

How to fix violations

- Use a secure serializer instead, and **don't allow an attacker to specify an arbitrary type to deserialize**. For more information see the [Preferred alternatives](#).
- Make the serialized data tamper-proof. After serialization, cryptographically sign the serialized data. Before deserialization, validate the cryptographic signature. Protect the cryptographic key from being disclosed and design for key rotations.
- This option makes code vulnerable to denial of service attacks and possible remote code execution attacks in the future. For more information, see the [BinaryFormatter security guide](#). Restrict serialized types. Implement a custom `System.Runtime.Serialization.SerializationBinder`. Before deserializing, set the `Binder` property to an instance of your custom `SerializationBinder` in all code paths. In the overridden `BindToType` method, if the type is unexpected, throw an exception to stop deserialization.

When to suppress warnings

`BinaryFormatter` is insecure and can't be made secure.

Pseudo-code examples

Violation

```
using System.IO;
using System.Runtime.Serialization.Formatters.Binary;

public class ExampleClass
{
    public object MyDeserialize(byte[] bytes)
    {
        BinaryFormatter formatter = new BinaryFormatter();
        return formatter.Deserialize(new MemoryStream(bytes));
    }
}
```

```
Imports System.IO
Imports System.Runtime.Serialization.Formatters.Binary

Public Class ExampleClass
    Public Function MyDeserialize(bytes As Byte()) As Object
        Dim formatter As BinaryFormatter = New BinaryFormatter()
        Return formatter.Deserialize(New MemoryStream(bytes))
    End Function
End Class
```

Related rules

[CA2301: Do not call BinaryFormatter.Deserialize without first setting BinaryFormatter.Binder](#)

[CA2302: Ensure BinaryFormatter.Binder is set before calling BinaryFormatter.Deserialize](#)

CA2301: Do not call BinaryFormatter.Deserialize without first setting BinaryFormatter.Binder

9/20/2022 • 5 minutes to read • [Edit Online](#)

	Value
Rule ID	CA2301
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

A `System.Runtime.Serialization.Formatters.Binary.BinaryFormatter` deserialization method was called or referenced without the `Binder` property set.

By default, this rule analyzes the entire codebase, but this is [configurable](#).

WARNING

Restricting types with a `SerializationBinder` can't prevent all attacks. For more information, see the [BinaryFormatter security guide](#).

Rule description

Insecure deserializers are vulnerable when deserializing untrusted data. An attacker could modify the serialized data to include unexpected types to inject objects with malicious side effects. An attack against an insecure deserializer could, for example, execute commands on the underlying operating system, communicate over the network, or delete files.

This rule finds `System.Runtime.Serialization.Formatters.Binary.BinaryFormatter` deserialization method calls or references, when `BinaryFormatter` doesn't have its `Binder` set. If you want to disallow any deserialization with `BinaryFormatter` regardless of the `Binder` property, disable this rule and [CA2302](#), and enable rule [CA2300](#).

How to fix violations

- Use a secure serializer instead, and **don't allow an attacker to specify an arbitrary type to deserialize**. For more information see the [Preferred alternatives](#).
- Make the serialized data tamper-proof. After serialization, cryptographically sign the serialized data. Before deserialization, validate the cryptographic signature. Protect the cryptographic key from being disclosed and design for key rotations.
- This option makes code vulnerable to denial of service attacks and possible remote code execution attacks in the future. For more information, see the [BinaryFormatter security guide](#). Restrict serialized types. Implement a custom `System.Runtime.Serialization.SerializationBinder`. Before deserializing, set the `Binder` property to an instance of your custom `SerializationBinder` in all code paths. In the overridden `BindToType` method, if the type is unexpected, throw an exception to stop deserialization.

When to suppress warnings

`BinaryFormatter` is insecure and can't be made secure.

Configure code to analyze

Use the following options to configure which parts of your codebase to run this rule on.

- [Exclude specific symbols](#)
- [Exclude specific types and their derived types](#)

You can configure these options for just this rule, for all rules, or for all rules in this category ([Security](#)). For more information, see [Code quality rule configuration options](#).

Exclude specific symbols

You can exclude specific symbols, such as types and methods, from analysis. For example, to specify that the rule should not run on any code within types named `MyType`, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Symbol name only (includes all symbols with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#). Each symbol name requires a symbol-kind prefix, such as `M:` for methods, `T:` for types, and `N:` for namespaces.
- `.ctor` for constructors and `.cctor` for static constructors.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType</code>	Matches all symbols named <code>MyType</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType1 MyType2</code>	Matches all symbols named either <code>MyType1</code> or <code>MyType2</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS.MyType.MyMethod(ParamType)</code>	Matches specific method <code>MyMethod</code> with the specified fully qualified signature.
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS1.MyType1.MyMethod1(ParamType) M:NS2.MyType2.MyMethod2(ParamType)</code>	Matches specific methods <code>MyMethod1</code> and <code>MyMethod2</code> with the respective fully qualified signatures.

Exclude specific types and their derived types

You can exclude specific types and their derived types from analysis. For example, to specify that the rule should not run on any methods within types named `MyType` and their derived types, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Type name only (includes all types with the name, regardless of the containing type or namespace).

- Fully qualified names in the symbol's [documentation ID format](#), with an optional `T:` prefix.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType</code>	Matches all types named <code>MyType</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType1 MyType2</code>	Matches all types named either <code>MyType1</code> or <code>MyType2</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS.MyType</code>	Matches specific type <code>MyType</code> with given fully qualified name and all of its derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS1.MyType1 M:NS2.MyType2</code>	Matches specific types <code>MyType1</code> and <code>MyType2</code> with the respective fully qualified names, and all of their derived types.

Pseudo-code examples

Violation

```
using System;
using System.IO;
using System.Runtime.Serialization.Formatters.Binary;

[Serializable]
public class BookRecord
{
    public string Title { get; set; }
    public AisleLocation Location { get; set; }
}

[Serializable]
public class AisleLocation
{
    public char Aisle { get; set; }
    public byte Shelf { get; set; }
}

public class ExampleClass
{
    public BookRecord DeserializeBookRecord(byte[] bytes)
    {
        BinaryFormatter formatter = new BinaryFormatter();
        using (MemoryStream ms = new MemoryStream(bytes))
        {
            return (BookRecord) formatter.Deserialize(ms);
        }
    }
}
```

```
Imports System
Imports System.IO
Imports System.Runtime.Serialization.Formatters.Binary

<Serializable()
Public Class BookRecord
    Public Property Title As String
    Public Property Location As AisleLocation
End Class

<Serializable()
Public Class AisleLocation
    Public Property Aisle As Char
    Public Property Shelf As Byte
End Class

Public Class ExampleClass
    Public Function DeserializeBookRecord(bytes As Byte()) As BookRecord
        Dim formatter As BinaryFormatter = New BinaryFormatter()
        Using ms As MemoryStream = New MemoryStream(bytes)
            Return CType(formatter.Deserialize(ms), BookRecord)
        End Using
    End Function
End Class
```

Solution

```

using System;
using System.IO;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;

public class BookRecordSerializationBinder : SerializationBinder
{
    public override Type BindToType(string assemblyName, string typeName)
    {
        // One way to discover expected types is through testing deserialization
        // of **valid** data and logging the types used.

        ////Console.WriteLine($"BindToType('{assemblyName}', '{typeName}')");

        if (typeName == "BookRecord" || typeName == "AisleLocation")
        {
            return null;
        }
        else
        {
            throw new ArgumentException("Unexpected type", nameof(typeName));
        }
    }
}

[Serializable]
public class BookRecord
{
    public string Title { get; set; }
    public AisleLocation Location { get; set; }
}

[Serializable]
public class AisleLocation
{
    public char Aisle { get; set; }
    public byte Shelf { get; set; }
}

public class ExampleClass
{
    public BookRecord DeserializeBookRecord(byte[] bytes)
    {
        BinaryFormatter formatter = new BinaryFormatter();
        formatter.Binder = new BookRecordSerializationBinder();
        using (MemoryStream ms = new MemoryStream(bytes))
        {
            return (BookRecord) formatter.Deserialize(ms);
        }
    }
}

```

```

Imports System
Imports System.IO
Imports System.Runtime.Serialization
Imports System.Runtime.Serialization.Formatters.Binary

Public Class BookRecordSerializationBinder
    Inherits SerializationBinder

    Public Overrides Function BindToType(assemblyName As String, typeName As String) As Type
        ' One way to discover expected types is through testing deserialization
        ' of **valid** data and logging the types used.

        'Console.WriteLine($"BindToType('{assemblyName}', '{typeName}')")

        If typeName = "BinaryFormatterVB.BookRecord" Or typeName = "BinaryFormatterVB.AisleLocation" Then
            Return Nothing
        Else
            Throw New ArgumentException("Unexpected type", NameOf(typeName))
        End If
    End Function
End Class

<Serializable()
Public Class BookRecord
    Public Property Title As String
    Public Property Location As AisleLocation
End Class

<Serializable()
Public Class AisleLocation
    Public Property Aisle As Char
    Public Property Shelf As Byte
End Class

Public Class ExampleClass
    Public Function DeserializeBookRecord(bytes As Byte()) As BookRecord
        Dim formatter As BinaryFormatter = New BinaryFormatter()
        formatter.Binder = New BookRecordSerializationBinder()
        Using ms As MemoryStream = New MemoryStream(bytes)
            Return CType(formatter.Deserialize(ms), BookRecord)
        End Using
    End Function
End Class

```

Related rules

[CA2300: Do not use insecure deserializer BinaryFormatter](#)

[CA2302: Ensure BinaryFormatter.Binder is set before calling BinaryFormatter.Deserialize](#)

CA2302: Ensure BinaryFormatter.Binder is set before calling BinaryFormatter.Deserialize

9/20/2022 • 8 minutes to read • [Edit Online](#)

	Value
Rule ID	CA2302
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

A `System.Runtime.Serialization.Formatters.Binary.BinaryFormatter` deserialization method was called or referenced and the `Binder` property may be null.

This rule is similar to [CA2301](#), but analysis can't determine if the `Binder` is definitely null.

By default, this rule analyzes the entire codebase, but this is [configurable](#).

WARNING

Restricting types with a `SerializationBinder` can't prevent all attacks. For more information, see the [BinaryFormatter security guide](#).

Rule description

Insecure deserializers are vulnerable when deserializing untrusted data. An attacker could modify the serialized data to include unexpected types to inject objects with malicious side effects. An attack against an insecure deserializer could, for example, execute commands on the underlying operating system, communicate over the network, or delete files.

This rule finds `System.Runtime.Serialization.Formatters.Binary.BinaryFormatter` deserialization method calls or references when the `Binder` might be null. If you want to disallow any deserialization with `BinaryFormatter` regardless of the `Binder` property, disable this rule and [CA2301](#), and enable rule [CA2300](#).

How to fix violations

- Use a secure serializer instead, and **don't allow an attacker to specify an arbitrary type to deserialize**. For more information see the [Preferred alternatives](#).
- Make the serialized data tamper-proof. After serialization, cryptographically sign the serialized data. Before deserialization, validate the cryptographic signature. Protect the cryptographic key from being disclosed and design for key rotations.
- This option makes code vulnerable to denial of service attacks and possible remote code execution attacks in the future. For more information, see the [BinaryFormatter security guide](#). Restrict serialized types. Implement a custom `System.Runtime.Serialization.SerializationBinder`. Before deserializing, set the `Binder` property to an instance of your custom `SerializationBinder` in all code paths. In the overridden `BindToType`

method, if the type is unexpected, throw an exception to stop deserialization.

When to suppress warnings

`BinaryFormatter` is insecure and can't be made secure.

Configure code to analyze

Use the following options to configure which parts of your codebase to run this rule on.

- Exclude specific symbols
- Exclude specific types and their derived types

You can configure these options for just this rule, for all rules, or for all rules in this category ([Security](#)). For more information, see [Code quality rule configuration options](#).

Exclude specific symbols

You can exclude specific symbols, such as types and methods, from analysis. For example, to specify that the rule should not run on any code within types named `MyType`, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Symbol name only (includes all symbols with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#). Each symbol name requires a symbol-kind prefix, such as `M:` for methods, `T:` for types, and `N:` for namespaces.
- `.ctor` for constructors and `.cctor` for static constructors.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType</code>	Matches all symbols named <code>MyType</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType1 MyType2</code>	Matches all symbols named either <code>MyType1</code> or <code>MyType2</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS.MyType.MyMethod(ParamType)</code>	Matches specific method <code>MyMethod</code> with the specified fully qualified signature.
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS1.MyType1.MyMethod1(ParamType) M:NS2.MyType2.MyMethod2(ParamType)</code>	Matches specific methods <code>MyMethod1</code> and <code>MyMethod2</code> with the respective fully qualified signatures.

Exclude specific types and their derived types

You can exclude specific types and their derived types from analysis. For example, to specify that the rule should not run on any methods within types named `MyType` and their derived types, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Type name only (includes all types with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#), with an optional `T:` prefix.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType</code>	Matches all types named <code>MyType</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType1 MyType2</code>	Matches all types named either <code>MyType1</code> or <code>MyType2</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS.MyType</code>	Matches specific type <code>MyType</code> with given fully qualified name and all of its derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS1.MyType1 M:NS2.MyType2</code>	Matches specific types <code>MyType1</code> and <code>MyType2</code> with the respective fully qualified names, and all of their derived types.

Pseudo-code examples

Violation 1

```

using System;
using System.IO;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;

public class BookRecordSerializationBinder : SerializationBinder
{
    public override Type BindToType(string assemblyName, string typeName)
    {
        // One way to discover expected types is through testing deserialization
        // of **valid** data and logging the types used.

        ////Console.WriteLine($"BindToType('{assemblyName}', '{typeName}')");

        if (typeName == "BookRecord" || typeName == "AisleLocation")
        {
            return null;
        }
        else
        {
            throw new ArgumentException("Unexpected type", nameof(typeName));
        }
    }
}

[Serializable]
public class BookRecord
{
    public string Title { get; set; }
    public AisleLocation Location { get; set; }
}

[Serializable]
public class AisleLocation
{
    public char Aisle { get; set; }
    public byte Shelf { get; set; }
}

public class Binders
{
    public static SerializationBinder BookRecord =
        new BookRecordSerializationBinder();
}

public class ExampleClass
{
    public BookRecord DeserializeBookRecord(byte[] bytes)
    {
        BinaryFormatter formatter = new BinaryFormatter();
        formatter.Binder = Binders.BookRecord;
        using (MemoryStream ms = new MemoryStream(bytes))
        {
            return (BookRecord)formatter.Deserialize(ms);      // CA2302 violation
        }
    }
}

```

```

Imports System
Imports System.IO
Imports System.Runtime.Serialization
Imports System.Runtime.Serialization.Formatters.Binary

Public Class BookRecordSerializationBinder
    Inherits SerializationBinder

    Public Overrides Function BindToType(assemblyName As String, typeName As String) As Type
        ' One way to discover expected types is through testing deserialization
        ' of **valid** data and logging the types used.

        'Console.WriteLine($"BindToType('{assemblyName}', '{typeName}')")

        If typeName = "BinaryFormatterVB.BookRecord" Or typeName = "BinaryFormatterVB.AisleLocation" Then
            Return Nothing
        Else
            Throw New ArgumentException("Unexpected type", NameOf(typeName))
        End If
    End Function
End Class

<Serializable()
Public Class BookRecord
    Public Property Title As String
    Public Property Location As AisleLocation
End Class

<Serializable()
Public Class AisleLocation
    Public Property Aisle As Char
    Public Property Shelf As Byte
End Class

Public Class Binders
    Public Shared Property BookRecord As SerializationBinder = New BookRecordSerializationBinder()
End Class

Public Class ExampleClass
    Public Function DeserializeBookRecord(bytes As Byte()) As BookRecord
        Dim formatter As BinaryFormatter = New BinaryFormatter()
        formatter.Binder = Binders.BookRecord
        Using ms As MemoryStream = New MemoryStream(bytes)
            Return CType(formatter.Deserialize(ms), BookRecord)      ' CA2302 violation
        End Using
    End Function
End Class

```

Solution 1

```

using System;
using System.IO;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;

public class BookRecordSerializationBinder : SerializationBinder
{
    public override Type BindToType(string assemblyName, string typeName)
    {
        // One way to discover expected types is through testing deserialization
        // of **valid** data and logging the types used.

        ////Console.WriteLine($"BindToType('{assemblyName}', '{typeName}')");

        if (typeName == "BookRecord" || typeName == "AisleLocation")
        {
            return null;
        }
        else
        {
            throw new ArgumentException("Unexpected type", nameof(typeName));
        }
    }
}

[Serializable]
public class BookRecord
{
    public string Title { get; set; }
    public AisleLocation Location { get; set; }
}

[Serializable]
public class AisleLocation
{
    public char Aisle { get; set; }
    public byte Shelf { get; set; }
}

public class Binders
{
    public static SerializationBinder BookRecord =
        new BookRecordSerializationBinder();
}

public class ExampleClass
{
    public BookRecord DeserializeBookRecord(byte[] bytes)
    {
        BinaryFormatter formatter = new BinaryFormatter();

        // Ensure that Binder is always non-null before deserializing
        formatter.Binder = Binders.BookRecord ?? throw new Exception("Expected non-null binder");

        using (MemoryStream ms = new MemoryStream(bytes))
        {
            return (BookRecord)formatter.Deserialize(ms);
        }
    }
}

```

```

Imports System
Imports System.IO
Imports System.Runtime.Serialization
Imports System.Runtime.Serialization.Formatters.Binary

Public Class BookRecordSerializationBinder
    Inherits SerializationBinder

    Public Overrides Function BindToType(assemblyName As String, typeName As String) As Type
        ' One way to discover expected types is through testing deserialization
        ' of **valid** data and logging the types used.

        'Console.WriteLine($"BindToType('{assemblyName}', '{typeName}')")

        If typeName = "BinaryFormatterVB.BookRecord" Or typeName = "BinaryFormatterVB.AisleLocation" Then
            Return Nothing
        Else
            Throw New ArgumentException("Unexpected type", NameOf(typeName))
        End If
    End Function
End Class

<Serializable()
Public Class BookRecord
    Public Property Title As String
    Public Property Location As AisleLocation
End Class

<Serializable()
Public Class AisleLocation
    Public Property Aisle As Char
    Public Property Shelf As Byte
End Class

Public Class Binders
    Public Shared Property BookRecord As SerializationBinder = New BookRecordSerializationBinder()
End Class

Public Class ExampleClass
    Public Function DeserializeBookRecord(bytes As Byte()) As BookRecord
        Dim formatter As BinaryFormatter = New BinaryFormatter()

        ' Ensure that Binder is always non-null before deserializing
        formatter.Binder = If(Binders.BookRecord, New Exception("Expected non-null"))

        Using ms As MemoryStream = New MemoryStream(bytes)
            Return CType(formatter.Deserialize(ms), BookRecord)
        End Using
    End Function
End Class

```

Violation 2

```

using System;
using System.IO;
using System.Runtime.Serialization.Formatters.Binary;

[Serializable]
public class BookRecord
{
    public string Title { get; set; }
    public AisleLocation Location { get; set; }
}

[Serializable]
public class AisleLocation
{
    public char Aisle { get; set; }
    public byte Shelf { get; set; }
}

public class ExampleClass
{
    public BinaryFormatter Formatter { get; set; }

    public BookRecord DeserializeBookRecord(byte[] bytes)
    {
        using (MemoryStream ms = new MemoryStream(bytes))
        {
            return (BookRecord) this.Formatter.Deserialize(ms);      // CA2302 violation
        }
    }
}

```

```

Imports System
Imports System.IO
Imports System.Runtime.Serialization.Formatters.Binary

<Serializable()
Public Class BookRecord
    Public Property Title As String
    Public Property Location As AisleLocation
End Class

<Serializable()
Public Class AisleLocation
    Public Property Aisle As Char
    Public Property Shelf As Byte
End Class

Public Class ExampleClass
    Public Property Formatter As BinaryFormatter

    Public Function DeserializeBookRecord(bytes As Byte()) As BookRecord
        Using ms As MemoryStream = New MemoryStream(bytes)
            Return CType(Me.Formatter.Deserialize(ms), BookRecord)      ' CA2302 violation
        End Using
    End Function
End Class

```

Solution 2

```

using System;
using System.IO;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;

public class BookRecordSerializationBinder : SerializationBinder
{
    public override Type BindToType(string assemblyName, string typeName)
    {
        // One way to discover expected types is through testing deserialization
        // of **valid** data and logging the types used.

        ////Console.WriteLine($"BindToType('{assemblyName}', '{typeName}')");

        if (typeName == "BookRecord" || typeName == "AisleLocation")
        {
            return null;
        }
        else
        {
            throw new ArgumentException("Unexpected type", nameof(typeName));
        }
    }
}

[Serializable]
public class BookRecord
{
    public string Title { get; set; }
    public AisleLocation Location { get; set; }
}

[Serializable]
public class AisleLocation
{
    public char Aisle { get; set; }
    public byte Shelf { get; set; }
}

public class ExampleClass
{
    public BookRecord DeserializeBookRecord(byte[] bytes)
    {
        BinaryFormatter formatter = new BinaryFormatter();
        formatter.Binder = new BookRecordSerializationBinder();
        using (MemoryStream ms = new MemoryStream(bytes))
        {
            return (BookRecord) formatter.Deserialize(ms);
        }
    }
}

```

```

Imports System
Imports System.IO
Imports System.Runtime.Serialization
Imports System.Runtime.Serialization.Formatters.Binary

Public Class BookRecordSerializationBinder
    Inherits SerializationBinder

    Public Overrides Function BindToType(assemblyName As String, typeName As String) As Type
        ' One way to discover expected types is through testing deserialization
        ' of **valid** data and logging the types used.

        'Console.WriteLine($"BindToType('{assemblyName}', '{typeName}')")

        If typeName = "BinaryFormatterVB.BookRecord" Or typeName = "BinaryFormatterVB.AisleLocation" Then
            Return Nothing
        Else
            Throw New ArgumentException("Unexpected type", NameOf(typeName))
        End If
    End Function
End Class

<Serializable()
Public Class BookRecord
    Public Property Title As String
    Public Property Location As AisleLocation
End Class

<Serializable()
Public Class AisleLocation
    Public Property Aisle As Char
    Public Property Shelf As Byte
End Class

Public Class ExampleClass
    Public Function DeserializeBookRecord(bytes As Byte()) As BookRecord
        Dim formatter As BinaryFormatter = New BinaryFormatter()
        formatter.Binder = New BookRecordSerializationBinder()
        Using ms As MemoryStream = New MemoryStream(bytes)
            Return CType(formatter.Deserialize(ms), BookRecord)
        End Using
    End Function
End Class

```

Related rules

- [CA2300: Do not use insecure deserializer BinaryFormatter](#)
- [CA2301: Do not call BinaryFormatter.Deserialize without first setting BinaryFormatter.Binder](#)

CA2305: Do not use insecure deserializer LosFormatter

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA2305
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

A [System.Web.UI.LosFormatter](#) deserialization method was called or referenced.

Rule description

Insecure deserializers are vulnerable when deserializing untrusted data. An attacker could modify the serialized data to include unexpected types to inject objects with malicious side effects. An attack against an insecure deserializer could, for example, execute commands on the underlying operating system, communicate over the network, or delete files.

This rule finds [System.Web.UI.LosFormatter](#) deserialization method calls or references.

[LosFormatter](#) is insecure and can't be made secure. For more information, see the [BinaryFormatter security guide](#).

How to fix violations

- Use a secure serializer instead, and **don't allow an attacker to specify an arbitrary type to deserialize**. For more information see the [Preferred alternatives](#).
- Make the serialized data tamper-proof. After serialization, cryptographically sign the serialized data. Before deserialization, validate the cryptographic signature. Protect the cryptographic key from being disclosed and design for key rotations.

When to suppress warnings

[LosFormatter](#) is insecure and can't be made secure.

Pseudo-code examples

Violation

```
using System.IO;
using System.Web.UI;

public class ExampleClass
{
    public object MyDeserialize(byte[] bytes)
    {
        LosFormatter formatter = new LosFormatter();
        return formatter.Deserialize(new MemoryStream(bytes));
    }
}
```

```
Imports System.IO
Imports System.Web.UI

Public Class ExampleClass
    Public Function MyDeserialize(bytes As Byte()) As Object
        Dim formatter As LosFormatter = New LosFormatter()
        Return formatter.Deserialize(New MemoryStream(bytes))
    End Function
End Class
```

CA2310: Do not use insecure deserializer NetDataContractSerializer

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA2310
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

A `System.Runtime.Serialization.NetDataContractSerializer` deserialization method was called or referenced.

Rule description

Insecure deserializers are vulnerable when deserializing untrusted data. An attacker could modify the serialized data to include unexpected types to inject objects with malicious side effects. An attack against an insecure deserializer could, for example, execute commands on the underlying operating system, communicate over the network, or delete files.

This rule finds `System.Runtime.Serialization.NetDataContractSerializer` deserialization method calls or references. If you want to deserialize only when the `Binder` property is set to restrict types, disable this rule and enable rules [CA2311](#) and [CA2312](#) instead. Limiting which types can be deserialized can help mitigate against known remote code execution attacks, but your deserialization will still be vulnerable to denial of service attacks.

`NetDataContractSerializer` is insecure and can't be made secure. For more information, see the [BinaryFormatter security guide](#).

How to fix violations

- Use a secure serializer instead, and **don't allow an attacker to specify an arbitrary type to deserialize**. For more information see the [Preferred alternatives](#).
- Make the serialized data tamper-proof. After serialization, cryptographically sign the serialized data. Before deserialization, validate the cryptographic signature. Protect the cryptographic key from being disclosed and design for key rotations.
- This option makes code vulnerable to denial of service attacks and possible remote code execution attacks in the future. For more information, see the [BinaryFormatter security guide](#). Restrict serialized types. Implement a custom `System.Runtime.Serialization.SerializationBinder`. Before deserializing, set the `Binder` property to an instance of your custom `SerializationBinder` in all code paths. In the overridden `BindToType` method, if the type is unexpected, throw an exception to stop deserialization.

When to suppress warnings

`NetDataContractSerializer` is insecure and can't be made secure.

Pseudo-code examples

Violation

```
using System.IO;
using System.Runtime.Serialization;

public class ExampleClass
{
    public object MyDeserialize(byte[] bytes)
    {
        NetDataContractSerializer serializer = new NetDataContractSerializer();
        return serializer.Deserialize(new MemoryStream(bytes));
    }
}
```

```
Imports System.IO
Imports System.Runtime.Serialization

Public Class ExampleClass
    Public Function MyDeserialize(bytes As Byte()) As Object
        Dim serializer As NetDataContractSerializer = New NetDataContractSerializer()
        Return serializer.Deserialize(New MemoryStream(bytes))
    End Function
End Class
```

Related rules

[CA2311: Do not deserialize without first setting NetDataContractSerializer.Binder](#)

[CA2312: Ensure NetDataContractSerializer.Binder is set before deserializing](#)

CA2311: Do not deserialize without first setting `NetDataContractSerializer.Binder`

9/20/2022 • 5 minutes to read • [Edit Online](#)

	Value
Rule ID	CA2311
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

A `System.Runtime.Serialization.NetDataContractSerializer` deserialization method was called or referenced without the `Binder` property set.

By default, this rule analyzes the entire codebase, but this is [configurable](#).

WARNING

Restricting types with a `SerializationBinder` can't prevent all attacks. For more information, see the [BinaryFormatter security guide](#).

Rule description

Insecure deserializers are vulnerable when deserializing untrusted data. An attacker could modify the serialized data to include unexpected types to inject objects with malicious side effects. An attack against an insecure deserializer could, for example, execute commands on the underlying operating system, communicate over the network, or delete files.

This rule finds `System.Runtime.Serialization.NetDataContractSerializer` deserialization method calls or references, when `NetDataContractSerializer` doesn't have its `Binder` set. If you want to disallow any deserialization with `NetDataContractSerializer` regardless of the `Binder` property, disable this rule and [CA2312](#), and enable rule [CA2310](#).

How to fix violations

- Use a secure serializer instead, and [don't allow an attacker to specify an arbitrary type to deserialize](#). For more information see the [Preferred alternatives](#).
- Make the serialized data tamper-proof. After serialization, cryptographically sign the serialized data. Before deserialization, validate the cryptographic signature. Protect the cryptographic key from being disclosed and design for key rotations.
- This option makes code vulnerable to denial of service attacks and possible remote code execution attacks in the future. For more information, see the [BinaryFormatter security guide](#). Restrict serialized types. Implement a custom `System.Runtime.Serialization.SerializationBinder`. Before deserializing, set the `Binder` property to an instance of your custom `SerializationBinder` in all code paths. In the overridden `BindToType` method, if the type is unexpected, throw an exception to stop deserialization.

When to suppress warnings

`NetDataContractSerializer` is insecure and can't be made secure.

Configure code to analyze

Use the following options to configure which parts of your codebase to run this rule on.

- [Exclude specific symbols](#)
- [Exclude specific types and their derived types](#)

You can configure these options for just this rule, for all rules, or for all rules in this category ([Security](#)). For more information, see [Code quality rule configuration options](#).

Exclude specific symbols

You can exclude specific symbols, such as types and methods, from analysis. For example, to specify that the rule should not run on any code within types named `MyType`, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Symbol name only (includes all symbols with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#). Each symbol name requires a symbol-kind prefix, such as `M:` for methods, `T:` for types, and `N:` for namespaces.
- `.ctor` for constructors and `.cctor` for static constructors.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType</code>	Matches all symbols named <code>MyType</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType1 MyType2</code>	Matches all symbols named either <code>MyType1</code> or <code>MyType2</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS.MyType.MyMethod(ParamType)</code>	Matches specific method <code>MyMethod</code> with the specified fully qualified signature.
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS1.MyType1.MyMethod1(ParamType) M:NS2.MyType2.MyMethod2(ParamType)</code>	Matches specific methods <code>MyMethod1</code> and <code>MyMethod2</code> with the respective fully qualified signatures.

Exclude specific types and their derived types

You can exclude specific types and their derived types from analysis. For example, to specify that the rule should not run on any methods within types named `MyType` and their derived types, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Type name only (includes all types with the name, regardless of the containing type or namespace).

- Fully qualified names in the symbol's [documentation ID format](#), with an optional `T:` prefix.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType</code>	Matches all types named <code>MyType</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType1 MyType2</code>	Matches all types named either <code>MyType1</code> or <code>MyType2</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS.MyType</code>	Matches specific type <code>MyType</code> with given fully qualified name and all of its derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS1.MyType1 M:NS2.MyType2</code>	Matches specific types <code>MyType1</code> and <code>MyType2</code> with the respective fully qualified names, and all of their derived types.

Pseudo-code examples

Violation

```

using System;
using System.IO;
using System.Runtime.Serialization;

[DataContract]
public class BookRecord
{
    [DataMember]
    public string Title { get; set; }

    [DataMember]
    public AisleLocation Location { get; set; }
}

[DataContract]
public class AisleLocation
{
    [DataMember]
    public char Aisle { get; set; }

    [DataMember]
    public byte Shelf { get; set; }
}

public class ExampleClass
{
    public BookRecord DeserializeBookRecord(byte[] bytes)
    {
        NetDataContractSerializer serializer = new NetDataContractSerializer();
        using (MemoryStream ms = new MemoryStream(bytes))
        {
            return (BookRecord) serializer.Deserialize(ms);      // CA2311 violation
        }
    }
}

```

```

Imports System
Imports System.IO
Imports System.Runtime.Serialization

<DataContract()
Public Class BookRecord
    <DataMember()
    Public Property Title As String

    <DataMember()
    Public Property Location As AisleLocation
End Class

<DataContract()
Public Class AisleLocation
    <DataMember()
    Public Property Aisle As Char

    <DataMember()
    Public Property Shelf As Byte
End Class

Public Class ExampleClass
    Public Function DeserializeBookRecord(bytes As Byte()) As BookRecord
        Dim serializer As NetDataContractSerializer = New NetDataContractSerializer()
        Using ms As MemoryStream = New MemoryStream(bytes)
            Return CType(serializer.Deserialize(ms), BookRecord)      ' CA2311 violation
        End Using
    End Function
End Class

```

Solution

```

using System;
using System.IO;
using System.Runtime.Serialization;

public class BookRecordSerializationBinder : SerializationBinder
{
    public override Type BindToType(string assemblyName, string typeName)
    {
        // One way to discover expected types is through testing deserialization
        // of **valid** data and logging the types used.

        ////Console.WriteLine($"BindToType('{assemblyName}', '{typeName}')");

        if (typeName == "BookRecord" || typeName == "AisleLocation")
        {
            return null;
        }
        else
        {
            throw new ArgumentException("Unexpected type", nameof(typeName));
        }
    }
}

[DataContract]
public class BookRecord
{
    [DataMember]
    public string Title { get; set; }

    [DataMember]
    public AisleLocation Location { get; set; }
}

[DataContract]
public class AisleLocation
{
    [DataMember]
    public char Aisle { get; set; }

    [DataMember]
    public byte Shelf { get; set; }
}

public class ExampleClass
{
    public BookRecord DeserializeBookRecord(byte[] bytes)
    {
        NetDataContractSerializer serializer = new NetDataContractSerializer();
        serializer.Binder = new BookRecordSerializationBinder();
        using (MemoryStream ms = new MemoryStream(bytes))
        {
            return (BookRecord) serializer.Deserialize(ms);
        }
    }
}

```

```

Imports System
Imports System.IO
Imports System.Runtime.Serialization

Public Class BookRecordSerializationBinder
    Inherits SerializationBinder

    Public Overrides Function BindToType(assemblyName As String, typeName As String) As Type
        ' One way to discover expected types is through testing deserialization
        ' of **valid** data and logging the types used.

        'Console.WriteLine($"BindToType('{assemblyName}', '{typeName}')")

        If typeName = "BinaryFormatterVB.BookRecord" Or typeName = "BinaryFormatterVB.AisleLocation" Then
            Return Nothing
        Else
            Throw New ArgumentException("Unexpected type", NameOf(typeName))
        End If
    End Function
End Class

<DataContract()
Public Class BookRecord
    <DataMember()
    Public Property Title As String

    <DataMember()
    Public Property Location As AisleLocation
End Class

<DataContract()
Public Class AisleLocation
    <DataMember()
    Public Property Aisle As Char

    <DataMember()
    Public Property Shelf As Byte
End Class

Public Class ExampleClass
    Public Function DeserializeBookRecord(bytes As Byte()) As BookRecord
        Dim serializer As NetDataContractSerializer = New NetDataContractSerializer()
        serializer.Binder = New BookRecordSerializationBinder()
        Using ms As MemoryStream = New MemoryStream(bytes)
            Return CType(serializer.Deserialize(ms), BookRecord)
        End Using
    End Function
End Class

```

Related rules

[CA2310: Do not use insecure deserializer NetDataContractSerializer](#)

[CA2312: Ensure NetDataContractSerializer.Binder is set before deserializing](#)

CA2312: Ensure `NetDataContractSerializer.Binder` is set before deserializing

9/20/2022 • 9 minutes to read • [Edit Online](#)

	Value
Rule ID	CA2312
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

A `System.Runtime.Serialization.NetDataContractSerializer` deserialization method was called or referenced and the `Binder` property may be null.

This rule is similar to [CA2311](#), but analysis can't determine if the `Binder` is definitely null.

By default, this rule analyzes the entire codebase, but this is [configurable](#).

WARNING

Restricting types with a `SerializationBinder` can't prevent all attacks. For more information, see the [BinaryFormatter security guide](#).

Rule description

Insecure deserializers are vulnerable when deserializing untrusted data. An attacker could modify the serialized data to include unexpected types to inject objects with malicious side effects. An attack against an insecure deserializer could, for example, execute commands on the underlying operating system, communicate over the network, or delete files.

This rule finds `System.Runtime.Serialization.NetDataContractSerializer` deserialization method calls or references when the `Binder` might be null. If you want to disallow any deserialization with `NetDataContractSerializer` regardless of the `Binder` property, disable this rule and [CA2311](#), and enable rule [CA2310](#).

`NetDataContractSerializer` is insecure and can't be made secure. For more information, see the [BinaryFormatter security guide](#).

How to fix violations

- Use a secure serializer instead, and **don't allow an attacker to specify an arbitrary type to deserialize**. For more information see the [Preferred alternatives](#).
- Make the serialized data tamper-proof. After serialization, cryptographically sign the serialized data. Before deserialization, validate the cryptographic signature. Protect the cryptographic key from being disclosed and design for key rotations.
- This option makes code vulnerable to denial of service attacks and possible remote code execution attacks in

the future. For more information, see the [BinaryFormatter security guide](#). Restrict deserialized types. Implement a custom [System.Runtime.Serialization.SerializationBinder](#). Before deserializing, set the [BINDER](#) property to an instance of your custom [SerializationBinder](#) in all code paths. In the overridden [BindToType](#) method, if the type is unexpected, throw an exception to stop deserialization.

When to suppress warnings

`NetDataContractSerializer` is insecure and can't be made secure.

Configure code to analyze

Use the following options to configure which parts of your codebase to run this rule on.

- [Exclude specific symbols](#)
- [Exclude specific types and their derived types](#)

You can configure these options for just this rule, for all rules, or for all rules in this category ([Security](#)). For more information, see [Code quality rule configuration options](#).

Exclude specific symbols

You can exclude specific symbols, such as types and methods, from analysis. For example, to specify that the rule should not run on any code within types named `MyType`, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Symbol name only (includes all symbols with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#). Each symbol name requires a symbol-kind prefix, such as `M:` for methods, `T:` for types, and `N:` for namespaces.
- `.ctor` for constructors and `.cctor` for static constructors.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType</code>	Matches all symbols named <code>MyType</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType1 MyType2</code>	Matches all symbols named either <code>MyType1</code> or <code>MyType2</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS.MyType.MyMethod(ParamType)</code>	Matches specific method <code>MyMethod</code> with the specified fully qualified signature.
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS1.MyType1.MyMethod1(ParamType) M:NS2.MyType2.MyMethod2(ParamType)</code>	Matches specific methods <code>MyMethod1</code> and <code>MyMethod2</code> with the respective fully qualified signatures.

Exclude specific types and their derived types

You can exclude specific types and their derived types from analysis. For example, to specify that the rule should not run on any methods within types named `MyType` and their derived types, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Type name only (includes all types with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#), with an optional `T:` prefix.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType</code>	Matches all types named <code>MyType</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType1 MyType2</code>	Matches all types named either <code>MyType1</code> or <code>MyType2</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS.MyType</code>	Matches specific type <code>MyType</code> with given fully qualified name and all of its derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS1.MyType1 M:NS2.MyType2</code>	Matches specific types <code>MyType1</code> and <code>MyType2</code> with the respective fully qualified names, and all of their derived types.

Pseudo-code examples

Violation 1

```

using System;
using System.IO;
using System.Runtime.Serialization;

public class BookRecordSerializationBinder : SerializationBinder
{
    public override Type BindToType(string assemblyName, string typeName)
    {
        // One way to discover expected types is through testing deserialization
        // of **valid** data and logging the types used.

        ////Console.WriteLine($"BindToType('{assemblyName}', '{typeName}')");

        if (typeName == "BookRecord" || typeName == "AisleLocation")
        {
            return null;
        }
        else
        {
            throw new ArgumentException("Unexpected type", nameof(typeName));
        }
    }
}

[DataContract]
public class BookRecord
{
    [DataMember]
    public string Title { get; set; }

    [DataMember]
    public AisleLocation Location { get; set; }
}

[DataContract]
public class AisleLocation
{
    [DataMember]
    public char Aisle { get; set; }

    [DataMember]
    public byte Shelf { get; set; }
}

public class Binders
{
    public static SerializationBinder BookRecord = new BookRecordSerializationBinder();
}

public class ExampleClass
{
    public BookRecord DeserializeBookRecord(byte[] bytes)
    {
        NetDataContractSerializer serializer = new NetDataContractSerializer();
        serializer.Binder = Binders.BookRecord;
        using (MemoryStream ms = new MemoryStream(bytes))
        {
            return (BookRecord) serializer.Deserialize(ms);
        }
    }
}

```

```

Imports System
Imports System.IO
Imports System.Runtime.Serialization

Public Class BookRecordSerializationBinder
    Inherits SerializationBinder

    Public Overrides Function BindToType(assemblyName As String, typeName As String) As Type
        ' One way to discover expected types is through testing deserialization
        ' of **valid** data and logging the types used.

        'Console.WriteLine($"BindToType('{assemblyName}', '{typeName}')")

        If typeName = "BinaryFormatterVB.BookRecord" Or typeName = "BinaryFormatterVB.AisleLocation" Then
            Return Nothing
        Else
            Throw New ArgumentException("Unexpected type", NameOf(typeName))
        End If
    End Function
End Class

<DataContract()
Public Class BookRecord
    <DataMember()
    Public Property Title As String

    <DataMember()
    Public Property Location As AisleLocation
End Class

<DataContract()
Public Class AisleLocation
    <DataMember()
    Public Property Aisle As Char

    <DataMember()
    Public Property Shelf As Byte
End Class

Public Class Binders
    Public Shared Property BookRecord As SerializationBinder = New BookRecordSerializationBinder()
End Class

Public Class ExampleClass
    Public Function DeserializeBookRecord(bytes As Byte()) As BookRecord
        Dim serializer As NetDataContractSerializer = New NetDataContractSerializer()
        serializer.Binder = Binders.BookRecord
        Using ms As MemoryStream = New MemoryStream(bytes)
            Return CType(serializer.Deserialize(ms), BookRecord)      ' CA2312 violation
        End Using
    End Function
End Class

```

Solution 1

```

using System;
using System.IO;
using System.Runtime.Serialization;

public class BookRecordSerializationBinder : SerializationBinder
{
    public override Type BindToType(string assemblyName, string typeName)
    {
        // One way to discover expected types is through testing deserialization
        // of **valid** data and logging the types used.

        ////Console.WriteLine($"BindToType('{assemblyName}', '{typeName}')");

        if (typeName == "BookRecord" || typeName == "AisleLocation")
        {
            return null;
        }
        else
        {
            throw new ArgumentException("Unexpected type", nameof(typeName));
        }
    }
}

[DataContract]
public class BookRecord
{
    [DataMember]
    public string Title { get; set; }

    [DataMember]
    public AisleLocation Location { get; set; }
}

[DataContract]
public class AisleLocation
{
    [DataMember]
    public char Aisle { get; set; }

    [DataMember]
    public byte Shelf { get; set; }
}

public class Binders
{
    public static SerializationBinder BookRecord = new BookRecordSerializationBinder();
}

public class ExampleClass
{
    public BookRecord DeserializeBookRecord(byte[] bytes)
    {
        NetDataContractSerializer serializer = new NetDataContractSerializer();

        // Ensure that Binder is always non-null before deserializing
        serializer.Binder = Binders.BookRecord ?? throw new Exception("Expected non-null");

        using (MemoryStream ms = new MemoryStream(bytes))
        {
            return (BookRecord) serializer.Deserialize(ms);
        }
    }
}

```

```

Imports System
Imports System.IO
Imports System.Runtime.Serialization

Public Class BookRecordSerializationBinder
    Inherits SerializationBinder

    Public Overrides Function BindToType(assemblyName As String, typeName As String) As Type
        ' One way to discover expected types is through testing deserialization
        ' of **valid** data and logging the types used.

        'Console.WriteLine($"BindToType('{assemblyName}', '{typeName}')")

        If typeName = "BinaryFormatterVB.BookRecord" Or typeName = "BinaryFormatterVB.AisleLocation" Then
            Return Nothing
        Else
            Throw New ArgumentException("Unexpected type", NameOf(typeName))
        End If
    End Function
End Class

<DataContract()
Public Class BookRecord
    <DataMember()
    Public Property Title As String

    <DataMember()
    Public Property Location As AisleLocation
End Class

<DataContract()
Public Class AisleLocation
    <DataMember()
    Public Property Aisle As Char

    <DataMember()
    Public Property Shelf As Byte
End Class

Public Class Binders
    Public Shared Property BookRecord As SerializationBinder = New BookRecordSerializationBinder()
End Class

Public Class ExampleClass
    Public Function DeserializeBookRecord(bytes As Byte()) As BookRecord
        Dim serializer As NetDataContractSerializer = New NetDataContractSerializer()

        ' Ensure that Binder is always non-null before deserializing
        serializer.Binder = If(Binders.BookRecord, New Exception("Expected non-null"))

        Using ms As MemoryStream = New MemoryStream(bytes)
            Return CType(serializer.Deserialize(ms), BookRecord)
        End Using
    End Function
End Class

```

Violation 2

```
using System;
using System.IO;
using System.Runtime.Serialization;

[DataContract]
public class BookRecord
{
    [DataMember]
    public string Title { get; set; }

    [DataMember]
    public string Author { get; set; }

    [DataMember]
    public int PageCount { get; set; }

    [DataMember]
    public AisleLocation Location { get; set; }
}

[DataContract]
public class AisleLocation
{
    [DataMember]
    public char Aisle { get; set; }

    [DataMember]
    public byte Shelf { get; set; }
}

public class ExampleClass
{
    public NetDataContractSerializer Serializer { get; set; }

    public BookRecord DeserializeBookRecord(byte[] bytes)
    {
        using (MemoryStream ms = new MemoryStream(bytes))
        {
            return (BookRecord) this.Serializer.Deserialize(ms);      // CA2312 violation
        }
    }
}
```

```

Imports System
Imports System.IO
Imports System.Runtime.Serialization

<DataContract()
Public Class BookRecord
    <DataMember()
    Public Property Title As String

    <DataMember()
    Public Property Author As String

    <DataMember()
    Public Property Location As AisleLocation
End Class

<DataContract()
Public Class AisleLocation
    <DataMember()
    Public Property Aisle As Char

    <DataMember()
    Public Property Shelf As Byte
End Class

Public Class ExampleClass
    Public Property Serializer As NetDataContractSerializer

    Public Function DeserializeBookRecord(bytes As Byte()) As BookRecord
        Using ms As MemoryStream = New MemoryStream(bytes)
            Return CType(Me.Serializer.Deserialize(ms), BookRecord)      ' CA2312 violation
        End Using
    End Function
End Class

```

Solution 2

```

using System;
using System.IO;
using System.Runtime.Serialization;

public class BookRecordSerializationBinder : SerializationBinder
{
    public override Type BindToType(string assemblyName, string typeName)
    {
        // One way to discover expected types is through testing deserialization
        // of **valid** data and logging the types used.

        ////Console.WriteLine($"BindToType('{assemblyName}', '{typeName}')");

        if (typeName == "BookRecord" || typeName == "AisleLocation")
        {
            return null;
        }
        else
        {
            throw new ArgumentException("Unexpected type", nameof(typeName));
        }
    }
}

[DataContract]
public class BookRecord
{
    [DataMember]
    public string Title { get; set; }

    [DataMember]
    public string Author { get; set; }

    [DataMember]
    public int PageCount { get; set; }

    [DataMember]
    public AisleLocation Location { get; set; }
}

[DataContract]
public class AisleLocation
{
    [DataMember]
    public char Aisle { get; set; }

    [DataMember]
    public byte Shelf { get; set; }
}

public class ExampleClass
{
    public BookRecord DeserializeBookRecord(byte[] bytes)
    {
        NetDataContractSerializer serializer = new NetDataContractSerializer();
        serializer.Binder = new BookRecordSerializationBinder();
        using (MemoryStream ms = new MemoryStream(bytes))
        {
            return (BookRecord) serializer.Deserialize(ms);
        }
    }
}

```

```

Imports System
Imports System.IO
Imports System.Runtime.Serialization

Public Class BookRecordSerializationBinder
    Inherits SerializationBinder

    Public Overrides Function BindToType(assemblyName As String, typeName As String) As Type
        ' One way to discover expected types is through testing deserialization
        ' of **valid** data and logging the types used.

        'Console.WriteLine($"BindToType('{assemblyName}', '{typeName}')")

        If typeName = "BinaryFormatterVB.BookRecord" Or typeName = "BinaryFormatterVB.AisleLocation" Then
            Return Nothing
        Else
            Throw New ArgumentException("Unexpected type", NameOf(typeName))
        End If
    End Function
End Class

<DataContract()
Public Class BookRecord
    <DataMember()
    Public Property Title As String

    <DataMember()
    Public Property Author As String

    <DataMember()
    Public Property Location As AisleLocation
End Class

<DataContract()
Public Class AisleLocation
    <DataMember()
    Public Property Aisle As Char

    <DataMember()
    Public Property Shelf As Byte
End Class

Public Class ExampleClass
    Public Function DeserializeBookRecord(bytes As Byte()) As BookRecord
        Dim serializer As NetDataContractSerializer = New NetDataContractSerializer()
        serializer.Binder = New BookRecordSerializationBinder()
        Using ms As MemoryStream = New MemoryStream(bytes)
            Return CType(serializer.Deserialize(ms), BookRecord)
        End Using
    End Function
End Class

```

Related rules

- [CA2310: Do not use insecure deserializer NetDataContractSerializer](#)
- [CA2311: Do not deserialize without first setting NetDataContractSerializer.Binder](#)

CA2315: Do not use insecure deserializer ObjectStateFormatter

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA2315
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

A [System.Web.UI.ObjectStateFormatter](#) deserialization method was called or referenced.

Rule description

Insecure deserializers are vulnerable when deserializing untrusted data. An attacker could modify the serialized data to include unexpected types to inject objects with malicious side effects. An attack against an insecure deserializer could, for example, execute commands on the underlying operating system, communicate over the network, or delete files.

This rule finds [System.Web.UI.ObjectStateFormatter](#) deserialization method calls or references.

How to fix violations

- If possible, use a secure serializer instead, and **don't allow an attacker to specify an arbitrary type to deserialize**. Some safer serializers include:
 - [System.Runtime.Serialization.DataContractSerializer](#)
 - [System.Runtime.Serialization.JsonDataContractJsonSerializer](#)
 - [System.Web.Script.Serialization.JavaScriptSerializer](#) - Never use [System.Web.Script.Serialization.SimpleTypeResolver](#). If you must use a type resolver, restrict serialized types to an expected list.
 - [System.Xml.Serialization.XmlSerializer](#)
 - Newtonsoft.Json.NET - Use `TypeNameHandling.None`. If you must use another value for `TypeNameHandling`, restrict serialized types to an expected list with a custom `ISerializationBinder`.
 - Protocol Buffers
- Make the serialized data tamper-proof. After serialization, cryptographically sign the serialized data. Before deserialization, validate the cryptographic signature. Protect the cryptographic key from being disclosed and design for key rotations.

When to suppress warnings

It's safe to suppress a warning from this rule if:

- You know the input is trusted. Consider that your application's trust boundary and data flows may change

over time.

- You've taken one of the precautions in [How to fix violations](#).

Pseudo-code examples

Violation

```
using System.IO;
using System.Web.UI;

public class ExampleClass
{
    public object MyDeserialize(byte[] bytes)
    {
        ObjectStateFormatter formatter = new ObjectStateFormatter();
        return formatter.Deserialize(new MemoryStream(bytes));
    }
}
```

```
Imports System.IO
Imports System.Web.UI

Public Class ExampleClass
    Public Function MyDeserialize(bytes As Byte()) As Object
        Dim formatter As ObjectStateFormatter = New ObjectStateFormatter()
        Return formatter.Deserialize(New MemoryStream(bytes))
    End Function
End Class
```

CA2321: Do not deserialize with JavaScriptSerializer using a SimpleTypeResolver

9/20/2022 • 6 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA2321
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

A `System.Web.Script.Serialization.JavaScriptSerializer` deserialization method was called or referenced after initializing with a `System.Web.Script.Serialization.SimpleTypeResolver`.

By default, this rule analyzes the entire codebase, but this is [configurable](#).

Rule description

Insecure deserializers are vulnerable when deserializing untrusted data. An attacker could modify the serialized data to include unexpected types to inject objects with malicious side effects. An attack against an insecure deserializer could, for example, execute commands on the underlying operating system, communicate over the network, or delete files.

This rule finds `System.Web.Script.Serialization.JavaScriptSerializer` deserialization method calls or references, after initializing the `JavaScriptSerializer` with a `System.Web.Script.Serialization.SimpleTypeResolver`.

How to fix violations

- Don't initialize `JavaScriptTypeResolver` with a `System.Web.Script.Serialization.SimpleTypeResolver`.
- If your code needs to read data serialized using a `SimpleTypeResolver`, restrict deserialized types to an expected list by implementing a custom `JavaScriptTypeResolver`.
- Make the serialized data tamper-proof. After serialization, cryptographically sign the serialized data. Before deserialization, validate the cryptographic signature. Protect the cryptographic key from being disclosed and design for key rotations.

When to suppress warnings

It's safe to suppress a warning from this rule if:

- You know the input is trusted. Consider that your application's trust boundary and data flows may change over time.
- You've taken one of the precautions in [How to fix violations](#).

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then

re-enable the rule.

```
#pragma warning disable CA2321
// The code that's violating the rule is on this line.
#pragma warning restore CA2321
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA2321.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

Use the following options to configure which parts of your codebase to run this rule on.

- [Exclude specific symbols](#)
- [Exclude specific types and their derived types](#)

You can configure these options for just this rule, for all rules, or for all rules in this category ([Security](#)). For more information, see [Code quality rule configuration options](#).

Exclude specific symbols

You can exclude specific symbols, such as types and methods, from analysis. For example, to specify that the rule should not run on any code within types named `MyType`, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Symbol name only (includes all symbols with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#). Each symbol name requires a symbol-kind prefix, such as `M:` for methods, `T:` for types, and `N:` for namespaces.
- `.ctor` for constructors and `.cctor` for static constructors.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType</code>	Matches all symbols named <code>MyType</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType1 MyType2</code>	Matches all symbols named either <code>MyType1</code> or <code>MyType2</code> .

OPTION VALUE	SUMMARY
dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS.MyType.MyMethod(ParamType)	Matches specific method <code>MyMethod</code> with the specified fully qualified signature.
dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS1.MyType1.MyMethod1(ParamType) M:NS2.MyType2.MyMethod2(ParamType)	Matches specific methods <code>MyMethod1</code> and <code>MyMethod2</code> with the respective fully qualified signatures.

Exclude specific types and their derived types

You can exclude specific types and their derived types from analysis. For example, to specify that the rule should not run on any methods within types named `MyType` and their derived types, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Type name only (includes all types with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#), with an optional `T:` prefix.

Examples:

OPTION VALUE	SUMMARY
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType	Matches all types named <code>MyType</code> and all of their derived types.
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType1 MyType2	Matches all types named either <code>MyType1</code> or <code>MyType2</code> and all of their derived types.
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS.MyType	Matches specific type <code>MyType</code> with given fully qualified name and all of its derived types.
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS1.MyType1 M:NS2.MyType2	Matches specific types <code>MyType1</code> and <code>MyType2</code> with the respective fully qualified names, and all of their derived types.

Pseudo-code examples

Violation 1

```
using System.Web.Script.Serialization;

public class ExampleClass
{
    public T Deserialize<T>(string str)
    {
        JavaScriptSerializer s = new JavaScriptSerializer(new SimpleTypeResolver());
        return s.Deserialize<T>(str);
    }
}
```

```

Imports System.Web.Script.Serialization

Public Class ExampleClass
    Public Function Deserialize(Of T)(str As String) As T
        Dim s As JavaScriptSerializer = New JavaScriptSerializer(New SimpleTypeResolver())
        Return s.Deserialize(Of T)(str)
    End Function
End Class

```

Solution 1

```

using System.Web.Script.Serialization;

public class ExampleClass
{
    public T Deserialize<T>(string str)
    {
        JavaScriptSerializer s = new JavaScriptSerializer();
        return s.Deserialize<T>(str);
    }
}

```

```

Imports System.Web.Script.Serialization

Public Class ExampleClass
    Public Function Deserialize(Of T)(str As String) As T
        Dim s As JavaScriptSerializer = New JavaScriptSerializer()
        Return s.Deserialize(Of T)(str)
    End Function
End Class

```

Violation 2

```

using System.Web.Script.Serialization;

public class BookRecord
{
    public string Title { get; set; }
    public string Author { get; set; }
    public int PageCount { get; set; }
    public AisleLocation Location { get; set; }
}

public class AisleLocation
{
    public char Aisle { get; set; }
    public byte Shelf { get; set; }
}

public class ExampleClass
{
    public BookRecord DeserializeBookRecord(string s)
    {
        JavaScriptSerializer serializer = new JavaScriptSerializer(new SimpleTypeResolver());
        return serializer.Deserialize<BookRecord>(s);
    }
}

```

```
Imports System.Web.Script.Serialization

Public Class BookRecord
    Public Property Title As String
    Public Property Author As String
    Public Property Location As AisleLocation
End Class

Public Class AisleLocation
    Public Property Aisle As Char
    Public Property Shelf As Byte
End Class

Public Class ExampleClass
    Public Function DeserializeBookRecord(str As String) As BookRecord
        Dim serializer As JavaScriptSerializer = New JavaScriptSerializer(New SimpleTypeResolver())
        Return serializer.Deserialize(Of BookRecord)(str)
    End Function
End Class
```

Solution 2

```

using System;
using System.Web.Script.Serialization;

public class BookRecordTypeResolver : JavaScriptTypeResolver
{
    // For compatibility with data serialized with a JavaScriptSerializer initialized with
    SimpleTypeResolver.
    private static readonly SimpleTypeResolver Simple = new SimpleTypeResolver();

    public override Type ResolveType(string id)
    {
        // One way to discover expected types is through testing deserialization
        // of **valid** data and logging the types used.

        ////Console.WriteLine($"ResolveType('{id}')");

        if (id == typeof(BookRecord).AssemblyQualifiedName || id ==
        typeof(AisleLocation).AssemblyQualifiedName)
        {
            return Simple.ResolveType(id);
        }
        else
        {
            throw new ArgumentException("Unexpected type ID", nameof(id));
        }
    }

    public override string ResolveTypeId(Type type)
    {
        return Simple.ResolveTypeId(type);
    }
}

public class BookRecord
{
    public string Title { get; set; }
    public string Author { get; set; }
    public int PageCount { get; set; }
    public AisleLocation Location { get; set; }
}

public class AisleLocation
{
    public char Aisle { get; set; }
    public byte Shelf { get; set; }
}

public class ExampleClass
{
    public BookRecord DeserializeBookRecord(string s)
    {
        JavaScriptSerializer serializer = new JavaScriptSerializer(new BookRecordTypeResolver());
        return serializer.Deserialize<BookRecord>(s);
    }
}

```

```

Imports System
Imports System.Web.Script.Serialization

Public Class BookRecordTypeResolver
    Inherits JavaScriptTypeResolver

    ' For compatibility with data serialized with a JavaScriptSerializer initialized with
    ' SimpleTypeResolver.
    Private Dim Simple As SimpleTypeResolver = New SimpleTypeResolver()

    Public Overrides Function ResolveType(id As String) As Type
        ' One way to discover expected types is through testing deserialization
        ' of **valid** data and logging the types used.

        ''Console.WriteLine($"ResolveType('{id}')")

        If id = GetType(BookRecord).AssemblyQualifiedName Or id =
        GetType(AisleLocation).AssemblyQualifiedName Then
            Return Simple.ResolveType(id)
        Else
            Throw New ArgumentException("Unexpected type", NameOf(id))
        End If
    End Function

    Public Overrides Function ResolveTypeId(type As Type) As String
        Return Simple.ResolveTypeId(type)
    End Function
End Class

Public Class BookRecord
    Public Property Title As String
    Public Property Author As String
    Public Property Location As AisleLocation
End Class

Public Class AisleLocation
    Public Property Aisle As Char
    Public Property Shelf As Byte
End Class

Public Class ExampleClass
    Public Function DeserializeBookRecord(str As String) As BookRecord
        Dim serializer As JavaScriptSerializer = New JavaScriptSerializer(New BookRecordTypeResolver())
        Return serializer.Deserialize(Of BookRecord)(str)
    End Function
End Class

```

Related rules

[CA2322: Ensure JavaScriptSerializer is not initialized with SimpleTypeResolver before deserializing](#)

CA2322: Ensure JavaScriptSerializer is not initialized with SimpleTypeResolver before deserializing

9/20/2022 • 6 minutes to read • [Edit Online](#)

	Value
Rule ID	CA2322
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

A `System.Web.Script.Serialization.JavaScriptSerializer` deserialization method was called or referenced and the `JavaScriptSerializer` may have been initialized with a `System.Web.Script.Serialization.SimpleTypeResolver`.

By default, this rule analyzes the entire codebase, but this is [configurable](#).

Rule description

Insecure deserializers are vulnerable when deserializing untrusted data. An attacker could modify the serialized data to include unexpected types to inject objects with malicious side effects. An attack against an insecure deserializer could, for example, execute commands on the underlying operating system, communicate over the network, or delete files.

This rule finds `System.Web.Script.Serialization.JavaScriptSerializer` deserialization method calls or references, when the `JavaScriptSerializer` may have been initialized with a `System.Web.Script.Serialization.SimpleTypeResolver`.

How to fix violations

- Ensure `JavaScriptTypeResolver` objects aren't initialized with a `System.Web.Script.Serialization.SimpleTypeResolver`.
- If your code needs to read data serialized using a `SimpleTypeResolver`, restrict deserialized types to an expected list by implementing a custom `JavaScriptTypeResolver`.
- Make the serialized data tamper-proof. After serialization, cryptographically sign the serialized data. Before deserialization, validate the cryptographic signature. Protect the cryptographic key from being disclosed and design for key rotations.

When to suppress warnings

It's safe to suppress a warning from this rule if:

- You know the input is trusted. Consider that your application's trust boundary and data flows may change over time.
- You've taken one of the precautions in [How to fix violations](#).

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA2322
// The code that's violating the rule is on this line.
#pragma warning restore CA2322
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA2322.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

Use the following options to configure which parts of your codebase to run this rule on.

- [Exclude specific symbols](#)
- [Exclude specific types and their derived types](#)

You can configure these options for just this rule, for all rules, or for all rules in this category ([Security](#)). For more information, see [Code quality rule configuration options](#).

Exclude specific symbols

You can exclude specific symbols, such as types and methods, from analysis. For example, to specify that the rule should not run on any code within types named `MyType`, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Symbol name only (includes all symbols with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#). Each symbol name requires a symbol-kind prefix, such as `M:` for methods, `T:` for types, and `N:` for namespaces.
- `.ctor` for constructors and `.cctor` for static constructors.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType</code>	Matches all symbols named <code>MyType</code> .

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType1 MyType2</code>	Matches all symbols named either <code>MyType1</code> or <code>MyType2</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS.MyType.MyMethod(ParamType)</code>	Matches specific method <code>MyMethod</code> with the specified fully qualified signature.
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS1.MyType1.MyMethod1(ParamType) M:NS2.MyType2.MyMethod2(ParamType)</code>	Matches specific methods <code>MyMethod1</code> and <code>MyMethod2</code> with the respective fully qualified signatures.

Exclude specific types and their derived types

You can exclude specific types and their derived types from analysis. For example, to specify that the rule should not run on any methods within types named `MyType` and their derived types, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Type name only (includes all types with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#), with an optional `T:` prefix.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType</code>	Matches all types named <code>MyType</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType1 MyType2</code>	Matches all types named either <code>MyType1</code> or <code>MyType2</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS.MyType</code>	Matches specific type <code>MyType</code> with given fully qualified name and all of its derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS1.MyType1 M:NS2.MyType2</code>	Matches specific types <code>MyType1</code> and <code>MyType2</code> with the respective fully qualified names, and all of their derived types.

Pseudo-code examples

Violation 1

```

using System.Web.Script.Serialization;

public class ExampleClass
{
    public JavaScriptSerializer Serializer { get; set; }

    public T Deserialize<T>(string str)
    {
        return this.Serializer.Deserialize<T>(str);
    }
}

```

```

Imports System.Web.Script.Serialization

Public Class ExampleClass
    Public Property Serializer As JavaScriptSerializer

    Public Function Deserialize(Of T)(str As String) As T
        Return Me.Serializer.Deserialize(Of T)(str)
    End Function
End Class

```

Solution 1

```

using System.Web.Script.Serialization;

public class ExampleClass
{
    public T Deserialize<T>(string str)
    {
        JavaScriptSerializer s = new JavaScriptSerializer();
        return s.Deserialize<T>(str);
    }
}

```

```

Imports System.Web.Script.Serialization

Public Class ExampleClass
    Public Function Deserialize(Of T)(str As String) As T
        Dim s As JavaScriptSerializer = New JavaScriptSerializer()
        Return s.Deserialize(Of T)(str)
    End Function
End Class

```

Violation 2

```

using System.Web.Script.Serialization;

public class BookRecord
{
    public string Title { get; set; }
    public string Author { get; set; }
    public int PageCount { get; set; }
    public AisleLocation Location { get; set; }
}

public class AisleLocation
{
    public char Aisle { get; set; }
    public byte Shelf { get; set; }
}

public class ExampleClass
{
    public JavaScriptSerializer Serializer { get; set; }

    public BookRecord DeserializeBookRecord(string s)
    {
        return this.Serializer.Deserialize<BookRecord>(s);
    }
}

```

```

Imports System.Web.Script.Serialization

Public Class BookRecord
    Public Property Title As String
    Public Property Author As String
    Public Property Location As AisleLocation
End Class

Public Class AisleLocation
    Public Property Aisle As Char
    Public Property Shelf As Byte
End Class

Public Class ExampleClass
    Public Property Serializer As JavaScriptSerializer

    Public Function DeserializeBookRecord(str As String) As BookRecord
        Return Me.Serializer.Deserialize(Of BookRecord)(str)
    End Function
End Class

```

Solution 2

```

using System;
using System.Web.Script.Serialization;

public class BookRecordTypeResolver : JavaScriptTypeResolver
{
    // For compatibility with data serialized with a JavaScriptSerializer initialized with
    SimpleTypeResolver.
    private static readonly SimpleTypeResolver Simple = new SimpleTypeResolver();

    public override Type ResolveType(string id)
    {
        // One way to discover expected types is through testing deserialization
        // of **valid** data and logging the types used.

        ////Console.WriteLine($"ResolveType('{id}')");

        if (id == typeof(BookRecord).AssemblyQualifiedName || id ==
        typeof(AisleLocation).AssemblyQualifiedName)
        {
            return Simple.ResolveType(id);
        }
        else
        {
            throw new ArgumentException("Unexpected type ID", nameof(id));
        }
    }

    public override string ResolveTypeId(Type type)
    {
        return Simple.ResolveTypeId(type);
    }
}

public class BookRecord
{
    public string Title { get; set; }
    public string Author { get; set; }
    public int PageCount { get; set; }
    public AisleLocation Location { get; set; }
}

public class AisleLocation
{
    public char Aisle { get; set; }
    public byte Shelf { get; set; }
}

public class ExampleClass
{
    public BookRecord DeserializeBookRecord(string s)
    {
        JavaScriptSerializer serializer = new JavaScriptSerializer(new BookRecordTypeResolver());
        return serializer.Deserialize<BookRecord>(s);
    }
}

```

```

Imports System
Imports System.Web.Script.Serialization

Public Class BookRecordTypeResolver
    Inherits JavaScriptTypeResolver

    ' For compatibility with data serialized with a JavaScriptSerializer initialized with
    ' SimpleTypeResolver.
    Private Dim Simple As SimpleTypeResolver = New SimpleTypeResolver()

    Public Overrides Function ResolveType(id As String) As Type
        ' One way to discover expected types is through testing deserialization
        ' of **valid** data and logging the types used.

        ''Console.WriteLine($"ResolveType('{id}')")

        If id = GetType(BookRecord).AssemblyQualifiedName Or id =
        GetType(AisleLocation).AssemblyQualifiedName Then
            Return Simple.ResolveType(id)
        Else
            Throw New ArgumentException("Unexpected type", NameOf(id))
        End If
    End Function

    Public Overrides Function ResolveTypeId(type As Type) As String
        Return Simple.ResolveTypeId(type)
    End Function
End Class

Public Class BookRecord
    Public Property Title As String
    Public Property Author As String
    Public Property Location As AisleLocation
End Class

Public Class AisleLocation
    Public Property Aisle As Char
    Public Property Shelf As Byte
End Class

Public Class ExampleClass
    Public Function DeserializeBookRecord(str As String) As BookRecord
        Dim serializer As JavaScriptSerializer = New JavaScriptSerializer(New BookRecordTypeResolver())
        Return serializer.Deserialize(Of BookRecord)(str)
    End Function
End Class

```

Related rules

[CA2321: Do not deserialize with JavaScriptSerializer using a SimpleTypeResolver](#)

CA2326: Do not use TypeNameHandling values other than None

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA2326
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

This rule fires when either of the following conditions are met:

- A `Newtonsoft.Json.TypeNameHandling` enumeration value, other than `None`, is referenced.
- An integer value representing a non-zero value is assigned to a `TypeNameHandling` variable.

Rule description

Insecure deserializers are vulnerable when deserializing untrusted data. An attacker could modify the serialized data to include unexpected types to inject objects with malicious side effects. An attack against an insecure deserializer could, for example, execute commands on the underlying operating system, communicate over the network, or delete files.

This rule finds `Newtonsoft.Json.TypeNameHandling` values other than `None`. If you want to deserialize only when a `Newtonsoft.Json.Serialization.ISerializationBinder` is specified to restrict serialized types, disable this rule and enable rules [CA2327](#), [CA2328](#), [CA2329](#), and [CA2330](#) instead.

How to fix violations

- Use `TypeNameHandling`'s `None` value, if possible.
- Make the serialized data tamper-proof. After serialization, cryptographically sign the serialized data. Before deserialization, validate the cryptographic signature. Protect the cryptographic key from being disclosed and design for key rotations.
- Restrict serialized types. Implement a custom `Newtonsoft.Json.Serialization.ISerializationBinder`. Before deserializing with Json.NET, ensure your custom `ISerializationBinder` is specified in the `Newtonsoft.Json.JsonSerializerSettings.SerializationBinder` property. In the overridden `Newtonsoft.Json.Serialization.ISerializationBinder.BindToType` method, if the type is unexpected, return `null` or throw an exception to stop deserialization.
 - If you restrict serialized types, you may want to disable this rule and enable rules [CA2327](#), [CA2328](#), [CA2329](#), and [CA2330](#). Rules [CA2327](#), [CA2328](#), [CA2329](#), and [CA2330](#) help to ensure that you use an `ISerializationBinder` when using `TypeNameHandling` values other than `None`.

When to suppress warnings

It's safe to suppress a warning from this rule if:

- You know the input is trusted. Consider that your application's trust boundary and data flows may change over time.
- You've taken one of the precautions in [How to fix violations](#).

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA2326
// The code that's violating the rule is on this line.
#pragma warning restore CA2326
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA2326.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Pseudo-code examples

Violation

```
using Newtonsoft.Json;

public class ExampleClass
{
    public JsonSerializerSettings Settings { get; }

    public ExampleClass()
    {
        Settings = new JsonSerializerSettings();
        Settings.TypeNameHandling = TypeNameHandling.All;      // CA2326 violation.
    }
}
```

```
Imports Newtonsoft.Json

Public Class ExampleClass
    Public ReadOnly Property Settings() As JsonSerializerSettings

        Public Sub New()
            Settings = New JsonSerializerSettings()
            Settings.TypeNameHandling = TypeNameHandling.All      ' CA2326 violation.
        End Sub
    End Class

```

Solution

```
using Newtonsoft.Json;

public class ExampleClass
{
    public JsonSerializerSettings Settings { get; }

    public ExampleClass()
    {
        Settings = new JsonSerializerSettings();
        // The default value of Settings.TypeNameHandling is TypeNameHandling.None.
    }
}
```

```
Imports Newtonsoft.Json

Public Class ExampleClass
    Public ReadOnly Property Settings() As JsonSerializerSettings

        Public Sub New()
            Settings = New JsonSerializerSettings()

            ' The default value of Settings.TypeNameHandling is TypeNameHandling.None.
        End Sub
    End Class
```

Related rules

[CA2327: Do not use insecure JsonSerializerSettings](#)

[CA2328: Ensure that JsonSerializerSettings are secure](#)

[CA2329: Do not deserialize with JsonSerializer using an insecure configuration](#)

[CA2330: Ensure that JsonSerializer has a secure configuration when deserializing](#)

CA2327: Do not use insecure JsonSerializerSettings

9/20/2022 • 7 minutes to read • [Edit Online](#)

	Value
Rule ID	CA2327
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

This rule fires when both of the following conditions are true for a `Newtonsoft.Json.JsonSerializerSettings` instance:

- The `TypeNameHandling` property is a value other than `None`.
- The `SerializationBinder` property is null.

The `JsonSerializerSettings` instance must be used in one of the following ways:

- Initialized as a class field or property.
- Returned by a method.
- Passed to `JsonSerializer.Create` or `JsonSerializer.CreateDefault`.
- Passed to a `JsonConvert` method that has a `JsonSerializerSettings` parameter.

By default, this rule analyzes the entire codebase, but this is [configurable](#).

Rule description

Insecure deserializers are vulnerable when deserializing untrusted data. An attacker could modify the serialized data to include unexpected types to inject objects with malicious side effects. An attack against an insecure deserializer could, for example, execute commands on the underlying operating system, communicate over the network, or delete files.

This rule finds `Newtonsoft.Json.JsonSerializerSettings` instances that are configured to deserialize types specified from input, but not configured to restrict deserialized types with a `Newtonsoft.Json.Serialization.ISerializationBinder`. If you want to disallow deserialization of types specified from input completely, disable rules CA2327, [CA2328](#), [CA2329](#), and [CA2330](#), and enable rule [CA2326](#) instead.

How to fix violations

- Use `TypeNameHandling`'s `None` value, if possible.
- Make the serialized data tamper-proof. After serialization, cryptographically sign the serialized data. Before deserialization, validate the cryptographic signature. Protect the cryptographic key from being disclosed and design for key rotations.
- Restrict deserialized types. Implement a custom `Newtonsoft.Json.Serialization.ISerializationBinder`. Before deserializing with Json.NET, ensure your custom `ISerializationBinder` is specified in the `Newtonsoft.Json.JsonSerializerSettings.SerializationBinder` property. In the overridden

`Newtonsoft.Json.Serialization.ISerializationBinder.BindToType` method, if the type is unexpected, return `null` or throw an exception to stop deserialization.

When to suppress warnings

It's safe to suppress a warning from this rule if:

- You know the input is trusted. Consider that your application's trust boundary and data flows may change over time.
- You've taken one of the precautions in [How to fix violations](#).

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA2327
// The code that's violating the rule is on this line.
#pragma warning restore CA2327
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA2327.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

Use the following options to configure which parts of your codebase to run this rule on.

- [Exclude specific symbols](#)
- [Exclude specific types and their derived types](#)

You can configure these options for just this rule, for all rules, or for all rules in this category ([Security](#)). For more information, see [Code quality rule configuration options](#).

Exclude specific symbols

You can exclude specific symbols, such as types and methods, from analysis. For example, to specify that the rule should not run on any code within types named `MyType`, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Symbol name only (includes all symbols with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#). Each symbol name requires a symbol-kind

prefix, such as `M:` for methods, `T:` for types, and `N:` for namespaces.

- `.ctor` for constructors and `.cctor` for static constructors.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType</code>	Matches all symbols named <code>MyType</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType1 MyType2</code>	Matches all symbols named either <code>MyType1</code> or <code>MyType2</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS.MyType.MyMethod(ParamType)</code>	Matches specific method <code>MyMethod</code> with the specified fully qualified signature.
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS1.MyType1.MyMethod1(ParamType) M:NS2.MyType2.MyMethod2(ParamType)</code>	Matches specific methods <code>MyMethod1</code> and <code>MyMethod2</code> with the respective fully qualified signatures.

Exclude specific types and their derived types

You can exclude specific types and their derived types from analysis. For example, to specify that the rule should not run on any methods within types named `MyType` and their derived types, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Type name only (includes all types with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#), with an optional `T:` prefix.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType</code>	Matches all types named <code>MyType</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType1 MyType2</code>	Matches all types named either <code>MyType1</code> or <code>MyType2</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS.MyType</code>	Matches specific type <code>MyType</code> with given fully qualified name and all of its derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS1.MyType1 M:NS2.MyType2</code>	Matches specific types <code>MyType1</code> and <code>MyType2</code> with the respective fully qualified names, and all of their derived types.

Pseudo-code examples

Violation

```
using Newtonsoft.Json;

public class BookRecord
{
    public string Title { get; set; }
    public object Location { get; set; }
}

public abstract class Location
{
    public string StoreId { get; set; }
}

public class AisleLocation : Location
{
    public char Aisle { get; set; }
    public byte Shelf { get; set; }
}

public class WarehouseLocation : Location
{
    public string Bay { get; set; }
    public byte Shelf { get; set; }
}

public class ExampleClass
{
    public BookRecord DeserializeBookRecord(string s)
    {
        JsonSerializerSettings settings = new JsonSerializerSettings();
        settings.TypeNameHandling = TypeNameHandling.Auto;
        return JsonConvert.DeserializeObject<BookRecord>(s, settings); // CA2327 violation
    }
}
```

```
Imports Newtonsoft.Json

Public Class BookRecord
    Public Property Title As String
    Public Property Location As Location
End Class

Public MustInherit Class Location
    Public Property StoreId As String
End Class

Public Class AisleLocation
    Inherits Location

    Public Property Aisle As Char
    Public Property Shelf As Byte
End Class

Public Class WarehouseLocation
    Inherits Location

    Public Property Bay As String
    Public Property Shelf As Byte
End Class

Public Class ExampleClass
    Public Function DeserializeBookRecord(s As String) As BookRecord
        Dim settings As JsonSerializerSettings = New JsonSerializerSettings()
        settings.TypeNameHandling = TypeNameHandling.Auto
        Return JsonConvert.DeserializeObject(Of BookRecord)(s, settings)      ' CA2327 violation
    End Function
End Class
```

Solution

```

using System;
using Newtonsoft.Json;
using Newtonsoft.Json.Serialization;

public class BookRecordSerializationBinder : ISerializationBinder
{
    // To maintain backwards compatibility with serialized data before using an ISerializationBinder.
    private static readonly DefaultSerializationBinder Binder = new DefaultSerializationBinder();

    public void BindToName(Type serializedType, out string assemblyName, out string typeName)
    {
        Binder.BindToName(serializedType, out assemblyName, out typeName);
    }

    public Type BindToType(string assemblyName, string typeName)
    {
        // If the type isn't expected, then stop deserialization.
        if (typeName != "BookRecord" && typeName != "AisleLocation" && typeName != "WarehouseLocation")
        {
            return null;
        }

        return Binder.BindToType(assemblyName, typeName);
    }
}

public class BookRecord
{
    public string Title { get; set; }
    public object Location { get; set; }
}

public abstract class Location
{
    public string StoreId { get; set; }
}

public class AisleLocation : Location
{
    public char Aisle { get; set; }
    public byte Shelf { get; set; }
}

public class WarehouseLocation : Location
{
    public string Bay { get; set; }
    public byte Shelf { get; set; }
}

public class ExampleClass
{
    public BookRecord DeserializeBookRecord(string s)
    {
        JsonSerializerSettings settings = new JsonSerializerSettings();
        settings.TypeNameHandling = TypeNameHandling.Auto;
        settings.SerializationBinder = new BookRecordSerializationBinder();
        return JsonConvert.DeserializeObject<BookRecord>(s, settings);
    }
}

```

```

Imports System
Imports Newtonsoft.Json
Imports Newtonsoft.Json.Serialization

Public Class BookRecordSerializationBinder
    Implements ISerializationBinder

        ' To maintain backwards compatibility with serialized data before using an ISerializationBinder.
        Private Shared ReadOnly Property Binder As New DefaultSerializationBinder()

        Public Sub BindToName(serializedType As Type, ByRef assemblyName As String, ByRef typeName As String)
            Implements ISerializationBinder.BindToName
            Binder.BindToName(serializedType, assemblyName, typeName)
        End Sub

        Public Function BindToType(assemblyName As String, typeName As String) As Type Implements
            ISerializationBinder.BindToType
            ' If the type isn't expected, then stop deserialization.
            If typeName <> "BookRecord" AndAlso typeName <> "AisleLocation" AndAlso typeName <>
                "WarehouseLocation" Then
                Return Nothing
            End If

            Return Binder.BindToType(assemblyName, typeName)
        End Function
    End Class

    Public Class BookRecord
        Public Property Title As String
        Public Property Location As Location
    End Class

    Public MustInherit Class Location
        Public Property StoreId As String
    End Class

    Public Class AisleLocation
        Inherits Location

        Public Property Aisle As Char
        Public Property Shelf As Byte
    End Class

    Public Class WarehouseLocation
        Inherits Location

        Public Property Bay As String
        Public Property Shelf As Byte
    End Class

    Public Class ExampleClass
        Public Function DeserializeBookRecord(s As String) As BookRecord
            Dim settings As JsonSerializerSettings = New JsonSerializerSettings()
            settings.TypeNameHandling = TypeNameHandling.Auto
            settings.SerializationBinder = New BookRecordSerializationBinder()
            Return JsonConvert.DeserializeObject(Of BookRecord)(s, settings)
        End Function
    End Class

```

Related rules

[CA2326: Do not use TypeNameHandling values other than None](#)

[CA2328: Ensure that JsonSerializerSettings are secure](#)

CA2329: Do not deserialize with JsonSerializer using an insecure configuration

CA2330: Ensure that JsonSerializer has a secure configuration when deserializing

CA2328: Ensure that JsonSerializerSettings are secure

9/20/2022 • 8 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA2328
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

This rule fires when both of the following conditions *might be* true for a `Newtonsoft.Json.JsonSerializerSettings` instance:

- The `TypeNameHandling` property is a value other than `None`.
- The `SerializationBinder` property is null.

The `JsonSerializerSettings` instance must be used in one of the following ways:

- Initialized as a class field or property.
- Returned by a method.
- Passed to `JsonSerializer.Create` or `JsonSerializer.CreateDefault`.
- Passed to a `JsonConvert` method that has a `JsonSerializerSettings` parameter.

This rule is similar to [CA2327](#), but in this case, analysis can't definitively determine if the settings are insecure.

By default, this rule analyzes the entire codebase, but this is [configurable](#).

Rule description

Insecure deserializers are vulnerable when deserializing untrusted data. An attacker could modify the serialized data to include unexpected types to inject objects with malicious side effects. An attack against an insecure deserializer could, for example, execute commands on the underlying operating system, communicate over the network, or delete files.

This rule finds `Newtonsoft.Json.JsonSerializerSettings` instances that might be configured to deserialize types specified from input, but may not be configured to restrict deserialized types with a `Newtonsoft.Json.Serialization.ISerializationBinder`. If you want to disallow deserialization of types specified from input completely, disable rules [CA2327](#), CA2328, [CA2329](#), and [CA2330](#), and enable rule [CA2326](#) instead.

How to fix violations

- Use `TypeNameHandling`'s `None` value, if possible.
- Make the serialized data tamper-proof. After serialization, cryptographically sign the serialized data. Before deserialization, validate the cryptographic signature. Protect the cryptographic key from being disclosed and design for key rotations.

- Restrict deserialized types. Implement a custom `Newtonsoft.Json.Serialization.ISerializationBinder`. Before deserializing with Json.NET, ensure your custom `ISerializationBinder` is specified in the `Newtonsoft.Json.JsonSerializerSettings.SerializationBinder` property. In the overridden `Newtonsoft.Json.Serialization.ISerializationBinder.BindToType` method, if the type is unexpected, return `null` or throw an exception to stop deserialization.

When to suppress warnings

It's safe to suppress a warning from this rule if:

- You know the input is trusted. Consider that your application's trust boundary and data flows may change over time.
- You've taken one of the precautions in [How to fix violations](#).
- You know that the `Newtonsoft.Json.JsonSerializerSettings.SerializationBinder` property is always set when `TypeNameHandling` property is a value other than `None`.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA2328
// The code that's violating the rule is on this line.
#pragma warning restore CA2328
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA2328.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

Use the following options to configure which parts of your codebase to run this rule on.

- [Exclude specific symbols](#)
- [Exclude specific types and their derived types](#)

You can configure these options for just this rule, for all rules, or for all rules in this category ([Security](#)). For more information, see [Code quality rule configuration options](#).

Exclude specific symbols

You can exclude specific symbols, such as types and methods, from analysis. For example, to specify that the rule should not run on any code within types named `MyType`, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Symbol name only (includes all symbols with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#). Each symbol name requires a symbol-kind prefix, such as `M:` for methods, `T:` for types, and `N:` for namespaces.
- `.ctor` for constructors and `.cctor` for static constructors.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType</code>	Matches all symbols named <code>MyType</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType1 MyType2</code>	Matches all symbols named either <code>MyType1</code> or <code>MyType2</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS.MyType.MyMethod(ParamType)</code>	Matches specific method <code>MyMethod</code> with the specified fully qualified signature.
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS1.MyType1.MyMethod1(ParamType) M:NS2.MyType2.MyMethod2(ParamType)</code>	Matches specific methods <code>MyMethod1</code> and <code>MyMethod2</code> with the respective fully qualified signatures.

Exclude specific types and their derived types

You can exclude specific types and their derived types from analysis. For example, to specify that the rule should not run on any methods within types named `MyType` and their derived types, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Type name only (includes all types with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#), with an optional `T:` prefix.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType</code>	Matches all types named <code>MyType</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType1 MyType2</code>	Matches all types named either <code>MyType1</code> or <code>MyType2</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS.MyType</code>	Matches specific type <code>MyType</code> with given fully qualified name and all of its derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS1.MyType1 M:NS2.MyType2</code>	Matches specific types <code>MyType1</code> and <code>MyType2</code> with the respective fully qualified names, and all of their derived types.

Pseudo-code examples

Violation

```

using System;
using Newtonsoft.Json;
using Newtonsoft.Json.Serialization;

public class BookRecordSerializationBinder : ISerializationBinder
{
    // To maintain backwards compatibility with serialized data before using an ISerializationBinder.
    private static readonly DefaultSerializationBinder Binder = new DefaultSerializationBinder();

    public void BindToName(Type serializedType, out string assemblyName, out string typeName)
    {
        Binder.BindToName(serializedType, out assemblyName, out typeName);
    }

    public Type BindToType(string assemblyName, string typeName)
    {
        // If the type isn't expected, then stop deserialization.
        if (typeName != "BookRecord" && typeName != "AisleLocation" && typeName != "WarehouseLocation")
        {
            return null;
        }

        return Binder.BindToType(assemblyName, typeName);
    }
}

public class BookRecord
{
    public string Title { get; set; }
    public object Location { get; set; }
}

public abstract class Location
{
    public string StoreId { get; set; }
}

public class AisleLocation : Location
{
    public char Aisle { get; set; }
    public byte Shelf { get; set; }
}

public class WarehouseLocation : Location
{
    public string Bay { get; set; }
    public byte Shelf { get; set; }
}

public static class Binders
{
    public static ISerializationBinder BookRecord = new BookRecordSerializationBinder();
}

public class ExampleClass
{
    public BookRecord DeserializeBookRecord(string s)
    {
        JsonSerializerSettings settings = new JsonSerializerSettings();
        settings.TypeNameHandling = TypeNameHandling.Auto;
        settings.SerializationBinder = Binders.BookRecord;
        return JsonConvert.DeserializeObject<BookRecord>(s, settings);    // CA2328 -- settings might be
null
    }
}

```

```

Imports System
Imports Newtonsoft.Json
Imports Newtonsoft.Json.Serialization

Public Class BookRecordSerializationBinder
    Implements ISerializationBinder

        ' To maintain backwards compatibility with serialized data before using an ISerializationBinder.
        Private Shared ReadOnly Property Binder As New DefaultSerializationBinder()

        Public Sub BindToName(serializedType As Type, ByRef assemblyName As String, ByRef typeName As String)
            Implements ISerializationBinder.BindToName
            Binder.BindToName(serializedType, assemblyName, typeName)
        End Sub

        Public Function BindToType(assemblyName As String, typeName As String) As Type Implements
            ISerializationBinder.BindToType
            ' If the type isn't expected, then stop deserialization.
            If typeName <> "BookRecord" AndAlso typeName <> "AisleLocation" AndAlso typeName <>
                "WarehouseLocation" Then
                Return Nothing
            End If

            Return Binder.BindToType(assemblyName, typeName)
        End Function
    End Class

    Public Class BookRecord
        Public Property Title As String
        Public Property Location As Location
    End Class

    Public MustInherit Class Location
        Public Property StoreId As String
    End Class

    Public Class AisleLocation
        Inherits Location

        Public Property Aisle As Char
        Public Property Shelf As Byte
    End Class

    Public Class WarehouseLocation
        Inherits Location

        Public Property Bay As String
        Public Property Shelf As Byte
    End Class

    Public Class Binders
        Public Shared Property BookRecord As ISerializationBinder = New BookRecordSerializationBinder()
    End Class

    Public Class ExampleClass
        Public Function DeserializeBookRecord(s As String) As BookRecord
            Dim settings As JsonSerializerSettings = New JsonSerializerSettings()
            settings.TypeNameHandling = TypeNameHandling.Auto
            settings.SerializationBinder = Binders.BookRecord
            Return JsonConvert.DeserializeObject(Of BookRecord)(s, settings)      ' CA2328 -- settings might be
        Nothing
        End Function
    End Class

```

Solution

```

using System;
using Newtonsoft.Json;
using Newtonsoft.Json.Serialization;

public class BookRecordSerializationBinder : ISerializationBinder
{
    // To maintain backwards compatibility with serialized data before using an ISerializationBinder.
    private static readonly DefaultSerializationBinder Binder = new DefaultSerializationBinder();

    public void BindToName(Type serializedType, out string assemblyName, out string typeName)
    {
        Binder.BindToName(serializedType, out assemblyName, out typeName);
    }

    public Type BindToType(string assemblyName, string typeName)
    {
        // If the type isn't expected, then stop deserialization.
        if (typeName != "BookRecord" && typeName != "AisleLocation" && typeName != "WarehouseLocation")
        {
            return null;
        }

        return Binder.BindToType(assemblyName, typeName);
    }
}

public class BookRecord
{
    public string Title { get; set; }
    public object Location { get; set; }
}

public abstract class Location
{
    public string StoreId { get; set; }
}

public class AisleLocation : Location
{
    public char Aisle { get; set; }
    public byte Shelf { get; set; }
}

public class WarehouseLocation : Location
{
    public string Bay { get; set; }
    public byte Shelf { get; set; }
}

public static class Binders
{
    public static ISerializationBinder BookRecord = new BookRecordSerializationBinder();
}

public class ExampleClass
{
    public BookRecord DeserializeBookRecord(string s)
    {
        JsonSerializerSettings settings = new JsonSerializerSettings();
        settings.TypeNameHandling = TypeNameHandling.Auto;

        // Ensure that SerializationBinder is non-null before deserializing
        settings.SerializationBinder = Binders.BookRecord ?? throw new Exception("Expected non-null");

        return JsonConvert.DeserializeObject<BookRecord>(s, settings);
    }
}

```

```

Imports System
Imports Newtonsoft.Json
Imports Newtonsoft.Json.Serialization

Public Class BookRecordSerializationBinder
    Implements ISerializationBinder

        ' To maintain backwards compatibility with serialized data before using an ISerializationBinder.
        Private Shared ReadOnly Property Binder As New DefaultSerializationBinder()

        Public Sub BindToName(serializedType As Type, ByRef assemblyName As String, ByRef typeName As String)
            Implements ISerializationBinder.BindToName
            Binder.BindToName(serializedType, assemblyName, typeName)
        End Sub

        Public Function BindToType(assemblyName As String, typeName As String) As Type Implements
            ISerializationBinder.BindToType
                ' If the type isn't expected, then stop deserialization.
                If typeName <> "BookRecord" AndAlso typeName <> "AisleLocation" AndAlso typeName <>
                    "WarehouseLocation" Then
                    Return Nothing
                End If

                Return Binder.BindToType(assemblyName, typeName)
            End Function
        End Class

        Public Class BookRecord
            Public Property Title As String
            Public Property Location As Location
        End Class

        Public MustInherit Class Location
            Public Property StoreId As String
        End Class

        Public Class AisleLocation
            Inherits Location

            Public Property Aisle As Char
            Public Property Shelf As Byte
        End Class

        Public Class WarehouseLocation
            Inherits Location

            Public Property Bay As String
            Public Property Shelf As Byte
        End Class

        Public Class Binders
            Public Shared Property BookRecord As ISerializationBinder = New BookRecordSerializationBinder()
        End Class

        Public Class ExampleClass
            Public Function DeserializeBookRecord(s As String) As BookRecord
                Dim settings As JsonSerializerSettings = New JsonSerializerSettings()
                settings.TypeNameHandling = TypeNameHandling.Auto

                ' Ensure that SerializationBinder is non-null before deserializing
                settings.SerializationBinder = If(Binders.BookRecord, New Exception("Expected non-null"))

                Return JsonConvert.DeserializeObject(Of BookRecord)(s, settings)
            End Function
        End Class
    End Class

```

Related rules

[CA2326: Do not use TypeNameHandling values other than None](#)

[CA2327: Do not use insecure JsonSerializerSettings](#)

[CA2329: Do not deserialize with JsonSerializer using an insecure configuration](#)

[CA2330: Ensure that JsonSerializer has a secure configuration when deserializing](#)

CA2329: Do not deserialize with JsonSerializer using an insecure configuration

9/20/2022 • 6 minutes to read • [Edit Online](#)

	Value
Rule ID	CA2329
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

This rule fires when both of the following conditions are true for a `Newtonsoft.Json.JsonSerializer` instance that's passed to a deserialization method or initialized as a field or property:

- The `TypeNameHandling` property is a value other than `None`.
- The `SerializationBinder` property is null.

By default, this rule analyzes the entire codebase, but this is [configurable](#).

Rule description

Insecure deserializers are vulnerable when deserializing untrusted data. An attacker could modify the serialized data to include unexpected types to inject objects with malicious side effects. An attack against an insecure deserializer could, for example, execute commands on the underlying operating system, communicate over the network, or delete files.

This rule finds `Newtonsoft.Json.JsonSerializer` instances that are configured to deserialize types specified from input, but not configured to restrict serialized types with a `Newtonsoft.Json.Serialization.ISerializationBinder`. If you want to disallow deserialization of types specified from input completely, disable rules [CA2327](#), [CA2328](#), [CA2329](#), and [CA2330](#), and enable rule [CA2326](#) instead.

How to fix violations

- Use `TypeNameHandling`'s `None` value, if possible.
- Make the serialized data tamper-proof. After serialization, cryptographically sign the serialized data. Before deserialization, validate the cryptographic signature. Protect the cryptographic key from being disclosed and design for key rotations.
- Restrict serialized types. Implement a custom `Newtonsoft.Json.Serialization.ISerializationBinder`. Before deserializing with Json.NET, ensure your custom `ISerializationBinder` is specified in the `Newtonsoft.Json.JsonSerializer.SerializationBinder` property. In the overridden `Newtonsoft.Json.Serialization.ISerializationBinder.BindToType` method, if the type is unexpected, return `null` or throw an exception to stop deserialization.

When to suppress warnings

It's safe to suppress a warning from this rule if:

- You know the input is trusted. Consider that your application's trust boundary and data flows may change over time.
- You've taken one of the precautions in [How to fix violations](#).

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA2329
// The code that's violating the rule is on this line.
#pragma warning restore CA2329
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA2329.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

Use the following options to configure which parts of your codebase to run this rule on.

- [Exclude specific symbols](#)
- [Exclude specific types and their derived types](#)

You can configure these options for just this rule, for all rules, or for all rules in this category ([Security](#)). For more information, see [Code quality rule configuration options](#).

Exclude specific symbols

You can exclude specific symbols, such as types and methods, from analysis. For example, to specify that the rule should not run on any code within types named `MyType`, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXX.excluded_symbol_names = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Symbol name only (includes all symbols with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#). Each symbol name requires a symbol-kind prefix, such as `M:` for methods, `T:` for types, and `N:` for namespaces.
- `.ctor` for constructors and `.cctor` for static constructors.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType</code>	Matches all symbols named <code>MyType</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType1 MyType2</code>	Matches all symbols named either <code>MyType1</code> or <code>MyType2</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS.MyType.MyMethod(ParamType)</code>	Matches specific method <code>MyMethod</code> with the specified fully qualified signature.
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS1.MyType1.MyMethod1(ParamType) M:NS2.MyType2.MyMethod2(ParamType)</code>	Matches specific methods <code>MyMethod1</code> and <code>MyMethod2</code> with the respective fully qualified signatures.

Exclude specific types and their derived types

You can exclude specific types and their derived types from analysis. For example, to specify that the rule should not run on any methods within types named `MyType` and their derived types, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Type name only (includes all types with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#), with an optional `T:` prefix.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType</code>	Matches all types named <code>MyType</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType1 MyType2</code>	Matches all types named either <code>MyType1</code> or <code>MyType2</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS.MyType</code>	Matches specific type <code>MyType</code> with given fully qualified name and all of its derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS1.MyType1 M:NS2.MyType2</code>	Matches specific types <code>MyType1</code> and <code>MyType2</code> with the respective fully qualified names, and all of their derived types.

Pseudo-code examples

Violation

```
using Newtonsoft.Json;

public class BookRecord
{
    public string Title { get; set; }
    public object Location { get; set; }
}

public abstract class Location
{
    public string StoreId { get; set; }
}

public class AisleLocation : Location
{
    public char Aisle { get; set; }
    public byte Shelf { get; set; }
}

public class WarehouseLocation : Location
{
    public string Bay { get; set; }
    public byte Shelf { get; set; }
}

public class ExampleClass
{
    public BookRecord DeserializeBookRecord(JsonReader reader)
    {
        JsonSerializer jsonSerializer = new JsonSerializer();
        jsonSerializer.TypeNameHandling = TypeNameHandling.Auto;
        return jsonSerializer.Deserialize<BookRecord>(reader);      // CA2329 violation
    }
}
```

```
Imports Newtonsoft.Json

Public Class BookRecord
    Public Property Title As String
    Public Property Location As Location
End Class

Public MustInherit Class Location
    Public Property StoreId As String
End Class

Public Class AisleLocation
    Inherits Location

    Public Property Aisle As Char
    Public Property Shelf As Byte
End Class

Public Class WarehouseLocation
    Inherits Location

    Public Property Bay As String
    Public Property Shelf As Byte
End Class

Public Class ExampleClass
    Public Function DeserializeBookRecord(reader As JsonReader) As BookRecord
        Dim jsonSerializer As JsonSerializer = New JsonSerializer()
        jsonSerializer.TypeNameHandling = TypeNameHandling.Auto
        Return JsonSerializer.Deserialize(Of BookRecord)(reader)      ' CA2329 violation
    End Function
End Class
```

Solution

```

using System;
using Newtonsoft.Json;
using Newtonsoft.Json.Serialization;

public class BookRecordSerializationBinder : ISerializationBinder
{
    // To maintain backwards compatibility with serialized data before using an ISerializationBinder.
    private static readonly DefaultSerializationBinder Binder = new DefaultSerializationBinder();

    public void BindToName(Type serializedType, out string assemblyName, out string typeName)
    {
        Binder.BindToName(serializedType, out assemblyName, out typeName);
    }

    public Type BindToType(string assemblyName, string typeName)
    {
        // If the type isn't expected, then stop deserialization.
        if (typeName != "BookRecord" && typeName != "AisleLocation" && typeName != "WarehouseLocation")
        {
            return null;
        }

        return Binder.BindToType(assemblyName, typeName);
    }
}

public class BookRecord
{
    public string Title { get; set; }
    public object Location { get; set; }
}

public abstract class Location
{
    public string StoreId { get; set; }
}

public class AisleLocation : Location
{
    public char Aisle { get; set; }
    public byte Shelf { get; set; }
}

public class WarehouseLocation : Location
{
    public string Bay { get; set; }
    public byte Shelf { get; set; }
}

public class ExampleClass
{
    public BookRecord DeserializeBookRecord(JsonReader reader)
    {
        JsonSerializer jsonSerializer = new JsonSerializer();
        jsonSerializer.TypeNameHandling = TypeNameHandling.Auto;
        jsonSerializer.SerializationBinder = new BookRecordSerializationBinder();
        return jsonSerializer.Deserialize<BookRecord>(reader);
    }
}

```

```

Imports System
Imports Newtonsoft.Json
Imports Newtonsoft.Json.Serialization

Public Class BookRecordSerializationBinder
    Implements ISerializationBinder

        ' To maintain backwards compatibility with serialized data before using an ISerializationBinder.
        Private Shared ReadOnly Property Binder As New DefaultSerializationBinder()

        Public Sub BindToName(serializedType As Type, ByRef assemblyName As String, ByRef typeName As String)
            Implements ISerializationBinder.BindToName
            Binder.BindToName(serializedType, assemblyName, typeName)
        End Sub

        Public Function BindToType(assemblyName As String, typeName As String) As Type Implements
            ISerializationBinder.BindToType
            ' If the type isn't expected, then stop deserialization.
            If typeName <> "BookRecord" AndAlso typeName <> "AisleLocation" AndAlso typeName <>
                "WarehouseLocation" Then
                Return Nothing
            End If

            Return Binder.BindToType(assemblyName, typeName)
        End Function
    End Class

    Public Class BookRecord
        Public Property Title As String
        Public Property Location As Location
    End Class

    Public MustInherit Class Location
        Public Property StoreId As String
    End Class

    Public Class AisleLocation
        Inherits Location

        Public Property Aisle As Char
        Public Property Shelf As Byte
    End Class

    Public Class WarehouseLocation
        Inherits Location

        Public Property Bay As String
        Public Property Shelf As Byte
    End Class

    Public Class ExampleClass
        Public Function DeserializeBookRecord(reader As JsonReader) As BookRecord
            Dim jsonSerializer As JsonSerializer = New JsonSerializer()
            jsonSerializer.TypeNameHandling = TypeNameHandling.Auto
            jsonSerializer.SerializationBinder = New BookRecordSerializationBinder()
            Return jsonSerializer.Deserialize(Of BookRecord)(reader)
        End Function
    End Class

```

Related rules

[CA2326: Do not use TypeNameHandling values other than None](#)

[CA2327: Do not use insecure JsonSerializerSettings](#)

CA2328: Ensure that JsonSerializerSettings are secure

CA2330: Ensure that JsonSerializer has a secure configuration when deserializing

CA2330: Ensure that JsonSerializer has a secure configuration when deserializing

9/20/2022 • 8 minutes to read • [Edit Online](#)

	Value
Rule ID	CA2330
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

This rule fires when both of the following conditions *might be* true for a `Newtonsoft.Json.JsonSerializer` instance that's passed to a deserialization method or initialized as a field or property:

- The `TypeNameHandling` property is a value other than `None`.
- The `SerializationBinder` property is null.

This rule is similar to [CA2329](#), but in this case, analysis can't definitively determine if the serializer is configured insecurely.

By default, this rule analyzes the entire codebase, but this is [configurable](#).

Rule description

Insecure deserializers are vulnerable when deserializing untrusted data. An attacker could modify the serialized data to include unexpected types to inject objects with malicious side effects. An attack against an insecure deserializer could, for example, execute commands on the underlying operating system, communicate over the network, or delete files.

This rule finds `Newtonsoft.Json.JsonSerializer` instances that might be configured to deserialize types specified from input, but may not be configured to restrict deserialized types with a `Newtonsoft.Json.Serialization.ISerializationBinder`. If you want to disallow deserialization of types specified from input completely, disable rules [CA2327](#), [CA2328](#), [CA2329](#), and [CA2330](#), and enable rule [CA2326](#) instead.

How to fix violations

- Use `TypeNameHandling`'s `None` value, if possible.
- Make the serialized data tamper-proof. After serialization, cryptographically sign the serialized data. Before deserialization, validate the cryptographic signature. Protect the cryptographic key from being disclosed and design for key rotations.
- Restrict deserialized types. Implement a custom `Newtonsoft.Json.Serialization.ISerializationBinder`. Before deserializing with Json.NET, ensure your custom `ISerializationBinder` is specified in the `Newtonsoft.Json.JsonSerializer.SerializationBinder` property. In the overridden `Newtonsoft.Json.Serialization.ISerializationBinder.BindToType` method, if the type is unexpected, return `null` or throw an exception to stop deserialization.

When to suppress warnings

It's safe to suppress a warning from this rule if:

- You know the input is trusted. Consider that your application's trust boundary and data flows may change over time.
- You've taken one of the precautions in [How to fix violations](#).
- You know that the `Newtonsoft.Json.JsonSerializer.SerializationBinder` property is always set when `TypeNameHandling` property is a value other than `None`.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA2330
// The code that's violating the rule is on this line.
#pragma warning restore CA2330
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA2330.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

Use the following options to configure which parts of your codebase to run this rule on.

- [Exclude specific symbols](#)
- [Exclude specific types and their derived types](#)

You can configure these options for just this rule, for all rules, or for all rules in this category ([Security](#)). For more information, see [Code quality rule configuration options](#).

Exclude specific symbols

You can exclude specific symbols, such as types and methods, from analysis. For example, to specify that the rule should not run on any code within types named `MyType`, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Symbol name only (includes all symbols with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#). Each symbol name requires a symbol-kind prefix, such as `M:` for methods, `T:` for types, and `N:` for namespaces.

- `.ctor` for constructors and `.cctor` for static constructors.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType</code>	Matches all symbols named <code>MyType</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType1 MyType2</code>	Matches all symbols named either <code>MyType1</code> or <code>MyType2</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS.MyType.MyMethod(ParamType)</code>	Matches specific method <code>MyMethod</code> with the specified fully qualified signature.
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS1.MyType1.MyMethod1(ParamType) M:NS2.MyType2.MyMethod2(ParamType)</code>	Matches specific methods <code>MyMethod1</code> and <code>MyMethod2</code> with the respective fully qualified signatures.

Exclude specific types and their derived types

You can exclude specific types and their derived types from analysis. For example, to specify that the rule should not run on any methods within types named `MyType` and their derived types, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Type name only (includes all types with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#), with an optional `T:` prefix.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType</code>	Matches all types named <code>MyType</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType1 MyType2</code>	Matches all types named either <code>MyType1</code> or <code>MyType2</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS.MyType</code>	Matches specific type <code>MyType</code> with given fully qualified name and all of its derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS1.MyType1 M:NS2.MyType2</code>	Matches specific types <code>MyType1</code> and <code>MyType2</code> with the respective fully qualified names, and all of their derived types.

Pseudo-code examples

Violation

```

using System;
using Newtonsoft.Json;
using Newtonsoft.Json.Serialization;

public class BookRecordSerializationBinder : ISerializationBinder
{
    // To maintain backwards compatibility with serialized data before using an ISerializationBinder.
    private static readonly DefaultSerializationBinder Binder = new DefaultSerializationBinder();

    public void BindToName(Type serializedType, out string assemblyName, out string typeName)
    {
        Binder.BindToName(serializedType, out assemblyName, out typeName);
    }

    public Type BindToType(string assemblyName, string typeName)
    {
        // If the type isn't expected, then stop deserialization.
        if (typeName != "BookRecord" && typeName != "AisleLocation" && typeName != "WarehouseLocation")
        {
            return null;
        }

        return Binder.BindToType(assemblyName, typeName);
    }
}

public class BookRecord
{
    public string Title { get; set; }
    public object Location { get; set; }
}

public abstract class Location
{
    public string StoreId { get; set; }
}

public class AisleLocation : Location
{
    public char Aisle { get; set; }
    public byte Shelf { get; set; }
}

public class WarehouseLocation : Location
{
    public string Bay { get; set; }
    public byte Shelf { get; set; }
}

public static class Binders
{
    public static ISerializationBinder BookRecord = new BookRecordSerializationBinder();
}

public class ExampleClass
{
    public BookRecord DeserializeBookRecord(JsonReader reader)
    {
        JsonSerializer jsonSerializer = new JsonSerializer();
        jsonSerializer.TypeNameHandling = TypeNameHandling.Auto;
        jsonSerializer.SerializationBinder = Binders.BookRecord;
        return jsonSerializer.Deserialize<BookRecord>(reader);      // CA2330 -- SerializationBinder might be
null
    }
}

```

```

Imports System
Imports Newtonsoft.Json
Imports Newtonsoft.Json.Serialization

Public Class BookRecordSerializationBinder
    Implements ISerializationBinder

        ' To maintain backwards compatibility with serialized data before using an ISerializationBinder.
        Private Shared ReadOnly Property Binder As New DefaultSerializationBinder()

        Public Sub BindToName(serializedType As Type, ByRef assemblyName As String, ByRef typeName As String)
            Implements ISerializationBinder.BindToName
            Binder.BindToName(serializedType, assemblyName, typeName)
        End Sub

        Public Function BindToType(assemblyName As String, typeName As String) As Type Implements
            ISerializationBinder.BindToType
            ' If the type isn't expected, then stop deserialization.
            If typeName <> "BookRecord" AndAlso typeName <> "AisleLocation" AndAlso typeName <>
                "WarehouseLocation" Then
                Return Nothing
            End If

            Return Binder.BindToType(assemblyName, typeName)
        End Function
    End Class

    Public Class BookRecord
        Public Property Title As String
        Public Property Location As Location
    End Class

    Public MustInherit Class Location
        Public Property StoreId As String
    End Class

    Public Class AisleLocation
        Inherits Location

        Public Property Aisle As Char
        Public Property Shelf As Byte
    End Class

    Public Class WarehouseLocation
        Inherits Location

        Public Property Bay As String
        Public Property Shelf As Byte
    End Class

    Public Class Binders
        Public Shared Property BookRecord As ISerializationBinder = New BookRecordSerializationBinder()
    End Class

    Public Class ExampleClass
        Public Function DeserializeBookRecord(reader As JsonReader) As BookRecord
            Dim jsonSerializer As JsonSerializer = New JsonSerializer()
            jsonSerializer.TypeNameHandling = TypeNameHandling.Auto
            jsonSerializer.SerializationBinder = Binders.BookRecord
            Return jsonSerializer.Deserialize(Of BookRecord)(reader)      ' CA2330 -- SerializationBinder might be
null
        End Function
    End Class

```

Solution

```

using System;
using Newtonsoft.Json;
using Newtonsoft.Json.Serialization;

public class BookRecordSerializationBinder : ISerializationBinder
{
    // To maintain backwards compatibility with serialized data before using an ISerializationBinder.
    private static readonly DefaultSerializationBinder Binder = new DefaultSerializationBinder();

    public void BindToName(Type serializedType, out string assemblyName, out string typeName)
    {
        Binder.BindToName(serializedType, out assemblyName, out typeName);
    }

    public Type BindToType(string assemblyName, string typeName)
    {
        // If the type isn't expected, then stop deserialization.
        if (typeName != "BookRecord" && typeName != "AisleLocation" && typeName != "WarehouseLocation")
        {
            return null;
        }

        return Binder.BindToType(assemblyName, typeName);
    }
}

public class BookRecord
{
    public string Title { get; set; }
    public object Location { get; set; }
}

public abstract class Location
{
    public string StoreId { get; set; }
}

public class AisleLocation : Location
{
    public char Aisle { get; set; }
    public byte Shelf { get; set; }
}

public class WarehouseLocation : Location
{
    public string Bay { get; set; }
    public byte Shelf { get; set; }
}

public static class Binders
{
    public static ISerializationBinder BookRecord = new BookRecordSerializationBinder();
}

public class ExampleClass
{
    public BookRecord DeserializeBookRecord(JsonReader reader)
    {
        JsonSerializer jsonSerializer = new JsonSerializer();
        jsonSerializer.TypeNameHandling = TypeNameHandling.Auto;

        // Ensure that SerializationBinder is assigned non-null before deserializing
        jsonSerializer.SerializationBinder = Binders.BookRecord ?? throw new Exception("Expected non-null");

        return jsonSerializer.Deserialize<BookRecord>(reader);
    }
}

```

```

Imports System
Imports Newtonsoft.Json
Imports Newtonsoft.Json.Serialization

Public Class BookRecordSerializationBinder
    Implements ISerializationBinder

        ' To maintain backwards compatibility with serialized data before using an ISerializationBinder.
        Private Shared ReadOnly Property Binder As New DefaultSerializationBinder()

        Public Sub BindToName(serializedType As Type, ByRef assemblyName As String, ByRef typeName As String)
            Implements ISerializationBinder.BindToName
            Binder.BindToName(serializedType, assemblyName, typeName)
        End Sub

        Public Function BindToType(assemblyName As String, typeName As String) As Type Implements
            ISerializationBinder.BindToType
                ' If the type isn't expected, then stop deserialization.
                If typeName <> "BookRecord" AndAlso typeName <> "AisleLocation" AndAlso typeName <>
                    "WarehouseLocation" Then
                    Return Nothing
                End If

                Return Binder.BindToType(assemblyName, typeName)
            End Function
        End Class

        Public Class BookRecord
            Public Property Title As String
            Public Property Location As Location
        End Class

        Public MustInherit Class Location
            Public Property StoreId As String
        End Class

        Public Class AisleLocation
            Inherits Location

            Public Property Aisle As Char
            Public Property Shelf As Byte
        End Class

        Public Class WarehouseLocation
            Inherits Location

            Public Property Bay As String
            Public Property Shelf As Byte
        End Class

        Public Class Binders
            Public Shared Property BookRecord As ISerializationBinder = New BookRecordSerializationBinder()
        End Class

        Public Class ExampleClass
            Public Function DeserializeBookRecord(reader As JsonReader) As BookRecord
                Dim jsonSerializer As JsonSerializer = New JsonSerializer()
                jsonSerializer.TypeNameHandling = TypeNameHandling.Auto

                ' Ensure SerializationBinder is non-null before deserializing
                jsonSerializer.SerializationBinder = If(Binders.BookRecord, New Exception("Expected non-null"))

                Return jsonSerializer.Deserialize(Of BookRecord)(reader)
            End Function
        End Class
    End Class

```

Related rules

[CA2326: Do not use TypeNameHandling values other than None](#)

[CA2327: Do not use insecure JsonSerializerSettings](#)

[CA2328: Ensure that JsonSerializerSettings are secure](#)

[CA2329: Do not deserialize with JsonSerializer using an insecure configuration](#)

CA2350: Ensure DataTable.ReadXml()'s input is trusted

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA2350
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

The `DataTable.ReadXml` method was called or referenced.

Rule description

When deserializing a `DataTable` with untrusted input, an attacker can craft malicious input to perform a denial of service attack. There may be unknown remote code execution vulnerabilities.

For more information, see [DataSet and DataTable security guidance](#).

How to fix violations

- If possible, use [Entity Framework](#) rather than the `DataTable`.
- Make the serialized data tamper-proof. After serialization, cryptographically sign the serialized data. Before deserialization, validate the cryptographic signature. Protect the cryptographic key from being disclosed and design for key rotations.

When to suppress warnings

It's safe to suppress a warning from this rule if:

- You know the input is trusted. Consider that your application's trust boundary and data flows may change over time.
- You've taken one of the precautions in [How to fix violations](#).

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA2350
// The code that's violating the rule is on this line.
#pragma warning restore CA2350
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA2350.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Pseudo-code examples

Violation

```
using System.Data;

public class ExampleClass
{
    public DataTable MyDeserialize(string untrustedXml)
    {
        DataTable dt = new DataTable();
        dt.ReadXml(untrustedXml);
    }
}
```

Related rules

[CA2351: Ensure DataSet.ReadXml\(\)'s input is trusted](#)

[CA2352: Unsafe DataSet or DataTable in serializable type can be vulnerable to remote code execution attacks](#)

[CA2353: Unsafe DataSet or DataTable in serializable type](#)

[CA2354: Unsafe DataSet or DataTable in deserialized object graph can be vulnerable to remote code execution attack](#)

[CA2355: Unsafe DataSet or DataTable in deserialized object graph](#)

[CA2356: Unsafe DataSet or DataTable in web deserialized object graph](#)

[CA2361: Ensure autogenerated class containing DataSet.ReadXml\(\) is not used with untrusted data](#)

[CA2362: Unsafe DataSet or DataTable in autogenerated serializable type can be vulnerable to remote code execution attacks](#)

CA2351: Ensure DataSet.ReadXml()'s input is trusted

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA2351
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

The `DataSet.ReadXml` method was called or referenced, and not within autogenerated code.

This rule classifies autogenerated code b:

- Being inside a method named `ReadXmlSerializable`.
- The `ReadXmlSerializable` method has a `System.Diagnostics.DebuggerNonUserCodeAttribute`.
- The `ReadXmlSerializable` method is within a type that has a `System.ComponentModel.DesignerCategoryAttribute`.

CA2361 is a similar rule, for when `DataSet.ReadXml` appears within autogenerated code.

Rule description

When deserializing a `DataSet` with untrusted input, an attacker can craft malicious input to perform a denial of service attack. There may be unknown remote code execution vulnerabilities.

For more information, see [DataSet and DataTable security guidance](#).

How to fix violations

- If possible, use [Entity Framework](#) rather than the `DataSet`.
- Make the serialized data tamper-proof. After serialization, cryptographically sign the serialized data. Before deserialization, validate the cryptographic signature. Protect the cryptographic key from being disclosed and design for key rotations.

When to suppress warnings

It's safe to suppress a warning from this rule if:

- You know the input is trusted. Consider that your application's trust boundary and data flows may change over time.
- You've taken one of the precautions in [How to fix violations](#).

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA2351
// The code that's violating the rule is on this line.
#pragma warning restore CA2351
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA2351.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Pseudo-code examples

Violation

```
using System.Data;

public class ExampleClass
{
    public DataSet MyDeserialize(string untrustedXml)
    {
        DataSet dt = new DataSet();
        dt.ReadXml(untrustedXml);
    }
}
```

Related rules

[CA2350: Ensure DataTable.ReadXml\(\)'s input is trusted](#)

[CA2352: Unsafe DataSet or DataTable in serializable type can be vulnerable to remote code execution attacks](#)

[CA2353: Unsafe DataSet or DataTable in serializable type](#)

[CA2354: Unsafe DataSet or DataTable in deserialized object graph can be vulnerable to remote code execution attack](#)

[CA2355: Unsafe DataSet or DataTable in deserialized object graph](#)

[CA2356: Unsafe DataSet or DataTable in web deserialized object graph](#)

[CA2361: Ensure autogenerated class containing DataSet.ReadXml\(\) is not used with untrusted data](#)

[CA2362: Unsafe DataSet or DataTable in autogenerated serializable type can be vulnerable to remote code execution attacks](#)

CA2352: Unsafe DataSet or DataTable in serializable type can be vulnerable to remote code execution attacks

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA2352
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

A class or struct marked with [SerializableAttribute](#) contains a [DataSet](#) or [DataTable](#) field or property, and doesn't have a [DesignerCategoryAttribute](#).

[CA2362](#) is a similar rule, for when there is a [DesignerCategoryAttribute](#).

Rule description

When deserializing untrusted input with [BinaryFormatter](#) and the deserialized object graph contains a [DataSet](#) or [DataTable](#), an attacker can craft a malicious payload to perform a remote code execution attack.

This rule finds types which are insecure when serialized. If your code doesn't deserialize the types found, then you don't have a deserialization vulnerability.

For more information, see [DataSet and DataTable security guidance](#).

How to fix violations

- If possible, use [Entity Framework](#) rather than [DataSet](#) and [DataTable](#).
- Make the serialized data tamper-proof. After serialization, cryptographically sign the serialized data. Before deserialization, validate the cryptographic signature. Protect the cryptographic key from being disclosed and design for key rotations.

When to suppress warnings

It's safe to suppress a warning from this rule if:

- The type found by this rule is never serialized, either directly or indirectly.
- You know the input is trusted. Consider that your application's trust boundary and data flows may change over time.
- You've taken one of the precautions in [How to fix violations](#).

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then

re-enable the rule.

```
#pragma warning disable CA2352
// The code that's violating the rule is on this line.
#pragma warning restore CA2352
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA2352.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Pseudo-code examples

Violation

```
using System.Data;
using System.Runtime.Serialization;

[Serializable]
public class MyClass
{
    public DataSet MyDataSet { get; set; }
}
```

Related rules

[CA2350: Ensure DataTable.ReadXml\(\)'s input is trusted](#)

[CA2351: Ensure DataSet.ReadXml\(\)'s input is trusted](#)

[CA2353: Unsafe DataSet or DataTable in serializable type](#)

[CA2354: Unsafe DataSet or DataTable in deserialized object graph can be vulnerable to remote code execution attack](#)

[CA2355: Unsafe DataSet or DataTable in deserialized object graph](#)

[CA2356: Unsafe DataSet or DataTable in web deserialized object graph](#)

[CA2361: Ensure autogenerated class containing DataSet.ReadXml\(\) is not used with untrusted data](#)

[CA2362: Unsafe DataSet or DataTable in autogenerated serializable type can be vulnerable to remote code execution attacks](#)

CA2353: Unsafe DataSet or DataTable in serializable type

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA2353
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

A class or struct marked with an XML serialization attribute or a data contract attribute contains a [DataSet](#) or [DataTable](#) field or property.

XML serialization attributes include:

- `XmlAnyAttributeAttribute`
 - `XmlAnyElementAttribute`
 - `XmlArrayAttribute`
 - `XmlArrayItemAttribute`
 - `XmlChoiceIdentifierAttribute`
 - `XmlElementAttribute`
 - `XmlAttributeAttribute`
 - `XmIgnoreAttribute`
 - `XmIncludeAttribute`
 - `XmlRootAttribute`
 - `XmlTextAttribute`
 - `XmlTypeAttribute`

Data contract serialization attributes include:

- `DataContractAttribute`
 - `DataMemberAttribute`
 - `IgnoreDataMemberAttribute`
 - `KnownTypeAttribute`

Rule description

When deserializing untrusted input and the deserialized object graph contains a [DataSet](#) or [DataTable](#), an attacker can craft a malicious payload to perform a denial of service attack. There may be unknown remote code execution vulnerabilities.

This rule finds types which are insecure when deserialized. If your code doesn't deserialize the types found, then you don't have a deserialization vulnerability.

For more information, see [DataSet and DataTable security guidance](#).

How to fix violations

- If possible, use [Entity Framework](#) rather than [DataSet](#) and [DataTable](#).
- Make the serialized data tamper-proof. After serialization, cryptographically sign the serialized data. Before deserialization, validate the cryptographic signature. Protect the cryptographic key from being disclosed and design for key rotations.

When to suppress warnings

It's safe to suppress a warning from this rule if:

- The type found by this rule is never deserialized, either directly or indirectly.
- You know the input is trusted. Consider that your application's trust boundary and data flows may change over time.
- You've taken one of the precautions in [How to fix violations](#).

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA2353
// The code that's violating the rule is on this line.
#pragma warning restore CA2353
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA2353.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Pseudo-code examples

Violation

```
using System.Data;
using System.Runtime.Serialization;

[XmlRoot]
public class MyClass
{
    public DataSet MyDataSet { get; set; }
}
```

Related rules

CA2350: Ensure DataTable.ReadXml()'s input is trusted

CA2351: Ensure DataSet.ReadXml()'s input is trusted

CA2352: Unsafe DataSet or DataTable in serializable type can be vulnerable to remote code execution attacks

CA2354: Unsafe DataSet or DataTable in deserialized object graph can be vulnerable to remote code execution attack

CA2355: Unsafe DataSet or DataTable in deserialized object graph

CA2356: Unsafe DataSet or DataTable in web deserialized object graph

CA2361: Ensure autogenerated class containing DataSet.ReadXml() is not used with untrusted data

CA2362: Unsafe DataSet or DataTable in autogenerated serializable type can be vulnerable to remote code execution attacks

CA2354: Unsafe DataSet or DataTable in deserialized object graph can be vulnerable to remote code execution attack

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA2354
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

Deserializing with an [System.Runtime.Serialization.IFormatter](#) serialized, and the casted type's object graph can include a [DataSet](#) or [DataTable](#).

This rule uses a different approach to a similar rule, [CA2352: Unsafe DataSet or DataTable in serializable type can be vulnerable to remote code execution attacks](#).

Rule description

When deserializing untrusted input with [BinaryFormatter](#) and the deserialized object graph contains a [DataSet](#) or [DataTable](#), an attacker can craft a malicious payload to perform a remote code execution attack.

For more information, see [DataSet and DataTable security guidance](#).

How to fix violations

- If possible, use [Entity Framework](#) rather than [DataSet](#) and [DataTable](#).
- Make the serialized data tamper-proof. After serialization, cryptographically sign the serialized data. Before deserialization, validate the cryptographic signature. Protect the cryptographic key from being disclosed and design for key rotations.

When to suppress warnings

It's safe to suppress a warning from this rule if:

- You know the input is trusted. Consider that your application's trust boundary and data flows may change over time.
- You've taken one of the precautions in [How to fix violations](#).

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA2354
// The code that's violating the rule is on this line.
#pragma warning restore CA2354
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA2354.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Pseudo-code examples

Violation

```
using System.Data;
using System.IO;
using System.Runtime.Serialization;

[Serializable]
public class MyClass
{
    public MyOtherClass OtherClass { get; set; }
}

[Serializable]
public class MyOtherClass
{
    private DataSet myDataSet;
}

public class ExampleClass
{
    public MyClass Deserialize(Stream stream)
    {
        BinaryFormatter bf = new BinaryFormatter();
        return (MyClass) bf.Deserialize(stream);
    }
}
```

Related rules

[CA2350: Ensure DataTable.ReadXml\(\)'s input is trusted](#)

[CA2351: Ensure DataSet.ReadXml\(\)'s input is trusted](#)

[CA2352: Unsafe DataSet or DataTable in serializable type can be vulnerable to remote code execution attacks](#)

[CA2353: Unsafe DataSet or DataTable in serializable type](#)

[CA2355: Unsafe DataSet or DataTable in deserialized object graph](#)

[CA2356: Unsafe DataSet or DataTable in web deserialized object graph](#)

CA2361: Ensure autogenerated class containing DataSet.ReadXml() is not used with untrusted data

CA2362: Unsafe DataSet or DataTable in autogenerated serializable type can be vulnerable to remote code execution attacks

CA2355: Unsafe DataSet or DataTable in deserialized object graph

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA2355
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

Deserializing when the casted or specified type's object graph can include a [DataSet](#) or [DataTable](#).

This rule uses a different approach to a similar rule, [CA2353: Unsafe DataSet or DataTable in serializable type](#).

The casted or specified type is evaluated when:

- Initializing a [DataContractSerializer](#) object
- Initializing a [DataContractJsonSerializer](#) object
- Initializing an [XmlSerializer](#) object
- Invoking [JavaScriptSerializer.Deserialize](#)
- Invoking [JavaScriptSerializer.DeserializeObject](#)
- Invoking [XmlSerializer.FromTypes](#)
- Invoking [Newtonsoft.Json.NET.JsonSerializer.Deserialize](#)
- Invoking [Newtonsoft.Json.NET.JsonConvert.DeserializeObject](#)

Rule description

When deserializing untrusted input with [BinaryFormatter](#) and the deserialized object graph contains a [DataSet](#) or [DataTable](#), an attacker can craft a malicious payload to perform a denial of service attack. There may be unknown remote code execution vulnerabilities.

For more information, see [DataSet and DataTable security guidance](#).

How to fix violations

- If possible, use [Entity Framework](#) rather than [DataSet](#) and [DataTable](#).
- Make the serialized data tamper-proof. After serialization, cryptographically sign the serialized data. Before deserialization, validate the cryptographic signature. Protect the cryptographic key from being disclosed and design for key rotations.

When to suppress warnings

It's safe to suppress a warning from this rule if:

- You know the input is trusted. Consider that your application's trust boundary and data flows may change

over time.

- You've taken one of the precautions in [How to fix violations](#).

SUPPRESS A WARNING

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA2355
// The code that's violating the rule is on this line.
#pragma warning restore CA2355
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA2355.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

PSEUDO-CODE EXAMPLES

Violation

```
using System.Data;
using System.IO;
using System.Runtime.Serialization;

[Serializable]
public class MyClass
{
    public MyOtherClass OtherClass { get; set; }
}

[Serializable]
public class MyOtherClass
{
    private DataSet myDataSet;
}

public class ExampleClass
{
    public MyClass Deserialize(Stream stream)
    {
        BinaryFormatter bf = new BinaryFormatter();
        return (MyClass) bf.Deserialize(stream);
    }
}
```

RELATED RULES

[CA2350: Ensure DataTable.ReadXml\(\)'s input is trusted](#)

CA2351: Ensure DataSet.ReadXml()'s input is trusted

CA2352: Unsafe DataSet or DataTable in serializable type can be vulnerable to remote code execution attacks

CA2353: Unsafe DataSet or DataTable in serializable type

CA2354: Unsafe DataSet or DataTable in deserialized object graph can be vulnerable to remote code execution attack

CA2356: Unsafe DataSet or DataTable in web deserialized object graph

CA2361: Ensure autogenerated class containing DataSet.ReadXml() is not used with untrusted data

CA2362: Unsafe DataSet or DataTable in autogenerated serializable type can be vulnerable to remote code execution attacks

CA2356: Unsafe DataSet or DataTable type in web deserialized object graph

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA2356
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

A method with a [System.Web.Services.WebMethodAttribute](#) or [System.ServiceModel.OperationContractAttribute](#) has a parameter that may reference a [DataSet](#) or [DataTable](#).

This rule uses a different approach to a similar rule, [CA2355: Unsafe DataSet or DataTable in deserialized object graph](#) and will find different warnings.

Rule description

When deserializing untrusted input and the deserialized object graph contains a [DataSet](#) or [DataTable](#), an attacker can craft a malicious payload to perform a denial of service attack. There may be unknown remote code execution vulnerabilities.

For more information, see [DataSet and DataTable security guidance](#).

How to fix violations

- If possible, use [Entity Framework](#) rather than [DataSet](#) and [DataTable](#).
- Make the serialized data tamper-proof. After serialization, cryptographically sign the serialized data. Before deserialization, validate the cryptographic signature. Protect the cryptographic key from being disclosed and design for key rotations.

When to suppress warnings

It's safe to suppress a warning from this rule if:

- You know the input is trusted. Consider that your application's trust boundary and data flows may change over time.
- You've taken one of the precautions in [How to fix violations](#).

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA2356
// The code that's violating the rule is on this line.
#pragma warning restore CA2356
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA2356.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Pseudo-code examples

Violation

```
using System;
using System.Data;
using System.Web.Services;

[WebService(Namespace = "http://contoso.example.com/")]
public class MyService : WebService
{
    [WebMethod]
    public string MyWebMethod(DataTable dataTable)
    {
        return null;
    }
}
```

Related rules

[CA2350: Ensure DataTable.ReadXml\(\)'s input is trusted](#)

[CA2351: Ensure DataSet.ReadXml\(\)'s input is trusted](#)

[CA2352: Unsafe DataSet or DataTable in serializable type can be vulnerable to remote code execution attacks](#)

[CA2353: Unsafe DataSet or DataTable in serializable type](#)

[CA2354: Unsafe DataSet or DataTable in deserialized object graph can be vulnerable to remote code execution attack](#)

[CA2355: Unsafe DataSet or DataTable in deserialized object graph](#)

[CA2361: Ensure DataSet.ReadXml\(\)'s input is trusted](#)

[CA2362: Unsafe DataSet or DataTable in autogenerated serializable type can be vulnerable to remote code execution attacks](#)

CA2361: Ensure autogenerated class containing DataSet.ReadXml() is not used with untrusted data

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA2361
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

The `DataSet.ReadXml` method was called or referenced, and is within autogenerated code.

This rule classifies autogenerated code b:

- Being inside a method named `ReadXmlSerializable`.
- The `ReadXmlSerializable` method has a `System.Diagnostics.DebuggerNonUserCodeAttribute`.
- The `ReadXmlSerializable` method is within a type that has a `System.ComponentModel.DesignerCategoryAttribute`.

[CA2351](#) is a similar rule, for when `DataSet.ReadXml` appears within non-autogenerated code.

Rule description

When deserializing a `DataSet` with untrusted input, an attacker can craft malicious input to perform a denial of service attack. There may be unknown remote code execution vulnerabilities.

This rule is like [CA2351](#), but for autogenerated code for an in-memory representation of data within a GUI application. Usually, these autogenerated classes aren't serialized from untrusted input. Your application's usage may vary.

For more information, see [DataSet and DataTable security guidance](#).

How to fix violations

- If possible, use [Entity Framework](#) rather than the `DataSet`.
- Make the serialized data tamper-proof. After serialization, cryptographically sign the serialized data. Before deserialization, validate the cryptographic signature. Protect the cryptographic key from being disclosed and design for key rotations.

When to suppress warnings

It's safe to suppress a warning from this rule if:

- You know the input is trusted. Consider that your application's trust boundary and data flows may change over time.
- You've taken one of the precautions in [How to fix violations](#).

SUPPRESS A WARNING

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA2361
// The code that's violating the rule is on this line.
#pragma warning restore CA2361
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.cs,vb]
dotnet_diagnostic.CA2361.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.cs,vb]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

PSEUDO-CODE EXAMPLES

Violation

```

namespace ExampleNamespace
{
    /// <summary>
    /// Represents a strongly typed in-memory cache of data.
    /// </summary>
    [global::System.Serializable()]
    [global::System.ComponentModel.DesignerCategoryAttribute("code")]
    [global::System.ComponentModel.ToolboxItem(true)]
    [global::System.Xml.Serialization.XmlSchemaProviderAttribute("GetTypedDataSetSchema")]
    [global::System.Xml.Serialization.XmlRootAttribute("Package")]
    [global::System.ComponentModel.Design.HelpKeywordAttribute("vs.data.DataSet")]
    public partial class Something : global::System.Data.DataSet {

        [global::System.Diagnostics.DebuggerNonUserCodeAttribute()]
        [global::System.CodeDom.Compiler.GeneratedCodeAttribute("System.Data.Design.TypedDataSetGenerator",
        "4.0.0.0")]
        protected override void ReadXmlSerializable(global::System.Xml.XmlReader reader) {
            if ((this.DetermineSchemaSerializationMode(reader) ==
global::System.Data.SchemaSerializationMode.IncludeSchema)) {
                this.Reset();
                global::System.Data.DataSet ds = new global::System.Data.DataSet();
                ds.ReadXml(reader);
                if ((ds.Tables["Something"] != null)) {
                    base.Tables.Add(new SomethingTable(ds.Tables["Something"]));
                }
                this.DataSetName = ds.DataSetName;
                this.Prefix = ds.Prefix;
                this.Namespace = ds.Namespace;
                this.Locale = ds.Locale;
                this.CaseSensitive = ds.CaseSensitive;
                this.EnforceConstraints = ds.EnforceConstraints;
                this.Merge(ds, false, global::System.Data.MissingSchemaAction.Add);
                this.InitVars();
            }
            else {
                this.ReadXml(reader);
                this.InitVars();
            }
        }
    }
}

```

Related rules

[CA2350: Ensure DataTable.ReadXml\(\)'s input is trusted](#)

[CA2351: Ensure DataSet.ReadXml\(\)'s input is trusted](#)

[CA2352: Unsafe DataSet or DataTable in serializable type can be vulnerable to remote code execution attacks](#)

[CA2353: Unsafe DataSet or DataTable in serializable type](#)

[CA2354: Unsafe DataSet or DataTable in deserialized object graph can be vulnerable to remote code execution attack](#)

[CA2355: Unsafe DataSet or DataTable in deserialized object graph](#)

[CA2356: Unsafe DataSet or DataTable in web deserialized object graph](#)

[CA2362: Unsafe DataSet or DataTable in autogenerated serializable type can be vulnerable to remote code execution attacks](#)

CA2362: Unsafe DataSet or DataTable in autogenerated serializable type can be vulnerable to remote code execution attacks

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA2362
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

A class or struct marked with [SerializableAttribute](#) contains a [DataSet](#) or [DataTable](#) field or property, and does have a [DesignerCategoryAttribute](#).

[CA2352](#) is a similar rule, for when there isn't a [DesignerCategoryAttribute](#).

Rule description

When deserializing untrusted input with [BinaryFormatter](#) and the deserialized object graph contains a [DataSet](#) or [DataTable](#), an attacker can craft a malicious payload to perform a remote code execution attack.

This rule is like [CA2352](#), but for autogenerated code for an in-memory representation of data within a GUI application. Usually, these autogenerated classes aren't deserialized from untrusted input. Your application's usage may vary.

This rule finds types which are insecure when serialized. If your code doesn't deserialize the types found, then you don't have a serialization vulnerability.

For more information, see [DataSet and DataTable security guidance](#).

How to fix violations

- If possible, use [Entity Framework](#) rather than [DataSet](#) and [DataTable](#).
- Make the serialized data tamper-proof. After serialization, cryptographically sign the serialized data. Before deserialization, validate the cryptographic signature. Protect the cryptographic key from being disclosed and design for key rotations.

When to suppress warnings

It's safe to suppress a warning from this rule if:

- The type found by this rule is never serialized, either directly or indirectly.
- You know the input is trusted. Consider that your application's trust boundary and data flows may change over time.
- You've taken one of the precautions in [How to fix violations](#).

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA2362
// The code that's violating the rule is on this line.
#pragma warning restore CA2362
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA2362.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Pseudo-code examples

Violation

```
using System.Data;
using System.Xml.Serialization;

namespace ExampleNamespace
{

[global::System.CodeDom.Compiler.GeneratedCode("System.Data.Design.TypedDataSetGenerator", "2.0.0.0")]
[global::System.Serializable()]
[global::System.ComponentModel.DesignerCategoryAttribute("code")]
[global::System.ComponentModel.ToolboxItem(true)]
[global::System.Xml.Serialization.XmlSchemaProviderAttribute("GetTypedDataSetSchema")]
[global::System.Xml.Serialization.XmlRootAttribute("Package")]
[global::System.ComponentModel.Design.HelpKeywordAttribute("vs.data.DataSet")]
public class ExampleClass : global::System.Data.DataSet {
    private DataTable table;
}
}
```

Related rules

[CA2350: Ensure DataTable.ReadXml\(\)'s input is trusted](#)

[CA2351: Ensure DataSet.ReadXml\(\)'s input is trusted](#)

[CA2352: Unsafe DataSet or DataTable in serializable type can be vulnerable to remote code execution attacks](#)

[CA2353: Unsafe DataSet or DataTable in serializable type](#)

[CA2354: Unsafe DataSet or DataTable in deserialized object graph can be vulnerable to remote code execution attack](#)

[CA2355: Unsafe DataSet or DataTable in deserialized object graph](#)

CA2356: Unsafe DataSet or DataTable in web deserialized object graph

CA2362: Unsafe DataSet or DataTable in autogenerated serializable type can be vulnerable to remote code execution attacks

CA3001: Review code for SQL injection vulnerabilities

9/20/2022 • 5 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA3001
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

Potentially untrusted HTTP request input reaches an SQL command's text.

By default, this rule analyzes the entire codebase, but this is [configurable](#).

Rule description

When working with untrusted input and SQL commands, be mindful of SQL injection attacks. An SQL injection attack can execute malicious SQL commands, compromising the security and integrity of your application.

Typical techniques include using a single quotation mark or apostrophe for delimiting literal strings, two dashes for a comment, and a semicolon for the end of a statement. For more information, see [SQL Injection](#).

This rule attempts to find input from HTTP requests reaching an SQL command's text.

NOTE

This rule can't track data across assemblies. For example, if one assembly reads the HTTP request input and then passes it to another assembly that executes the SQL command, this rule won't produce a warning.

NOTE

There is a configurable limit to how deep this rule will analyze data flow across method calls. See [Analyzer Configuration](#) for how to configure the limit in an EditorConfig file.

How to fix violations

Use parameterized SQL commands, or stored procedures, with parameters containing the untrusted input.

When to suppress warnings

It's safe to suppress a warning from this rule if you know that the input is always validated against a known safe set of characters.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA3001
// The code that's violating the rule is on this line.
#pragma warning restore CA3001
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA3001.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

Use the following options to configure which parts of your codebase to run this rule on.

- [Exclude specific symbols](#)
- [Exclude specific types and their derived types](#)

You can configure these options for just this rule, for all rules, or for all rules in this category ([Security](#)). For more information, see [Code quality rule configuration options](#).

Exclude specific symbols

You can exclude specific symbols, such as types and methods, from analysis. For example, to specify that the rule should not run on any code within types named `MyType`, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Symbol name only (includes all symbols with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#). Each symbol name requires a symbol-kind prefix, such as `M:` for methods, `T:` for types, and `N:` for namespaces.
- `.ctor` for constructors and `.cctor` for static constructors.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType</code>	Matches all symbols named <code>MyType</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType1 MyType2</code>	Matches all symbols named either <code>MyType1</code> or <code>MyType2</code> .

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS.MyType.MyMethod(ParamType)</code>	Matches specific method <code>MyMethod</code> with the specified fully qualified signature.
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS1.MyType1.MyMethod1(ParamType) M:NS2.MyType2.MyMethod2(ParamType)</code>	Matches specific methods <code>MyMethod1</code> and <code>MyMethod2</code> with the respective fully qualified signatures.

Exclude specific types and their derived types

You can exclude specific types and their derived types from analysis. For example, to specify that the rule should not run on any methods within types named `MyType` and their derived types, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Type name only (includes all types with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#), with an optional `T:` prefix.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType</code>	Matches all types named <code>MyType</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType1 MyType2</code>	Matches all types named either <code>MyType1</code> or <code>MyType2</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS.MyType</code>	Matches specific type <code>MyType</code> with given fully qualified name and all of its derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS1.MyType1 M:NS2.MyType2</code>	Matches specific types <code>MyType1</code> and <code>MyType2</code> with the respective fully qualified names, and all of their derived types.

Pseudo-code examples

Violation

```

using System;
using System.Data;
using System.Data.SqlClient;

namespace TestNamespace
{
    public partial class WebForm : System.Web.UI.Page
    {
        public static string ConnectionString { get; set; }

        protected void Page_Load(object sender, EventArgs e)
        {
            string name = Request.Form["product_name"];
            using (SqlConnection connection = new SqlConnection(ConnectionString))
            {
                SqlCommand sqlCommand = new SqlCommand()
                {
                    CommandText = "SELECT ProductId FROM Products WHERE ProductName = '" + name + "'",
                    CommandType = CommandType.Text,
                };

                SqlDataReader reader = sqlCommand.ExecuteReader();
            }
        }
    }
}

```

```

Imports System
Imports System.Data
Imports System.Data.SqlClient
Imports System.Linq

Namespace VulnerableWebApp
    Partial Public Class WebForm
        Inherits System.Web.UI.Page

        Public Property ConnectionString As String

        Protected Sub Page_Load(sender As Object, e As EventArgs)
            Dim name As String = Me.Request.Form("product_name")
            Using connection As SqlConnection = New SqlConnection(ConnectionString)
                Dim sqlCommand As SqlCommand = New SqlCommand With {.CommandText = "SELECT ProductId FROM
Products WHERE ProductName = '" + name + "'",
.CommandType = CommandType.Text}
                Dim reader As SqlDataReader = sqlCommand.ExecuteReader()
            End Using
        End Sub
    End Class
End Namespace

```

Parameterized solution

```

using System;
using System.Data;
using System.Data.SqlClient;

namespace TestNamespace
{
    public partial class WebForm : System.Web.UI.Page
    {
        public static string ConnectionString { get; set; }

        protected void Page_Load(object sender, EventArgs e)
        {
            string name = Request.Form["product_name"];
            using (SqlConnection connection = new SqlConnection(ConnectionString))
            {
                SqlCommand sqlCommand = new SqlCommand()
                {
                    CommandText = "SELECT ProductId FROM Products WHERE ProductName = @productName",
                    CommandType = CommandType.Text,
                };

                sqlCommand.Parameters.Add("@productName", SqlDbType.NVarChar, 128).Value = name;

                SqlDataReader reader = sqlCommand.ExecuteReader();
            }
        }
    }
}

```

```

Imports System
Imports System.Data
Imports System.Data.SqlClient
Imports System.Linq

Namespace VulnerableWebApp
    Partial Public Class WebForm
        Inherits System.Web.UI.Page

        Public Property ConnectionString As String

        Protected Sub Page_Load(sender As Object, e As EventArgs)
            Dim name As String = Me.Request.Form("product_name")
            Using connection As SqlConnection = New SqlConnection(ConnectionString)
                Dim sqlCommand As SqlCommand = New SqlCommand With {.CommandText = "SELECT ProductId FROM
Products WHERE ProductName = @productName",
                                                       .CommandType = CommandType.Text}
                sqlCommand.Parameters.Add("@productName", SqlDbType.NVarChar, 128).Value = name
                Dim reader As SqlDataReader = sqlCommand.ExecuteReader()
            End Using
        End Sub
    End Class
End Namespace

```

Stored procedure solution

```

using System;
using System.Data;
using System.Data.SqlClient;

namespace TestNamespace
{
    public partial class WebForm : System.Web.UI.Page
    {
        public static string ConnectionString { get; set; }

        protected void Page_Load(object sender, EventArgs e)
        {
            string name = Request.Form["product_name"];
            using (SqlConnection connection = new SqlConnection(ConnectionString))
            {
                SqlCommand sqlCommand = new SqlCommand()
                {
                    CommandText = "sp_GetProductIdFromName",
                    CommandType = CommandType.StoredProcedure,
                };

                sqlCommand.Parameters.Add("@productName", SqlDbType.NVarChar, 128).Value = name;

                SqlDataReader reader = sqlCommand.ExecuteReader();
            }
        }
    }
}

```

```

Imports System
Imports System.Data
Imports System.Data.SqlClient
Imports System.Linq

Namespace VulnerableWebApp
    Partial Public Class WebForm
        Inherits System.Web.UI.Page

        Public Property ConnectionString As String

        Protected Sub Page_Load(sender As Object, e As EventArgs)
            Dim name As String = Me.Request.Form("product_name")
            Using connection As SqlConnection = New SqlConnection(ConnectionString)
                Dim sqlCommand As SqlCommand = New SqlCommand With {.CommandText =
"sp_GetProductIdFromName",
                                                 .CommandType =
.CommandType.StoredProcedure}
                    sqlCommand.Parameters.Add("@productName", SqlDbType.NVarChar, 128).Value = name
                    Dim reader As SqlDataReader = sqlCommand.ExecuteReader()
                End Using
            End Sub
        End Class
    End Namespace

```

CA3002: Review code for XSS vulnerabilities

9/20/2022 • 4 minutes to read • [Edit Online](#)

	Value
Rule ID	CA3002
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

Potentially untrusted HTTP request input reaches raw HTML output.

By default, this rule analyzes the entire codebase, but this is [configurable](#).

Rule description

When working with untrusted input from web requests, be mindful of cross-site scripting (XSS) attacks. An XSS attack injects untrusted input into raw HTML output, allowing the attacker to execute malicious scripts or maliciously modify content in your web page. A typical technique is putting `<script>` elements with malicious code in input. For more information, see [OWASP's XSS](#).

This rule attempts to find input from HTTP requests reaching raw HTML output.

Note

This rule can't track data across assemblies. For example, if one assembly reads the HTTP request input and then passes it to another assembly that outputs raw HTML, this rule won't produce a warning.

Note

There is a configurable limit to how deep this rule will analyze data flow across method calls. See [Analyzer Configuration](#) for how to configure the limit in an EditorConfig file.

How to fix violations

- Instead of outputting raw HTML, use a method or property that first HTML-encodes its input.
- HTML-encode untrusted data before outputting raw HTML.

When to suppress warnings

It's safe to suppress a warning from this rule if:

- You know that the input is validated against a known safe set of characters not containing HTML.
- You know the data is HTML-encoded in a way not detected by this rule.

NOTE

This rule may report false positives for some methods or properties that HTML-encode their input.

SUPPRESS A WARNING

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA3002
// The code that's violating the rule is on this line.
#pragma warning restore CA3002
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA3002.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

CONFIGURE CODE TO ANALYZE

Use the following options to configure which parts of your codebase to run this rule on.

- [Exclude specific symbols](#)
- [Exclude specific types and their derived types](#)

You can configure these options for just this rule, for all rules, or for all rules in this category ([Security](#)). For more information, see [Code quality rule configuration options](#).

Exclude specific symbols

You can exclude specific symbols, such as types and methods, from analysis. For example, to specify that the rule should not run on any code within types named `MyType`, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Symbol name only (includes all symbols with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#). Each symbol name requires a symbol-kind prefix, such as `M:` for methods, `T:` for types, and `N:` for namespaces.
- `.ctor` for constructors and `.cctor` for static constructors.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType</code>	Matches all symbols named <code>MyType</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType1 MyType2</code>	Matches all symbols named either <code>MyType1</code> or <code>MyType2</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS.MyType.MyMethod(ParamType)</code>	Matches specific method <code>MyMethod</code> with the specified fully qualified signature.
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS1.MyType1.MyMethod1(ParamType) M:NS2.MyType2.MyMethod2(ParamType)</code>	Matches specific methods <code>MyMethod1</code> and <code>MyMethod2</code> with the respective fully qualified signatures.

Exclude specific types and their derived types

You can exclude specific types and their derived types from analysis. For example, to specify that the rule should not run on any methods within types named `MyType` and their derived types, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Type name only (includes all types with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#), with an optional `T:` prefix.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType</code>	Matches all types named <code>MyType</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType1 MyType2</code>	Matches all types named either <code>MyType1</code> or <code>MyType2</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS.MyType</code>	Matches specific type <code>MyType</code> with given fully qualified name and all of its derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS1.MyType1 M:NS2.MyType2</code>	Matches specific types <code>MyType1</code> and <code>MyType2</code> with the respective fully qualified names, and all of their derived types.

Pseudo-code examples

Violation

```

using System;

public partial class WebForm : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        string input = Request.Form["in"];
        Response.Write("<HTML>" + input + "</HTML>");
    }
}

```

```

Imports System

Partial Public Class WebForm
    Inherits System.Web.UI.Page

    Protected Sub Page_Load(sender As Object, e As EventArgs)
        Dim input As String = Me.Request.Form("in")
        Me.Response.Write("<HTML>" + input + "</HTML>")
    End Sub
End Class

```

Solution

```

using System;

public partial class WebForm : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        string input = Request.Form["in"];

        // Example usage of System.Web.HttpServerUtility.HtmlEncode().
        Response.Write("<HTML>" + Server.HtmlEncode(input) + "</HTML>");
    }
}

```

```

Imports System

Partial Public Class WebForm
    Inherits System.Web.UI.Page

    Protected Sub Page_Load(sender As Object, e As EventArgs)
        Dim input As String = Me.Request.Form("in")

        ' Example usage of System.Web.HttpServerUtility.HtmlEncode().
        Me.Response.Write("<HTML>" + Me.Server.HtmlEncode(input) + "</HTML>")
    End Sub
End Class

```

CA3003: Review code for file path injection vulnerabilities

9/20/2022 • 4 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA3003
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

Potentially untrusted HTTP request input reaches the path of a file operation.

By default, this rule analyzes the entire codebase, but this is [configurable](#).

Rule description

When working with untrusted input from web requests, be mindful of using user-controlled input when specifying paths to files. An attacker may be able to read an unintended file, resulting in information disclosure of sensitive data. Or, an attacker may be able to write to an unintended file, resulting in unauthorized modification of sensitive data or compromising the server's security. A common attacker technique is [Path Traversal](#) to access files outside of the intended directory.

This rule attempts to find input from HTTP requests reaching a path in a file operation.

NOTE

This rule can't track data across assemblies. For example, if one assembly reads the HTTP request input and then passes it to another assembly that writes to a file, this rule won't produce a warning.

NOTE

There is a configurable limit to how deep this rule will analyze data flow across method calls. See [Analyzer Configuration](#) for how to configure the limit in an EditorConfig file.

How to fix violations

- If possible, limit file paths based on user input to an explicitly known safe list. For example, if your application only needs to access "red.txt", "green.txt", or "blue.txt", only allow those values.
- Check for untrusted filenames and validate that the name is well formed.
- Use full path names when specifying paths.
- Avoid potentially dangerous constructs such as path environment variables.
- Only accept long filenames and validate long name if user submits short names.
- Restrict end user input to valid characters.

- Reject names where MAX_PATH length is exceeded.
- Handle filenames literally, without interpretation.
- Determine if the filename represents a file or a device.

When to suppress warnings

If you've validated input as described in the previous section, it's okay to suppress this warning.

Configure code to analyze

Use the following options to configure which parts of your codebase to run this rule on.

- [Exclude specific symbols](#)
- [Exclude specific types and their derived types](#)

You can configure these options for just this rule, for all rules, or for all rules in this category ([Security](#)). For more information, see [Code quality rule configuration options](#).

Exclude specific symbols

You can exclude specific symbols, such as types and methods, from analysis. For example, to specify that the rule should not run on any code within types named `MyType`, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Symbol name only (includes all symbols with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#). Each symbol name requires a symbol-kind prefix, such as `M:` for methods, `T:` for types, and `N:` for namespaces.
- `.ctor` for constructors and `.cctor` for static constructors.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType</code>	Matches all symbols named <code>MyType</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType1 MyType2</code>	Matches all symbols named either <code>MyType1</code> or <code>MyType2</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS.MyType.MyMethod(ParamType)</code>	Matches specific method <code>MyMethod</code> with the specified fully qualified signature.
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS1.MyType1.MyMethod1(ParamType) M:NS2.MyType2.MyMethod2(ParamType)</code>	Matches specific methods <code>MyMethod1</code> and <code>MyMethod2</code> with the respective fully qualified signatures.

Exclude specific types and their derived types

You can exclude specific types and their derived types from analysis. For example, to specify that the rule should not run on any methods within types named `MyType` and their derived types, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Type name only (includes all types with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#), with an optional `T:` prefix.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType</code>	Matches all types named <code>MyType</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType1 MyType2</code>	Matches all types named either <code>MyType1</code> or <code>MyType2</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS.MyType</code>	Matches specific type <code>MyType</code> with given fully qualified name and all of its derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS1.MyType1 M:NS2.MyType2</code>	Matches specific types <code>MyType1</code> and <code>MyType2</code> with the respective fully qualified names, and all of their derived types.

Pseudo-code examples

Violation

```
using System;
using System.IO;

public partial class WebForm : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        string userInput = Request.Params["UserInput"];
        // Assume the following directory structure:
        // wwwroot\currentWebDirectory\user1.txt
        // wwwroot\currentWebDirectory\user2.txt
        // wwwroot\secret\allsecrets.txt
        // There is nothing wrong if the user inputs:
        // user1.txt
        // However, if the user input is:
        // ..\secret\allsecrets.txt
        // Then an attacker can now see all the secrets.

        // Avoid this:
        using (File.Open(userInput, FileMode.Open))
        {
            // Read a file with the name supplied by user
            // Input through request's query string and display
            // The content to the webpage.
        }
    }
}
```

```
Imports System
Imports System.IO

Partial Public Class WebForm
    Inherits System.Web.UI.Page

    Protected Sub Page_Load(sender As Object, e As EventArgs)
        Dim userInput As String = Me.Request.Params("UserInput")
        ' Assume the following directory structure:
        '   wwwroot\currentWebDirectory\user1.txt
        '   wwwroot\currentWebDirectory\user2.txt
        '   wwwroot\secret\allsecrets.txt
        ' There is nothing wrong if the user inputs:
        '   user1.txt
        ' However, if the user input is:
        '   ..\secret\allsecrets.txt
        ' Then an attacker can now see all the secrets.

        ' Avoid this:
        Using File.Open(userInput, FileMode.Open)
            ' Read a file with the name supplied by user
            ' Input through request's query string and display
            ' The content to the webpage.
        End Using
    End Sub
End Class
```

CA3004: Review code for information disclosure vulnerabilities

9/20/2022 • 4 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA3004
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

An exception's message, stack trace, or string representation reaches web output.

By default, this rule analyzes the entire codebase, but this is [configurable](#).

Rule description

Disclosing exception information gives attackers insight into the internals of your application, which can help attackers find other vulnerabilities to exploit.

This rule attempts to find an exception message, stack trace, or string representation being output to an HTTP response.

NOTE

This rule can't track data across assemblies. For example, if one assembly catches an exception and then passes it to another assembly that outputs the exception, this rule won't produce a warning.

NOTE

There is a configurable limit to how deep this rule will analyze data flow across method calls. For information about how to configure the limit in an EditorConfig file, see [Analyzer Configuration](#).

How to fix violations

Don't output exception information to HTTP responses. Instead, provide a generic error message. For more information, see [OWASP's Improper Error Handling page](#).

When to suppress warnings

If you know your web output is within your application's trust boundary and never exposed outside, it's okay to suppress this warning. This is rare. Take into consideration that your application's trust boundary and data flows may change over time.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA3004
// The code that's violating the rule is on this line.
#pragma warning restore CA3004
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA3004.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

Use the following options to configure which parts of your codebase to run this rule on.

- [Exclude specific symbols](#)
- [Exclude specific types and their derived types](#)

You can configure these options for just this rule, for all rules, or for all rules in this category ([Security](#)). For more information, see [Code quality rule configuration options](#).

Exclude specific symbols

You can exclude specific symbols, such as types and methods, from analysis. For example, to specify that the rule should not run on any code within types named `MyType`, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Symbol name only (includes all symbols with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#). Each symbol name requires a symbol-kind prefix, such as `M:` for methods, `T:` for types, and `N:` for namespaces.
- `.ctor` for constructors and `.cctor` for static constructors.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType</code>	Matches all symbols named <code>MyType</code> .

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType1 MyType2</code>	Matches all symbols named either <code>MyType1</code> or <code>MyType2</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS.MyType.MyMethod(ParamType)</code>	Matches specific method <code>MyMethod</code> with the specified fully qualified signature.
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS1.MyType1.MyMethod1(ParamType) M:NS2.MyType2.MyMethod2(ParamType)</code>	Matches specific methods <code>MyMethod1</code> and <code>MyMethod2</code> with the respective fully qualified signatures.

Exclude specific types and their derived types

You can exclude specific types and their derived types from analysis. For example, to specify that the rule should not run on any methods within types named `MyType` and their derived types, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Type name only (includes all types with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#), with an optional `T:` prefix.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType</code>	Matches all types named <code>MyType</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType1 MyType2</code>	Matches all types named either <code>MyType1</code> or <code>MyType2</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS.MyType</code>	Matches specific type <code>MyType</code> with given fully qualified name and all of its derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS1.MyType1 M:NS2.MyType2</code>	Matches specific types <code>MyType1</code> and <code>MyType2</code> with the respective fully qualified names, and all of their derived types.

Pseudo-code examples

Violation

```

using System;

public partial class WebForm : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs eventArgs)
    {
        try
        {
            object o = null;
            o.ToString();
        }
        catch (Exception e)
        {
            this.Response.Write(e.ToString());
        }
    }
}

```

```

Imports System

Partial Public Class WebForm
    Inherits System.Web.UI.Page

    Protected Sub Page_Load(sender As Object, eventArgs As EventArgs)
        Try
            Dim o As Object = Nothing
            o.ToString()
        Catch e As Exception
            Me.Response.Write(e.ToString())
        End Try
    End Sub
End Class

```

Solution

```

using System;

public partial class WebForm : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs eventArgs)
    {
        try
        {
            object o = null;
            o.ToString();
        }
        catch (Exception e)
        {
            this.Response.Write("An error occurred. Please try again later.");
        }
    }
}

```

```
Imports System

Partial Public Class WebForm
    Inherits System.Web.UI.Page

    Protected Sub Page_Load(sender As Object, eventArgs As EventArgs)
        Try
            Dim o As Object = Nothing
            o.ToString()
        Catch e As Exception
            Me.Response.Write("An error occurred. Please try again later.")
        End Try
    End Sub
End Class
```

CA3005: Review code for LDAP injection vulnerabilities

9/20/2022 • 4 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA3005
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

Potentially untrusted HTTP request input reaches an LDAP statement.

By default, this rule analyzes the entire codebase, but this is [configurable](#).

Rule description

When working with untrusted input, be mindful of Lightweight Directory Access Protocol (LDAP) injection attacks. An attacker can potentially run malicious LDAP statements against information directories. Applications that use user input to construct dynamic LDAP statements to access directory services are particularly vulnerable.

This rule attempts to find input from HTTP requests reaching an LDAP statement.

NOTE

This rule can't track data across assemblies. For example, if one assembly reads the HTTP request input and then passes it to another assembly that executes an LDAP statement, this rule won't produce a warning.

NOTE

There is a configurable limit to how deep this rule will analyze data flow across method calls. See [Analyzer Configuration](#) for how to configure the limit in an EditorConfig file.

How to fix violations

For the user-controlled portion of LDAP statements, consider one of:

- Allow only a safe list of non-special characters.
- Disallow special character
- Escape special characters.

See [OWASP's LDAP Injection Prevention Cheat Sheet](#) for more guidance.

When to suppress warnings

If you know the input has been validated or escaped to be safe, it's okay to suppress this warning.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA3005
// The code that's violating the rule is on this line.
#pragma warning restore CA3005
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA3005.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

Use the following options to configure which parts of your codebase to run this rule on.

- [Exclude specific symbols](#)
- [Exclude specific types and their derived types](#)

You can configure these options for just this rule, for all rules, or for all rules in this category ([Security](#)). For more information, see [Code quality rule configuration options](#).

Exclude specific symbols

You can exclude specific symbols, such as types and methods, from analysis. For example, to specify that the rule should not run on any code within types named `MyType`, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXX.excluded_symbol_names = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Symbol name only (includes all symbols with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#). Each symbol name requires a symbol-kind prefix, such as `M:` for methods, `T:` for types, and `N:` for namespaces.
- `.ctor` for constructors and `.cctor` for static constructors.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXX.excluded_symbol_names = MyType</code>	Matches all symbols named <code>MyType</code> .

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType1 MyType2</code>	Matches all symbols named either <code>MyType1</code> or <code>MyType2</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS.MyType.MyMethod(ParamType)</code>	Matches specific method <code>MyMethod</code> with the specified fully qualified signature.
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS1.MyType1.MyMethod1(ParamType) M:NS2.MyType2.MyMethod2(ParamType)</code>	Matches specific methods <code>MyMethod1</code> and <code>MyMethod2</code> with the respective fully qualified signatures.

Exclude specific types and their derived types

You can exclude specific types and their derived types from analysis. For example, to specify that the rule should not run on any methods within types named `MyType` and their derived types, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Type name only (includes all types with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#), with an optional `T:` prefix.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType</code>	Matches all types named <code>MyType</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType1 MyType2</code>	Matches all types named either <code>MyType1</code> or <code>MyType2</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS.MyType</code>	Matches specific type <code>MyType</code> with given fully qualified name and all of its derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS1.MyType1 M:NS2.MyType2</code>	Matches specific types <code>MyType1</code> and <code>MyType2</code> with the respective fully qualified names, and all of their derived types.

Pseudo-code examples

Violation

```

using System;
using System.DirectoryServices;

public partial class WebForm : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        string userName = Request.Params["user"];
        string filter = "(uid=" + userName + ")"; // searching for the user entry

        // In this example, if we send the * character in the user parameter which will
        // result in the filter variable in the code to be initialized with (uid=*).
        // The resulting LDAP statement will make the server return any object that
        // contains a uid attribute.
        DirectorySearcher searcher = new DirectorySearcher(filter);
        SearchResultCollection results = searcher.FindAll();

        // Iterate through each SearchResult in the SearchResultCollection.
        foreach (SearchResult searchResult in results)
        {
            // ...
        }
    }
}

```

```

Imports System
Imports System.DirectoryServices

Partial Public Class WebForm
    Inherits System.Web.UI.Page

    Protected Sub Page_Load(send As Object, e As EventArgs)
        Dim userName As String = Me.Request.Params("user")
        Dim filter As String = """(uid="" + userName + """)""      ' searching for the user entry

        ' In this example, if we send the * character in the user parameter which will
        ' result in the filter variable in the code to be initialized with (uid=*).
        ' The resulting LDAP statement will make the server return any object that
        ' contains a uid attribute.
        Dim searcher As DirectorySearcher = new DirectorySearcher(filter)
        Dim results As SearchResultCollection = searcher.FindAll()

        ' Iterate through each SearchResult in the SearchResultCollection.
        For Each searchResult As SearchResult in results
            ' ...
            Next searchResult
        End Sub
    End Class

```

CA3006: Review code for process command injection vulnerabilities

9/20/2022 • 3 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA3006
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

Potentially untrusted HTTP request input reaches a process command.

By default, this rule analyzes the entire codebase, but this is [configurable](#).

Rule description

When working with untrusted input, be mindful of command injection attacks. A command injection attack can execute malicious commands on the underlying operating system, compromising the security and integrity of your server.

This rule attempts to find input from HTTP requests reaching a process command.

NOTE

This rule can't track data across assemblies. For example, if one assembly reads the HTTP request input and then passes it to another assembly that starts a process, this rule won't produce a warning.

NOTE

There is a configurable limit to how deep this rule will analyze data flow across method calls. See [Analyzer Configuration](#) for how to configure the limit in an EditorConfig file.

How to fix violations

- If possible, avoid starting processes based on user input.
- Validate input against a known safe set of characters and length.

When to suppress warnings

If you know the input has been validated or escaped to be safe, it's safe to suppress this warning.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then

re-enable the rule.

```
#pragma warning disable CA3006
// The code that's violating the rule is on this line.
#pragma warning restore CA3006
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA3006.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

Use the following options to configure which parts of your codebase to run this rule on.

- [Exclude specific symbols](#)
- [Exclude specific types and their derived types](#)

You can configure these options for just this rule, for all rules, or for all rules in this category ([Security](#)). For more information, see [Code quality rule configuration options](#).

Exclude specific symbols

You can exclude specific symbols, such as types and methods, from analysis. For example, to specify that the rule should not run on any code within types named `MyType`, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Symbol name only (includes all symbols with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#). Each symbol name requires a symbol-kind prefix, such as `M:` for methods, `T:` for types, and `N:` for namespaces.
- `.ctor` for constructors and `.cctor` for static constructors.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType</code>	Matches all symbols named <code>MyType</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType1 MyType2</code>	Matches all symbols named either <code>MyType1</code> or <code>MyType2</code> .

OPTION VALUE	SUMMARY
dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS.MyType.MyMethod(ParamType)	Matches specific method <code>MyMethod</code> with the specified fully qualified signature.
dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS1.MyType1.MyMethod1(ParamType) M:NS2.MyType2.MyMethod2(ParamType)	Matches specific methods <code>MyMethod1</code> and <code>MyMethod2</code> with the respective fully qualified signatures.

Exclude specific types and their derived types

You can exclude specific types and their derived types from analysis. For example, to specify that the rule should not run on any methods within types named `MyType` and their derived types, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Type name only (includes all types with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#), with an optional `T:` prefix.

Examples:

OPTION VALUE	SUMMARY
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType	Matches all types named <code>MyType</code> and all of their derived types.
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType1 MyType2	Matches all types named either <code>MyType1</code> or <code>MyType2</code> and all of their derived types.
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS.MyType	Matches specific type <code>MyType</code> with given fully qualified name and all of its derived types.
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS1.MyType1 M:NS2.MyType2	Matches specific types <code>MyType1</code> and <code>MyType2</code> with the respective fully qualified names, and all of their derived types.

Pseudo-code examples

Violation

```
using System;
using System.Diagnostics;

public partial class WebForm : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        string input = Request.Form["in"];
        Process p = Process.Start(input);
    }
}
```

```
Imports System
Imports System.Diagnostics

Partial Public Class WebForm
    Inherits System.Web.UI.Page

    Protected Sub Page_Load(sender As Object, eventArgs as EventArgs)
        Dim input As String = Me.Request.Form("in")
        Dim p As Process = Process.Start(input)
    End Sub
End Class
```

CA3007: Review code for open redirect vulnerabilities

9/20/2022 • 4 minutes to read • [Edit Online](#)

	Value
Rule ID	CA3007
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

Potentially untrusted HTTP request input reaches an HTTP response redirect.

By default, this rule analyzes the entire codebase, but this is [configurable](#).

Rule description

When working with untrusted input, be mindful of open redirect vulnerabilities. An attacker can exploit an open redirect vulnerability to use your website to give the appearance of a legitimate URL, but redirect an unsuspecting visitor to a phishing or other malicious webpage.

This rule attempts to find input from HTTP requests reaching an HTTP redirect URL.

Note

This rule can't track data across assemblies. For example, if one assembly reads the HTTP request input and then passes it to another assembly that responds with an HTTP redirect, this rule won't produce a warning.

Note

There is a configurable limit to how deep this rule will analyze data flow across method calls. See [Analyzer Configuration](#) for how to configure the limit in an EditorConfig file.

How to fix violations

Some approaches to fixing open redirect vulnerabilities include:

- Don't allow users to initiate redirects.
- Don't allow users to specify any part of the URL in a redirect scenario.
- Restrict redirects to a predefined "allow list" of URLs.
- Validate redirect URLs.
- If applicable, consider using a disclaimer page when users are being redirected away from your site.

When to suppress warnings

If you know you've validated the input to be restricted to intended URLs, it's okay to suppress this warning.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA3007
// The code that's violating the rule is on this line.
#pragma warning restore CA3007
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA3007.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

Use the following options to configure which parts of your codebase to run this rule on.

- [Exclude specific symbols](#)
- [Exclude specific types and their derived types](#)

You can configure these options for just this rule, for all rules, or for all rules in this category ([Security](#)). For more information, see [Code quality rule configuration options](#).

Exclude specific symbols

You can exclude specific symbols, such as types and methods, from analysis. For example, to specify that the rule should not run on any code within types named `MyType`, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Symbol name only (includes all symbols with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#). Each symbol name requires a symbol-kind prefix, such as `M:` for methods, `T:` for types, and `N:` for namespaces.
- `.ctor` for constructors and `.cctor` for static constructors.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType</code>	Matches all symbols named <code>MyType</code> .

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType1 MyType2</code>	Matches all symbols named either <code>MyType1</code> or <code>MyType2</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS.MyType.MyMethod(ParamType)</code>	Matches specific method <code>MyMethod</code> with the specified fully qualified signature.
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS1.MyType1.MyMethod1(ParamType) M:NS2.MyType2.MyMethod2(ParamType)</code>	Matches specific methods <code>MyMethod1</code> and <code>MyMethod2</code> with the respective fully qualified signatures.

Exclude specific types and their derived types

You can exclude specific types and their derived types from analysis. For example, to specify that the rule should not run on any methods within types named `MyType` and their derived types, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Type name only (includes all types with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#), with an optional `T:` prefix.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType</code>	Matches all types named <code>MyType</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType1 MyType2</code>	Matches all types named either <code>MyType1</code> or <code>MyType2</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS.MyType</code>	Matches specific type <code>MyType</code> with given fully qualified name and all of its derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS1.MyType1 M:NS2.MyType2</code>	Matches specific types <code>MyType1</code> and <code>MyType2</code> with the respective fully qualified names, and all of their derived types.

Pseudo-code examples

Violation

```
using System;

public partial class WebForm : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        string input = Request.Form["url"];
        this.Response.Redirect(input);
    }
}
```

```
Imports System

Partial Public Class WebForm
    Inherits System.Web.UI.Page

    Protected Sub Page_Load(sender As Object, eventArgs As EventArgs)
        Dim input As String = Me.Request.Form("url")
        Me.Response.Redirect(input)
    End Sub
End Class
```

Solution

```
using System;

public partial class WebForm : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        string input = Request.Form["in"];
        if (String.IsNullOrEmpty(input))
        {
            this.Response.Redirect("https://example.org/login.html");
        }
    }
}
```

```
Imports System

Partial Public Class WebForm
    Inherits System.Web.UI.Page

    Protected Sub Page_Load(sender As Object, eventArgs As EventArgs)
        Dim input As String = Me.Request.Form("in")
        If String.IsNullOrEmpty(input) Then
            Me.Response.Redirect("https://example.org/login.html")
        End If
    End Sub
End Class
```

CA3008: Review code for XPath injection vulnerabilities

9/20/2022 • 4 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA3008
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

Potentially untrusted HTTP request input reaches an XPath query.

By default, this rule analyzes the entire codebase, but this is [configurable](#).

Rule description

When working with untrusted input, be mindful of XPath injection attacks. Constructing XPath queries using untrusted input may allow an attacker to maliciously manipulate the query to return an unintended result, and possibly disclose the contents of the queried XML.

This rule attempts to find input from HTTP requests reaching an XPath expression.

NOTE

This rule can't track data across assemblies. For example, if one assembly reads the HTTP request input and then passes it to another assembly that performs an XPath query, this rule won't produce a warning.

NOTE

There is a configurable limit to how deep this rule will analyze data flow across method calls. See [Analyzer Configuration](#) for how to configure the limit in an EditorConfig file.

How to fix violations

Some approaches to fixing XPath injection vulnerabilities include:

- Don't construct XPath queries from user input.
- Validate that the input only contains a safe set of characters.
- Escape quotation marks.

When to suppress warnings

If you know you've validated the input to be safe, it's okay to suppress this warning.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA3008
// The code that's violating the rule is on this line.
#pragma warning restore CA3008
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.cs,vb]
dotnet_diagnostic.CA3008.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.cs,vb]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

Use the following options to configure which parts of your codebase to run this rule on.

- [Exclude specific symbols](#)
- [Exclude specific types and their derived types](#)

You can configure these options for just this rule, for all rules, or for all rules in this category ([Security](#)). For more information, see [Code quality rule configuration options](#).

Exclude specific symbols

You can exclude specific symbols, such as types and methods, from analysis. For example, to specify that the rule should not run on any code within types named `MyType`, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Symbol name only (includes all symbols with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#). Each symbol name requires a symbol-kind prefix, such as `M:` for methods, `T:` for types, and `N:` for namespaces.
- `.ctor` for constructors and `.cctor` for static constructors.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType</code>	Matches all symbols named <code>MyType</code> .

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType1 MyType2</code>	Matches all symbols named either <code>MyType1</code> or <code>MyType2</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS.MyType.MyMethod(ParamType)</code>	Matches specific method <code>MyMethod</code> with the specified fully qualified signature.
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS1.MyType1.MyMethod1(ParamType) M:NS2.MyType2.MyMethod2(ParamType)</code>	Matches specific methods <code>MyMethod1</code> and <code>MyMethod2</code> with the respective fully qualified signatures.

Exclude specific types and their derived types

You can exclude specific types and their derived types from analysis. For example, to specify that the rule should not run on any methods within types named `MyType` and their derived types, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Type name only (includes all types with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#), with an optional `T:` prefix.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType</code>	Matches all types named <code>MyType</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType1 MyType2</code>	Matches all types named either <code>MyType1</code> or <code>MyType2</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS.MyType</code>	Matches specific type <code>MyType</code> with given fully qualified name and all of its derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS1.MyType1 M:NS2.MyType2</code>	Matches specific types <code>MyType1</code> and <code>MyType2</code> with the respective fully qualified names, and all of their derived types.

Pseudo-code examples

Violation

```

using System;
using System.Xml.XPath;

public partial class WebForm : System.Web.UI.Page
{
    public XPathNavigator AuthorizedOperations { get; set; }

    protected void Page_Load(object sender, EventArgs e)
    {
        string operation = Request.Form["operation"];

        // If an attacker uses this for input:
        //     ' or 'a' = 'a
        // Then the XPath query will be:
        //     authorizedOperation[@username = 'anonymous' and @operationName = '' or 'a' = 'a']
        // and it will return any authorizedOperation node.
        XPathNavigator node = AuthorizedOperations.SelectSingleNode(
            "//authorizedOperation[@username = 'anonymous' and @operationName = '" + operation + "']");
    }
}

```

```

Imports System
Imports System.Xml.XPath

Partial Public Class WebForm
    Inherits System.Web.UI.Page

    Public Property AuthorizedOperations As XPathNavigator

    Protected Sub Page_Load(sender As Object, e As EventArgs)
        Dim operation As String = Me.Request.Form("operation")

        ' If an attacker uses this for input:
        '     ' or 'a' = 'a
        ' Then the XPath query will be:
        '     authorizedOperation[@username = 'anonymous' and @operationName = '' or 'a' = 'a']
        ' and it will return any authorizedOperation node.
        Dim node As XPathNavigator = AuthorizedOperations.SelectSingleNode(
            "//authorizedOperation[@username = 'anonymous' and @operationName = '" + operation + "']")
    End Sub
End Class

```

CA3009: Review code for XML injection vulnerabilities

9/20/2022 • 4 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA3009
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

Potentially untrusted HTTP request input reaches raw XML output.

By default, this rule analyzes the entire codebase, but this is [configurable](#).

Rule description

When working with untrusted input, be mindful of XML injection attacks. An attacker can use XML injection to insert special characters into an XML document, making the document invalid XML. Or, an attacker could maliciously insert XML nodes of their choosing.

This rule attempts to find input from HTTP requests reaching a raw XML write.

NOTE

This rule can't track data across assemblies. For example, if one assembly reads the HTTP request input and then passes it to another assembly that writes raw XML, this rule won't produce a warning.

NOTE

There is a configurable limit to how deep this rule will analyze data flow across method calls. See [Analyzer Configuration](#) for how to configure the limit in an EditorConfig file.

How to fix violations

Don't write raw XML. Instead, use methods or properties that XML-encode their input.

Or, XML-encode input before writing raw XML.

Or, validate user input by using sanitizers for primitive type conversion and XML encoding.

When to suppress warnings

Don't suppress warnings from this rule.

Configure code to analyze

Use the following options to configure which parts of your codebase to run this rule on.

- [Exclude specific symbols](#)
- [Exclude specific types and their derived types](#)

You can configure these options for just this rule, for all rules, or for all rules in this category ([Security](#)). For more information, see [Code quality rule configuration options](#).

Exclude specific symbols

You can exclude specific symbols, such as types and methods, from analysis. For example, to specify that the rule should not run on any code within types named `MyType`, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Symbol name only (includes all symbols with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#). Each symbol name requires a symbol-kind prefix, such as `M:` for methods, `T:` for types, and `N:` for namespaces.
- `.ctor` for constructors and `.cctor` for static constructors.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType</code>	Matches all symbols named <code>MyType</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType1 MyType2</code>	Matches all symbols named either <code>MyType1</code> or <code>MyType2</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS.MyType.MyMethod(ParamType)</code>	Matches specific method <code>MyMethod</code> with the specified fully qualified signature.
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS1.MyType1.MyMethod1(ParamType) M:NS2.MyType2.MyMethod2(ParamType)</code>	Matches specific methods <code>MyMethod1</code> and <code>MyMethod2</code> with the respective fully qualified signatures.

Exclude specific types and their derived types

You can exclude specific types and their derived types from analysis. For example, to specify that the rule should not run on any methods within types named `MyType` and their derived types, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Type name only (includes all types with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#), with an optional `T:` prefix.

Examples:

OPTION VALUE	SUMMARY
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType	Matches all types named <code>MyType</code> and all of their derived types.
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType1 MyType2	Matches all types named either <code>MyType1</code> or <code>MyType2</code> and all of their derived types.
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS.MyType	Matches specific type <code>MyType</code> with given fully qualified name and all of its derived types.
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS1.MyType1 M:NS2.MyType2	Matches specific types <code>MyType1</code> and <code>MyType2</code> with the respective fully qualified names, and all of their derived types.

Pseudo-code examples

Violation

```

using System;
using System.Xml;

public partial class WebForm : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        string input = Request.Form["in"];
        XmlDocument d = new XmlDocument();
        XmlElement root = d.CreateElement("root");
        d.AppendChild(root);

        XmlElement allowedUser = d.CreateElement("allowedUser");
        root.AppendChild(allowedUser);

        allowedUser.InnerXml = "alice";

        // If an attacker uses this for input:
        //     some text<allowedUser>oscar</allowedUser>
        // Then the XML document will be:
        //     <root>some text<allowedUser>oscar</allowedUser></root>
        root.InnerXml = input;
    }
}

```

```

Imports System
Imports System.Xml

Public Partial Class WebForm
    Inherits System.Web.UI.Page

    Sub Page_Load(sender As Object, e As EventArgs)
        Dim input As String = Request.Form("in")
        Dim d As XmlDocument = New XmlDocument()
        Dim root As XmlElement = d.CreateElement("root")
        d.AppendChild(root)

        Dim allowedUser As XmlElement = d.CreateElement("allowedUser")
        root.AppendChild(allowedUser)

        allowedUser.InnerXml = "alice"

        ' If an attacker uses this for input:
        '     some text<allowedUser>oscar</allowedUser>
        ' Then the XML document will be:
        '     <root>some text<allowedUser>oscar</allowedUser></root>
        root.InnerXml = input
    End Sub
End Class

```

Solution

```

using System;
using System.Xml;

public partial class WebForm : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        string input = Request.Form["in"];
        XmlDocument d = new XmlDocument();
        XmlElement root = d.CreateElement("root");
        d.AppendChild(root);

        XmlElement allowedUser = d.CreateElement("allowedUser");
        root.AppendChild(allowedUser);

        allowedUser.InnerText = "alice";

        // If an attacker uses this for input:
        //     some text<allowedUser>oscar</allowedUser>
        // Then the XML document will be:
        //     <root>&lt;allowedUser&gt;oscar&lt;/allowedUser&gt;some text<allowedUser>alice</allowedUser>
    }
}

```

```
Imports System
Imports System.Xml

Public Partial Class WebForm
    Inherits System.Web.UI.Page

    Sub Page_Load(sender As Object, e As EventArgs)
        Dim input As String = Request.Form("in")
        Dim d As XmlDocument = New XmlDocument()
        Dim root As XmlElement = d.CreateElement("root")
        d.AppendChild(root)

        Dim allowedUser As XmlElement = d.CreateElement("allowedUser")
        root.AppendChild(allowedUser)

        allowedUser.InnerText = "alice"

        ' If an attacker uses this for input:
        '     some text<allowedUser>oscar</allowedUser>
        ' Then the XML document will be:
        '     <root>&lt;allowedUser&gt;oscar&lt;/allowedUser&gt;some text<allowedUser>alice</allowedUser>
    End Sub
End Class
```

CA3010: Review code for XAML injection vulnerabilities

9/20/2022 • 3 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA3010
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

Potentially untrusted HTTP request input reaches a [System.Windows.Markup.XamlReader Load](#) method.

By default, this rule analyzes the entire codebase, but this is [configurable](#).

Rule description

When working with untrusted input, be mindful of XAML injection attacks. XAML is a markup language that directly represents object instantiation and execution. That means elements created in XAML can interact with system resources (for example, network access and file system IO). If an attacker can control the input to a [System.Windows.Markup.XamlReader Load](#) method call, then the attacker can execute code.

This rule attempts to find input from HTTP requests that reaches a [System.Windows.Markup.XamlReader Load](#) method.

NOTE

This rule can't track data across assemblies. For example, if one assembly reads the HTTP request input and then passes it to another assembly that loads XAML, this rule won't produce a warning.

NOTE

There is a configurable limit to how deep this rule will analyze data flow across method calls. See [Analyzer Configuration](#) for how to configure the limit in an EditorConfig file.

How to fix violations

Don't load untrusted XAML.

When to suppress warnings

Don't suppress warnings from this rule.

Configure code to analyze

Use the following options to configure which parts of your codebase to run this rule on.

- [Exclude specific symbols](#)
- [Exclude specific types and their derived types](#)

You can configure these options for just this rule, for all rules, or for all rules in this category ([Security](#)). For more information, see [Code quality rule configuration options](#).

Exclude specific symbols

You can exclude specific symbols, such as types and methods, from analysis. For example, to specify that the rule should not run on any code within types named `MyType`, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Symbol name only (includes all symbols with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#). Each symbol name requires a symbol-kind prefix, such as `M:` for methods, `T:` for types, and `N:` for namespaces.
- `.ctor` for constructors and `.cctor` for static constructors.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType</code>	Matches all symbols named <code>MyType</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType1 MyType2</code>	Matches all symbols named either <code>MyType1</code> or <code>MyType2</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS.MyType.MyMethod(ParamType)</code>	Matches specific method <code>MyMethod</code> with the specified fully qualified signature.
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS1.MyType1.MyMethod1(ParamType) M:NS2.MyType2.MyMethod2(ParamType)</code>	Matches specific methods <code>MyMethod1</code> and <code>MyMethod2</code> with the respective fully qualified signatures.

Exclude specific types and their derived types

You can exclude specific types and their derived types from analysis. For example, to specify that the rule should not run on any methods within types named `MyType` and their derived types, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Type name only (includes all types with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#), with an optional `T:` prefix.

Examples:

OPTION VALUE	SUMMARY
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType	Matches all types named <code>MyType</code> and all of their derived types.
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType1 MyType2	Matches all types named either <code>MyType1</code> or <code>MyType2</code> and all of their derived types.
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS.MyType	Matches specific type <code>MyType</code> with given fully qualified name and all of its derived types.
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS1.MyType1 M:NS2.MyType2	Matches specific types <code>MyType1</code> and <code>MyType2</code> with the respective fully qualified names, and all of their derived types.

Pseudo-code examples

Violation

```
using System;
using System.IO;

public partial class WebForm : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        string input = Request.Form["in"];
        byte[] bytes = Convert.FromBase64String(input);
        MemoryStream ms = new MemoryStream(bytes);
        System.Windows.Markup.XamlReader.Load(ms);
    }
}
```

```
Imports System
Imports System.IO

Public Partial Class WebForm
    Inherits System.Web.UI.Page

    Protected Sub Page_Load(sender As Object, e As EventArgs)
        Dim input As String = Request.Form("in")
        Dim bytes As Byte() = Convert.FromBase64String(input)
        Dim ms As MemoryStream = New MemoryStream(bytes)
        System.Windows.Markup.XamlReader.Load(ms)
    End Sub
End Class
```

CA3011: Review code for DLL injection vulnerabilities

9/20/2022 • 3 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA3011
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

Potentially untrusted HTTP request input reaches a method that loads an assembly.

By default, this rule analyzes the entire codebase, but this is [configurable](#).

Rule description

When working with untrusted input, be mindful of loading untrusted code. If your web application loads untrusted code, an attacker may be able to inject malicious DLLs into your process and execute malicious code.

This rule attempts to find input from an HTTP request that reaches a method that loads an assembly.

NOTE

This rule can't track data across assemblies. For example, if one assembly reads the HTTP request input and then passes it to another assembly that loads an assembly, this rule won't produce a warning.

NOTE

There is a configurable limit to how deep this rule will analyze data flow across method calls. See [Analyzer Configuration](#) for how to configure the limit in an EditorConfig file.

How to fix violations

Don't load untrusted DLLs from user input.

When to suppress warnings

Don't suppress warnings from this rule.

Configure code to analyze

Use the following options to configure which parts of your codebase to run this rule on.

- [Exclude specific symbols](#)

- Exclude specific types and their derived types

You can configure these options for just this rule, for all rules, or for all rules in this category ([Security](#)). For more information, see [Code quality rule configuration options](#).

Exclude specific symbols

You can exclude specific symbols, such as types and methods, from analysis. For example, to specify that the rule should not run on any code within types named `MyType`, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Symbol name only (includes all symbols with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#). Each symbol name requires a symbol-kind prefix, such as `M:` for methods, `T:` for types, and `N:` for namespaces.
- `.ctor` for constructors and `.cctor` for static constructors.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType</code>	Matches all symbols named <code>MyType</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType1 MyType2</code>	Matches all symbols named either <code>MyType1</code> or <code>MyType2</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS.MyType.MyMethod(ParamType)</code>	Matches specific method <code>MyMethod</code> with the specified fully qualified signature.
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS1.MyType1.MyMethod1(ParamType) M:NS2.MyType2.MyMethod2(ParamType)</code>	Matches specific methods <code>MyMethod1</code> and <code>MyMethod2</code> with the respective fully qualified signatures.

Exclude specific types and their derived types

You can exclude specific types and their derived types from analysis. For example, to specify that the rule should not run on any methods within types named `MyType` and their derived types, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Type name only (includes all types with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#), with an optional `T:` prefix.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType</code>	Matches all types named <code>MyType</code> and all of their derived types.

OPTION VALUE	SUMMARY
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType1 MyType2	Matches all types named either <code>MyType1</code> or <code>MyType2</code> and all of their derived types.
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS.MyType	Matches specific type <code>MyType</code> with given fully qualified name and all of its derived types.
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS1.MyType1 M:NS2.MyType2	Matches specific types <code>MyType1</code> and <code>MyType2</code> with the respective fully qualified names, and all of their derived types.

Pseudo-code examples

Violation

```
using System;
using System.Reflection;

public partial class WebForm : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        string input = Request.Form["in"];
        byte[] rawAssembly = Convert.FromBase64String(input);
        Assembly.Load(rawAssembly);
    }
}
```

```
Imports System
Imports System.Reflection

Public Partial Class WebForm
    Inherits System.Web.UI.Page

    Protected Sub Page_Load(sender As Object, e As EventArgs)
        Dim input As String = Request.Form("in")
        Dim rawAssembly As Byte() = Convert.FromBase64String(input)
        Assembly.Load(rawAssembly)
    End Sub
End Class
```

CA3012: Review code for regex injection vulnerabilities

9/20/2022 • 4 minutes to read • [Edit Online](#)

	Value
Rule ID	CA3012
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

Potentially untrusted HTTP request input reaches a regular expression.

By default, this rule analyzes the entire codebase, but this is [configurable](#).

Rule description

When working with untrusted input, be mindful of regex injection attacks. An attacker can use regex injection to maliciously modify a regular expression, to make the regex match unintended results, or to make the regex consume excessive CPU resulting in a Denial of Service attack.

This rule attempts to find input from HTTP requests reaching a regular expression.

Note

This rule can't track data across assemblies. For example, if one assembly reads the HTTP request input and then passes it to another assembly that creates a regular expression, this rule won't produce a warning.

Note

There is a configurable limit to how deep this rule will analyze data flow across method calls. See [Analyzer Configuration](#) for how to configure the limit in an EditorConfig file.

How to fix violations

Some mitigations against regex injections include:

- Always use a [match timeout](#) when using regular expressions.
- Avoid using regular expressions based on user input.
- Escape special characters from user input by calling [System.Text.RegularExpressions.Regex.Escape](#) or another method.
- Allow only non-special characters from user input.

When to suppress warnings

If you know you're using a [match timeout](#) and the user input is free of special characters, it's okay to suppress this warning.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA3012
// The code that's violating the rule is on this line.
#pragma warning restore CA3012
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA3012.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

Use the following options to configure which parts of your codebase to run this rule on.

- [Exclude specific symbols](#)
- [Exclude specific types and their derived types](#)

You can configure these options for just this rule, for all rules, or for all rules in this category ([Security](#)). For more information, see [Code quality rule configuration options](#).

Exclude specific symbols

You can exclude specific symbols, such as types and methods, from analysis. For example, to specify that the rule should not run on any code within types named `MyType`, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Symbol name only (includes all symbols with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#). Each symbol name requires a symbol-kind prefix, such as `M:` for methods, `T:` for types, and `N:` for namespaces.
- `.ctor` for constructors and `.cctor` for static constructors.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType</code>	Matches all symbols named <code>MyType</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType1 MyType2</code>	Matches all symbols named either <code>MyType1</code> or <code>MyType2</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS.MyType.MyMethod(ParamType)</code>	Matches specific method <code>MyMethod</code> with the specified fully qualified signature.
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS1.MyType1.MyMethod1(ParamType) M:NS2.MyType2.MyMethod2(ParamType)</code>	Matches specific methods <code>MyMethod1</code> and <code>MyMethod2</code> with the respective fully qualified signatures.

Exclude specific types and their derived types

You can exclude specific types and their derived types from analysis. For example, to specify that the rule should not run on any methods within types named `MyType` and their derived types, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Type name only (includes all types with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#), with an optional `T:` prefix.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType</code>	Matches all types named <code>MyType</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType1 MyType2</code>	Matches all types named either <code>MyType1</code> or <code>MyType2</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS.MyType</code>	Matches specific type <code>MyType</code> with given fully qualified name and all of its derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS1.MyType1 M:NS2.MyType2</code>	Matches specific types <code>MyType1</code> and <code>MyType2</code> with the respective fully qualified names, and all of their derived types.

Pseudo-code examples

Violation

```
using System;
using System.Text.RegularExpressions;

public partial class WebForm : System.Web.UI.Page
{
    public string SearchableText { get; set; }

    protected void Page_Load(object sender, EventArgs e)
    {
        string findTerm = Request.Form["findTerm"];
        Match m = Regex.Match(SearchableText, "^term=" + findTerm);
    }
}
```

```
Imports System
Imports System.Text.RegularExpressions

Public Partial Class WebForm
    Inherits System.Web.UI.Page

    Public Property SearchableText As String

    Protected Sub Page_Load(sender As Object, e As EventArgs)
        Dim findTerm As String = Request.Form("findTerm")
        Dim m As Match = Regex.Match(SearchableText, "^term=" + findTerm)
    End Sub
End Class
```

CA3061: Do not add schema by URL

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA3061
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

Overload of `XmlSchemaCollection.Add(String, String)` is using `XmlUrlResolver` to specify external XML schema in the form of an URI. If the URI String is tainted, it may lead to parsing of a malicious XML schema, which allows for the inclusion of XML bombs and malicious external entities. This could allow a malicious attacker to perform a denial of service, information disclosure, or server-side request forgery attack.

Rule description

Do not use the unsafe overload of the `Add` method because it may cause dangerous external references.

How to fix violations

- Do not use `XmlSchemaCollection.Add(String, String)`.

When to suppress warnings

Suppress this rule if you are sure your XML does not resolve dangerous external references.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA3061
// The code that's violating the rule is on this line.
#pragma warning restore CA3061
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA3061.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Pseudo-code examples

Violation

The following pseudo-code sample illustrates the pattern detected by this rule. The second parameter's type is `string`.

```
using System;
using System.Xml.Schema;
...
XmlSchemaCollection xsc = new XmlSchemaCollection();
xsc.Add("urn: bookstore - schema", "books.xsd");
```

Solution

```
using System;
using System.IO;
using System.Xml;
using System.Xml.Schema;
...
XmlSchemaCollection xsc = new XmlSchemaCollection();
xsc.Add("urn: bookstore - schema", new XmlTextReader(new FileStream("xmlFilename", FileMode.Open)));
```

CA3075: Insecure DTD Processing

9/20/2022 • 5 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA3075
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

If you use insecure [DtdProcessing](#) instances or reference external entity sources, the parser may accept untrusted input and disclose sensitive information to attackers.

Rule description

A *Document Type Definition (DTD)* is one of two ways an XML parser can determine the validity of a document, as defined by the [World Wide Web Consortium \(W3C\) Extensible Markup Language \(XML\) 1.0](#). This rule seeks properties and instances where untrusted data is accepted to warn developers about potential [Information Disclosure](#) threats or [Denial of Service \(DoS\)](#) attacks. This rule triggers when:

- DtdProcessing is enabled on the [XmlReader](#) instance, which resolves external XML entities using [XmlUrlResolver](#).
- The [InnerText](#) property in the XML is set.
- [DtdProcessing](#) property is set to Parse.
- Untrusted input is processed using [XmlResolver](#) instead of [XmlSecureResolver](#).
- The [XmlReader.Create](#) method is invoked with an insecure [XmlReaderSettings](#) instance or no instance at all.
- [XmlReader](#) is created with insecure default settings or values.

In each of these cases, the outcome is the same: the contents from either the file system or network shares from the machine where the XML is processed will be exposed to the attacker, or DTD processing can be used as a DoS vector.

How to fix violations

- Catch and process all [XmlTextReader](#) exceptions properly to avoid path information disclosure.
- Use the [XmlSecureResolver](#) to restrict the resources that the [XmlTextReader](#) can access.
- Do not allow the [XmlReader](#) to open any external resources by setting the [XmlResolver](#) property to **null**.
- Ensure that the [DataViewManager.DataViewSettingCollectionString](#) property is assigned from a trusted source.

.NET Framework 3.5 and earlier

- Disable DTD processing if you are dealing with untrusted sources by setting the [ProhibitDtd](#) property to `true`.
- `XmlTextReader` class has a full trust inheritance demand.

.NET Framework 4 and later

- Avoid enabling `DtdProcessing` if you're dealing with untrusted sources by setting the [XmlReaderSettings.DtdProcessing](#) property to `Prohibit` or `Ignore`.
- Ensure that the `Load()` method takes an `XmlReader` instance in all `InnerXml` cases.

NOTE

This rule might report false positives on some valid `XmlSecureResolver` instances.

When to suppress warnings

Unless you're sure that the input is known to be from a trusted source, do not suppress a rule from this warning.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA3075
// The code that's violating the rule is on this line.
#pragma warning restore CA3075
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA3075.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Pseudo-code examples

Violation 1

```

using System.IO;
using System.Xml.Schema;

class TestClass
{
    public XmlSchema Test
    {
        get
        {
            var src = "";
            TextReader tr = new StreamReader(src);
            XmlSchema schema = XmlSchema.Read(tr, null); // warn
            return schema;
        }
    }
}

```

Solution 1

```

using System.IO;
using System.Xml;
using System.Xml.Schema;

class TestClass
{
    public XmlSchema Test
    {
        get
        {
            var src = "";
            TextReader tr = new StreamReader(src);
            XmlReader reader = XmlReader.Create(tr, new XmlReaderSettings() { XmlResolver = null });
            XmlSchema schema = XmlSchema.Read(reader, null);
            return schema;
        }
    }
}

```

Violation 2

```

using System.Xml;

namespace TestNamespace
{
    public class TestClass
    {
        public XmlReaderSettings settings = new XmlReaderSettings();
        public void TestMethod(string path)
        {
            var reader = XmlReader.Create(path, settings); // warn
        }
    }
}

```

Solution 2

```

using System.Xml;

namespace TestNamespace
{
    public class TestClass
    {
        public XmlReaderSettings settings = new XmlReaderSettings()
        {
            DtdProcessing = DtdProcessing.Prohibit
        };

        public void TestMethod(string path)
        {
            var reader = XmlReader.Create(path, settings);
        }
    }
}

```

Violation 3

```

using System.Xml;

namespace TestNamespace
{
    public class DoNotUseSetInnerXml
    {
        public void TestMethod(string xml)
        {
            XmlDocument doc = new XmlDocument() { XmlResolver = null };
            doc.InnerXml = xml; // warn
        }
    }
}

```

```

using System.Xml;

namespace TestNamespace
{
    public class DoNotUseLoadXml
    {
        public void TestMethod(string xml)
        {
            XmlDocument doc = new XmlDocument(){ XmlResolver = null };
            doc.LoadXml(xml); // warn
        }
    }
}

```

Solution 3

```

using System.Xml;

public static void TestMethod(string xml)
{
    XmlDocument doc = new XmlDocument() { XmlResolver = null };
    System.IO.StringReader sreader = new System.IO.StringReader(xml);
    XmlReader reader = XmlReader.Create(sreader, new XmlReaderSettings() { XmlResolver = null });
    doc.Load(reader);
}

```

Violation 4

```

using System.IO;
using System.Xml;
using System.Xml.Serialization;

namespace TestNamespace
{
    public class UseXmlReaderForDeserialize
    {
        public void TestMethod(Stream stream)
        {
            XmlSerializer serializer = new XmlSerializer(typeof(UseXmlReaderForDeserialize));
            serializer.Deserialize(stream); // warn
        }
    }
}

```

Solution 4

```

using System.IO;
using System.Xml;
using System.Xml.Serialization;

namespace TestNamespace
{
    public class UseXmlReaderForDeserialize
    {
        public void TestMethod(Stream stream)
        {
            XmlSerializer serializer = new XmlSerializer(typeof(UseXmlReaderForDeserialize));
            XmlReader reader = XmlReader.Create(stream, new XmlReaderSettings() { XmlResolver = null });
            serializer.Deserialize(reader );
        }
    }
}

```

Violation 5

```

using System.Xml;
using System.Xml.XPath;

namespace TestNamespace
{
    public class UseXmlReaderForXPathDocument
    {
        public void TestMethod(string path)
        {
            XPathDocument doc = new XPathDocument(path); // warn
        }
    }
}

```

Solution 5

```

using System.Xml;
using System.Xml.XPath;

namespace TestNamespace
{
    public class UseXmlReaderForXPathDocument
    {
        public void TestMethod(string path)
        {
            XmlReader reader = XmlReader.Create(path, new XmlReaderSettings() { XmlResolver = null });
            XPathDocument doc = new XPathDocument(reader);
        }
    }
}

```

Violation 6

```

using System.Xml;

namespace TestNamespace
{
    class TestClass
    {
        public XmlDocument doc = new XmlDocument() { XmlResolver = new XmlUrlResolver() };
    }
}

```

Solution 6

```

using System.Xml;

namespace TestNamespace
{
    class TestClass
    {
        public XmlDocument doc = new XmlDocument() { XmlResolver = null }; // or set to a XmlSecureResolver
instance
    }
}

```

Violation 7

```

using System.Xml;

namespace TestNamespace
{
    class TestClass
    {
        private static void TestMethod()
        {
            var reader = XmlTextReader.Create("doc.xml"); //warn
        }
    }
}

```

```
using System.Xml;

namespace TestNamespace
{
    public class TestClass
    {
        public void TestMethod(string path)
        {
            try {
                XmlTextReader reader = new XmlTextReader(path); // warn
            }
            catch { throw ; }
            finally {}
        }
    }
}
```

Solution 7

```
using System.Xml;

namespace TestNamespace
{
    public class TestClass
    {
        public void TestMethod(string path)
        {
            XmlReaderSettings settings = new XmlReaderSettings() { XmlResolver = null };
            XmlReader reader = XmlReader.Create(path, settings);
        }
    }
}
```

NOTE

Although `XmlReader.Create` is the recommended way to create an `XmlReader` instance, there are behavior differences from `XmlTextReader`. An `XmlReader` from `Create` normalizes `\r\n` to `\n` in XML values, while `XmlTextReader` preserves the `\r\n` sequence.

CA3076: Insecure XSLT Script Execution

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA3076
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

If you execute Extensible Stylesheets Language Transformations (XSLT) in .NET applications insecurely, the processor may resolve untrusted URL references that could disclose sensitive information to attackers, leading to Denial of Service and Cross-Site attacks. For more information, see [XSLT Security Considerations\(.NET Guide\)](#).

Rule description

XSLT is a World Wide Web Consortium (W3C) standard for transforming XML data. XSLT is typically used to write style sheets to transform XML data to other formats such as HTML, fixed-length text, comma-separated text, or a different XML format. Although prohibited by default, you may choose to enable it for your project.

To ensure you're not exposing an attack surface, this rule triggers whenever the `XslCompiledTransform.Load` receives insecure combination instances of `XsltSettings` and `XmlResolver`, which allows malicious script processing.

How to fix violations

- Replace the insecure `XsltSettings` argument with `XsltSettings.Default` or with an instance that has disabled document function and script execution.
- Replace the `XmlResolver` argument with null or an `XmlSecureResolver` instance.

When to suppress warnings

Unless you're sure that the input is known to be from a trusted source, do not suppress a rule from this warning.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA3076
// The code that's violating the rule is on this line.
#pragma warning restore CA3076
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA3076.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Pseudo-code examples

Violation that uses `XsltSettings.TrustedXslt`

```
using System.Xml;
using System.Xml.Xsl;

namespace TestNamespace
{
    class TestClass
    {
        void TestMethod()
        {
            XslCompiledTransform xslCompiledTransform = new XslCompiledTransform();
            var settings = XsltSettings.TrustedXslt;
            var resolver = new XmlUrlResolver();
            xslCompiledTransform.Load("testStylesheet", settings, resolver); // warn
        }
    }
}
```

Solution that uses `XsltSettings.Default`

```
using System.Xml;
using System.Xml.Xsl;

namespace TestNamespace
{
    class TestClass
    {
        void TestMethod()
        {
            XslCompiledTransform xslCompiledTransform = new XslCompiledTransform();
            var settings = XsltSettings.Default;
            var resolver = new XmlUrlResolver();
            xslCompiledTransform.Load("testStylesheet", settings, resolver);
        }
    }
}
```

Violation—document function and script execution not disabled

```

using System.Xml;
using System.Xml.Xsl;

namespace TestNamespace
{
    class TestClass
    {
        private static void TestMethod(XsltSettings settings)
        {
            try
            {
                XslCompiledTransform xslCompiledTransform = new XslCompiledTransform();
                var resolver = new XmlUrlResolver();
                xslCompiledTransform.Load("testStylesheet", settings, resolver); // warn
            }
            catch { throw; }
            finally { }
        }
    }
}

```

Solution—disable document function and script execution

```

using System.Xml;
using System.Xml.Xsl;

namespace TestNamespace
{
    class TestClass
    {
        private static void TestMethod(XsltSettings settings)
        {
            try
            {
                XslCompiledTransform xslCompiledTransform = new XslCompiledTransform();
                settings.EnableDocumentFunction = false;
                settings.EnableScript = false;
                var resolver = new XmlUrlResolver();
                xslCompiledTransform.Load("testStylesheet", settings, resolver);
            }
            catch { throw; }
            finally { }
        }
    }
}

```

See also

- [XSLT Security Considerations\(.NET Guide\)](#)

CA3077: Insecure Processing in API Design, XML Document and XML Text Reader

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA3077
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

When designing an API derived from `XMLDocument` and `XMLTextReader`, be mindful of [DtdProcessing](#). Using insecure `DTDProcessing` instances when referencing or resolving external entity sources or setting insecure values in the XML may lead to information disclosure.

Rule description

A *Document Type Definition (DTD)* is one of two ways an XML parser can determine the validity of a document, as defined by the [World Wide Web Consortium \(W3C\) Extensible Markup Language \(XML\) 1.0](#). This rule seeks properties and instances where untrusted data is accepted to warn developers about potential [Information Disclosure](#) threats, which may lead to [Denial of Service \(DoS\)](#) attacks. This rule triggers when:

- `XmlDocument` or `XmlTextReader` classes use default resolver values for DTD processing .
- No constructor is defined for the `XmlDocument` or `XmlTextReader` derived classes or no secure value is used for `XmlResolver`.

How to fix violations

- Catch and process all `XmlTextReader` exceptions properly to avoid path information disclosure .
- Use `XmlSecureResolver` instead of `XmlResolver` to restrict the resources the `XmlTextReader` can access.

When to suppress warnings

Unless you're sure that the input is known to be from a trusted source, do not suppress a rule from this warning.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA3077
// The code that's violating the rule is on this line.
#pragma warning restore CA3077
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA3077.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Pseudo-code examples

Violation

```
using System;
using System.Xml;

namespace TestNamespace
{
    class TestClass : XmlDocument
    {
        public TestClass () {} // warn
    }

    class TestClass2 : XmlTextReader
    {
        public TestClass2() // warn
        {
        }
    }
}
```

Solution

```
using System;
using System.Xml;

namespace TestNamespace
{
    class TestClass : XmlDocument
    {
        public TestClass ()
        {
            XmlResolver = null;
        }
    }

    class TestClass2 : XmlTextReader
    {
        public TestClass2()
        {
            XmlResolver = null;
        }
    }
}
```

CA3147: Mark verb handlers with ValidateAntiForgeryTokenToken

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA3147
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

An ASP.NET MVC controller action method isn't marked with [ValidateAntiForgeryTokenAttribute](#), or an attribute specifying the HTTP verb, such as [HttpGetAttribute](#) or [AcceptVerbsAttribute](#).

Rule description

When designing an ASP.NET MVC controller, be mindful of cross-site request forgery attacks. A cross-site request forgery attack can send malicious requests from an authenticated user to your ASP.NET MVC controller. For more information, see [XSRF/CSRF prevention in ASP.NET MVC and web pages](#).

This rule checks that ASP.NET MVC controller action methods either:

- Have the [ValidateAntiForgeryTokenAttribute](#) and specify allowed HTTP verbs, not including HTTP GET.
- Specify HTTP GET as an allowed verb.

How to fix violations

- For ASP.NET MVC controller actions that handle HTTP GET requests and don't have potentially harmful side effects, add an [HttpGetAttribute](#) to the method.

If you have an ASP.NET MVC controller action that handles HTTP GET requests and has potentially harmful side effects such as modifying sensitive data, then your application is vulnerable to cross-site request forgery attacks. You'll need to redesign your application so that only HTTP POST, PUT, or DELETE requests perform sensitive operations.

- For ASP.NET MVC controller actions that handle HTTP POST, PUT, or DELETE requests, add [ValidateAntiForgeryTokenAttribute](#) and attributes specifying the allowed HTTP verbs ([AcceptVerbsAttribute](#), [HttpPostAttribute](#), [HttpPutAttribute](#), or [HttpDeleteAttribute](#)). Additionally, you need to call the [HtmlHelper.AntiForgeryToken\(\)](#) method from your MVC view or Razor web page. For an example, see [Examining the edit methods and edit view](#).

When to suppress warnings

It's safe to suppress a warning from this rule if:

- The ASP.NET MVC controller action has no harmful side effects.

- The application validates the antiforgery token in a different way.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA3147
// The code that's violating the rule is on this line.
#pragma warning restore CA3147
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA3147.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

ValidateAntiForgeryToken attribute example

Violation:

```
namespace TestNamespace
{
    using System.Web.Mvc;

    public class TestController : Controller
    {
        public ActionResult TransferMoney(string toAccount, string amount)
        {
            // You don't want an attacker to specify to who and how much money to transfer.

            return null;
        }
    }
}
```

Solution:

```
using System;
using System.Xml;

namespace TestNamespace
{
    using System.Web.Mvc;

    public class TestController : Controller
    {
        [HttpPost]
        [ValidateAntiForgeryToken]
        public ActionResult TransferMoney(string toAccount, string amount)
        {
            return null;
        }
    }
}
```

HttpGet attribute example

Violation:

```
namespace TestNamespace
{
    using System.Web.Mvc;

    public class TestController : Controller
    {
        public ActionResult Help(int topicId)
        {
            // This Help method is an example of a read-only operation with no harmful side effects.
            return null;
        }
    }
}
```

Solution:

```
namespace TestNamespace
{
    using System.Web.Mvc;

    public class TestController : Controller
    {
        [HttpGet]
        public ActionResult Help(int topicId)
        {
            return null;
        }
    }
}
```

CA5350: Do Not Use Weak Cryptographic Algorithms

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA5350
Category	Security
Fix is breaking or non-breaking	Non-breaking

Note

This warning was last updated on November 2015.

Cause

Encryption algorithms such as [TripleDES](#) and hashing algorithms such as [SHA1](#) and [RIPEMD160](#) are considered to be weak.

These cryptographic algorithms do not provide as much security assurance as more modern counterparts. Cryptographic hashing algorithms [SHA1](#) and [RIPEMD160](#) provide less collision resistance than more modern hashing algorithms. The encryption algorithm [TripleDES](#) provides fewer bits of security than more modern encryption algorithms.

Rule description

Weak encryption algorithms and hashing functions are used today for a number of reasons, but they should not be used to guarantee the confidentiality of the data they protect.

The rule triggers when it finds 3DES, SHA1 or RIPEMD160 algorithms in the code and throws a warning to the user.

How to fix violations

Use cryptographically stronger options:

- For TripleDES encryption, use [Aes](#) encryption.
- For SHA1 or RIPEMD160 hashing functions, use ones in the [SHA-2](#) family (e.g. [SHA512](#), [SHA384](#), [SHA256](#)).

When to suppress warnings

SUPPRESS a warning from this rule when the level of protection needed for the data does not require a security guarantee.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA5350
// The code that's violating the rule is on this line.
#pragma warning restore CA5350
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA5350.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Pseudo-code examples

As of the time of this writing, the following pseudo-code sample illustrates the pattern detected by this rule.

SHA-1 Hashing Violation

```
using System.Security.Cryptography;
...
var hashAlg = SHA1.Create();
```

Solution:

```
using System.Security.Cryptography;
...
var hashAlg = SHA256.Create();
```

RIPEMD160 Hashing Violation

```
using System.Security.Cryptography;
...
var hashAlg = RIPEMD160Managed.Create();
```

Solution:

```
using System.Security.Cryptography;
...
var hashAlg = SHA256.Create();
```

TripleDES Encryption Violation

```
using System.Security.Cryptography;
...
using (TripleDES encAlg = TripleDES.Create())
{
    ...
}
```

Solution:

```
using System.Security.Cryptography;
...
using (AesManaged encAlg = new AesManaged())
{
    ...
}
```

CA5351 Do Not Use Broken Cryptographic Algorithms

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA5351
Category	Security
Fix is breaking or non-breaking	Non-breaking

Note

This warning was last updated on November 2015.

Cause

Hashing functions such as [MD5](#) and encryption algorithms such as [DES](#) and [RC2](#) can expose significant risk and may result in the exposure of sensitive information through trivial attack techniques, such as brute force attacks and hash collisions.

The cryptographic algorithms list below are subject to known cryptographic attacks. The cryptographic hash algorithm [MD5](#) is subject to hash collision attacks. Depending on the usage, a hash collision may lead to impersonation, tampering, or other kinds of attacks on systems that rely on the unique cryptographic output of a hashing function. The encryption algorithms [DES](#) and [RC2](#) are subject to cryptographic attacks that may result in unintended disclosure of encrypted data.

Rule description

Broken cryptographic algorithms are not considered secure and their use should be discouraged. The MD5 hash algorithm is susceptible to known collision attacks, though the specific vulnerability will vary based on the context of use. Hashing algorithms used to ensure data integrity (for example, file signature or digital certificate) are particularly vulnerable. In this context, attackers could generate two separate pieces of data, such that benign data can be substituted with malicious data, without changing the hash value or invalidating an associated digital signature.

For encryption algorithms:

- [DES](#) encryption contains a small key size, which could be brute-forced in less than a day.
- [RC2](#) encryption is susceptible to a related-key attack, where the attacker finds mathematical relationships between all key values.

This rule triggers when it finds any of the above cryptographic functions in source code and throws a warning to the user.

How to fix violations

Use cryptographically stronger options:

- For MD5, use hashes in the [SHA-2](#) family (for example, [SHA512](#), [SHA384](#), [SHA256](#)).
- For DES and RC2, use [Aes](#) encryption.

When to suppress warnings

Do not suppress a warning from this rule, unless it's been reviewed by a cryptographic expert.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA5351
// The code that's violating the rule is on this line.
#pragma warning restore CA5351
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA5351.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Pseudo-code examples

The following pseudo-code samples illustrate the pattern detected by this rule and possible alternatives.

MD5 Hashing Violation

```
using System.Security.Cryptography;
...
var hashAlg = MD5.Create();
```

Solution:

```
using System.Security.Cryptography;
...
var hashAlg = SHA256.Create();
```

RC2 Encryption Violation

```
using System.Security.Cryptography;
...
RC2 encAlg = RC2.Create();
```

Solution:

```
using System.Security.Cryptography;
...
using (AesManaged encAlg = new AesManaged())
{
    ...
}
```

DES Encryption Violation

```
using System.Security.Cryptography;
...
DES encAlg = DES.Create();
```

Solution:

```
using System.Security.Cryptography;
...
using (AesManaged encAlg = new AesManaged())
{
    ...
}
```

CA5358: Do Not Use Unsafe Cipher Modes

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA5358
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

Use of one of the following unsafe encryption modes that is not approved:

- [System.Security.Cryptography.CipherMode.ECB](#)
- [System.Security.Cryptography.CipherMode.OFB](#)
- [System.Security.Cryptography.CipherMode.CFB](#)

Rule description

These modes are vulnerable to attacks and may cause exposure of sensitive information. For example, using [ECB](#) to encrypt a plaintext block always produces a same cipher text, so it can easily tell if two encrypted messages are identical. Using approved modes can avoid these unnecessary risks.

How to fix violations

- Use only approved modes ([System.Security.Cryptography.CipherMode.CBC](#), [System.Security.Cryptography.CipherMode.CTS](#)).

When to suppress warnings

It's safe to suppress a warning from this rule if:

- Cryptography experts have reviewed and approved the cipher mode's usage.
- The referenced [CipherMode](#) isn't used for a cryptographic operation.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA5358
// The code that's violating the rule is on this line.
#pragma warning restore CA5358
```

To disable the rule for a file, folder, or project, set its severity to [none](#) in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA5358.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Pseudo-code examples

Assign ECB to Mode property

```
using System.Security.Cryptography;

class ExampleClass {
    private static void ExampleMethod () {
        RijndaelManaged rijn = new RijndaelManaged
        {
            Mode = CipherMode.ECB
        };
    }
}
```

Using the value ECB

```
using System;
using System.Security.Cryptography;

class ExampleClass
{
    private static void ExampleMethod()
    {
        Console.WriteLine(CipherMode.ECB);
    }
}
```

Solution

```
using System.Security.Cryptography;

class ExampleClass {
    private static void ExampleMethod () {
        RijndaelManaged rijn = new RijndaelManaged
        {
            Mode = CipherMode.CBC
        };
    }
}
```

CA5359: Do not disable certificate validation

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA5359
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

The callback assigned to `ServicePointManager.ServerCertificateValidationCallback` always returns `true`.

Rule description

A certificate can help authenticate the identity of the server. Clients should validate the server certificate to ensure requests are sent to the intended server. If the `ServicePointManager.ServerCertificateValidationCallback` always returns `true`, then by default any certificate will pass validation for all outgoing HTTPS requests.

How to fix violations

- Considering overriding certificate validation logic on the specific outgoing HTTPS requests that require custom certificate validation, instead of overriding the global `ServicePointManager.ServerCertificateValidationCallback`.
- Apply custom validation logic to only specific hostnames and certificates, and otherwise check that the `SslPolicyErrors` enum value is `None`.

When to suppress warnings

If multiple delegates are attached to `ServerCertificateValidationCallback`, only the value from the last delegate is respected, so it's safe to suppress warnings from other delegates. However, you may want to remove the unused delegates entirely.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA5359
// The code that's violating the rule is on this line.
#pragma warning restore CA5359
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA5359.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Pseudo-code examples

Violation

```
using System.Net;

class ExampleClass
{
    public void ExampleMethod()
    {
        ServicePointManager.ServerCertificateValidationCallback += (sender, cert, chain, error) => { return
true; };
    }
}
```

Solution

```
using System.Net;
using System.Net.Security;
using System.Security.Cryptography.X509Certificates;

class ExampleClass
{
    public void ExampleMethod()
    {
        ServicePointManager.ServerCertificateValidationCallback += SelfSignedForlocalhost;
    }

    private static bool SelfSignedForlocalhost(object sender, X509Certificate certificate, X509Chain chain,
SslPolicyErrors sslPolicyErrors)
    {
        if (sslPolicyErrors == SslPolicyErrors.None)
        {
            return true;
        }

        // For HTTPS requests to this specific host, we expect this specific certificate.
        // In practice, you'd want this to be configurable and allow for multiple certificates per host, to
enable
        // seamless certificate rotations.
        return sender is HttpWebRequest httpWebRequest
            && httpWebRequest.RequestUri.Host == "localhost"
            && certificate is X509Certificate2 x509Certificate2
            && x509Certificate2.Thumbprint == "AAAAAAAAAAAAAAAAAAAAAAA"
            && sslPolicyErrors == SslPolicyErrors.RemoteCertificateChainErrors;
    }
}
```

CA5360: Do not call dangerous methods in deserialization

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA5360
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

Calling one of the following dangerous methods in deserialization:

- [System.IO.Directory.Delete](#)
- [System.IO.DirectoryInfo.Delete](#)
- [System.IO.File.AppendAllLines](#)
- [System.IO.File.AppendAllText](#)
- [System.IO.File.AppendText](#)
- [System.IO.File.Copy](#)
- [System.IO.File.Delete](#)
- [System.IO.File.WriteAllBytes](#)
- [System.IO.File.WriteAllLines](#)
- [System.IO.File.WriteAllText](#)
- [System.IO.FileInfo.Delete](#)
- [System.IO.Log.LogStore.Delete](#)
- [System.Reflection.Assembly.GetLoadedModules](#)
- [System.Reflection.Assembly.Load](#)
- [System.Reflection.Assembly.LoadFrom](#)
- [System.Reflection.Assembly.LoadFile](#)
- [System.Reflection.Assembly.LoadModule](#)
- [System.Reflection.Assembly.LoadWithPartialName](#)
- [System.Reflection.Assembly.ReflectionOnlyLoad](#)
- [System.Reflection.Assembly.ReflectionOnlyLoadFrom](#)
- [System.Reflection.Assembly.UnsafeLoadFrom](#)

All methods meets one of the following requirements could be the callback of deserialization:

- Marked with [System.Runtime.Serialization.OnDeserializingAttribute](#).
- Marked with [System.Runtime.Serialization.OnDeserializedAttribute](#).
- Implementing [System.Runtime.Serialization.IDeserializationCallback.OnDeserialization](#).
- Implementing [System.IDisposable.Dispose](#).
- Is a destructor.

Rule description

Insecure deserialization is a vulnerability which occurs when untrusted data is used to abuse the logic of an application, inflict a Denial-of-Service (DoS) attack, or even execute arbitrary code upon it being deserialized. It's frequently possible for malicious users to abuse these deserialization features when the application is deserializing untrusted data which is under their control. Specifically, invoke dangerous methods in the process of deserialization. Successful insecure deserialization attacks could allow an attacker to carry out attacks such as DoS attacks, authentication bypasses, and remote code execution.

How to fix violations

Remove these dangerous methods from automatically run deserialization callbacks. Call dangerous methods only after validating the input.

When to suppress warnings

It's safe to suppress this rule if:

- You know the input is trusted. Consider that your application's trust boundary and data flows may change over time.
- The serialized data is tamper-proof. After serialization, cryptographically sign the serialized data. Before deserialization, validate the cryptographic signature. Protect the cryptographic key from being disclosed and design for key rotations.
- The data is validated as safe to the application.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA5360
// The code that's violating the rule is on this line.
#pragma warning restore CA5360
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA5360.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Pseudo-code examples

Violation

```
using System;
using System.IO;
using System.Runtime.Serialization;

[Serializable()]
public class ExampleClass : IDeserializationCallback
{
    private string member;

    void IDeserializationCallback.OnDeserialization(Object sender)
    {
        var sourceFileName = "malicious file";
        var destFileName = "sensitive file";
        File.Copy(sourceFileName, destFileName);
    }
}
```

Solution

```
using System;
using System.IO;
using System.Runtime.Serialization;

[Serializable()]
public class ExampleClass : IDeserializationCallback
{
    private string member;

    void IDeserializationCallback.OnDeserialization(Object sender)
    {
        var sourceFileName = "malicious file";
        var destFileName = "sensitive file";
        // Remove the potential dangerous operation.
        // File.Copy(sourceFileName, destFileName);
    }
}
```

CA5361: Do not disable SChannel use of strong crypto

9/20/2022 • 3 minutes to read • [Edit Online](#)

	Value
Rule ID	CA5361
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

A `AppContext.SetSwitch` method call sets `Switch.System.Net.DontEnableSchUseStrongCrypto` to `true`.

By default, this rule analyzes the entire codebase, but this is [configurable](#).

Rule description

Setting `Switch.System.Net.DontEnableSchUseStrongCrypto` to `true` weakens the cryptography used in outgoing Transport Layer Security (TLS) connections. Weaker cryptography can compromise the confidentiality of communication between your application and the server, making it easier for attackers to eavesdrop sensitive data. For more information, see [Transport Layer Security \(TLS\) best practices with .NET Framework](#).

How to fix violations

- If your application targets .NET Framework v4.6 or later, you can either remove the `AppContext.SetSwitch` method call, or set the switch's value to `false`.
- If your application targets .NET Framework earlier than v4.6 and runs on .NET Framework v4.6 or later, set the switch's value to `false`.
- Otherwise, refer to [Transport Layer Security \(TLS\) best practices with .NET Framework](#) for mitigations.

When to suppress warnings

You can suppress this warning if you need to connect to a legacy service that can't be upgraded to use secure TLS configurations.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA5361
// The code that's violating the rule is on this line.
#pragma warning restore CA5361
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA5361.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

Use the following options to configure which parts of your codebase to run this rule on.

- [Exclude specific symbols](#)
- [Exclude specific types and their derived types](#)

You can configure these options for just this rule, for all rules, or for all rules in this category ([Security](#)). For more information, see [Code quality rule configuration options](#).

Exclude specific symbols

You can exclude specific symbols, such as types and methods, from analysis. For example, to specify that the rule should not run on any code within types named `MyType`, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Symbol name only (includes all symbols with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#). Each symbol name requires a symbol-kind prefix, such as `M:` for methods, `T:` for types, and `N:` for namespaces.
- `.ctor` for constructors and `.cctor` for static constructors.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType</code>	Matches all symbols named <code>MyType</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType1 MyType2</code>	Matches all symbols named either <code>MyType1</code> or <code>MyType2</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS.MyType.MyMethod(ParamType)</code>	Matches specific method <code>MyMethod</code> with the specified fully qualified signature.
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS1.MyType1.MyMethod1(ParamType) M:NS2.MyType2.MyMethod2(ParamType)</code>	Matches specific methods <code>MyMethod1</code> and <code>MyMethod2</code> with the respective fully qualified signatures.

Exclude specific types and their derived types

You can exclude specific types and their derived types from analysis. For example, to specify that the rule should not run on any methods within types named `MyType` and their derived types, add the following key-value pair

to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Type name only (includes all types with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#), with an optional `T:` prefix.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType</code>	Matches all types named <code>MyType</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType1 MyType2</code>	Matches all types named either <code>MyType1</code> or <code>MyType2</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS.MyType</code>	Matches specific type <code>MyType</code> with given fully qualified name and all of its derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS1.MyType1 M:NS2.MyType2</code>	Matches specific types <code>MyType1</code> and <code>MyType2</code> with the respective fully qualified names, and all of their derived types.

Pseudo-code examples

Violation

```
using System;

public class ExampleClass
{
    public void ExampleMethod()
    {
        // CA5361 violation
        ApplicationContext.SetSwitch("Switch.System.Net.DontEnableSchUseStrongCrypto", true);
    }
}
```

```
Imports System

Public Class ExampleClass
    Public Sub ExampleMethod()
        ' CA5361 violation
        ApplicationContext.SetSwitch("Switch.System.Net.DontEnableSchUseStrongCrypto", true)
    End Sub
End Class
```

Solution

```
using System;

public class ExampleClass
{
    public void ExampleMethod()
    {
        ApplicationContext.SetSwitch("Switch.System.Net.DontEnableSchUseStrongCrypto", false);
    }
}
```

```
Imports System

Public Class ExampleClass
    Public Sub ExampleMethod()
        ApplicationContext.SetSwitch("Switch.System.Net.DontEnableSchUseStrongCrypto", false)
    End Sub
End Class
```

CA5362: Potential reference cycle in deserialized object graph

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA5362
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

A class marked with the [System.SerializableAttribute](#) has a field or property may refer to the containing object directly or indirectly, allowing for a potential reference cycle.

Rule description

If deserializing untrusted data, then any code processing the deserialized object graph needs to handle reference cycles without going into infinite loops. This includes both code that's part of a deserialization callback and code that processes the object graph after deserialization completed. Otherwise, an attacker could perform a Denial-of-Service attack with malicious data containing a reference cycle.

This rule doesn't necessarily mean there's a vulnerability, but just flags potential reference cycles in serialized object graphs.

How to fix violations

Don't serialize the class and remove the [SerializableAttribute](#). Or, redesign your application so that the self-referred members can be removed out of the serializable class.

When to suppress warnings

It's safe to suppress a warning from this rule if:

- You know the input is trusted. Consider that your application's trust boundary and data flows may change over time.
- All code processing the serialized data detects and handles reference cycles without going into an infinite loop or using excessive resources.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA5362
// The code that's violating the rule is on this line.
#pragma warning restore CA5362
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA5362.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Pseudo-code examples

Potential reference cycle violation

```
using System;

[Serializable()]
class ExampleClass
{
    public ExampleClass ExampleProperty {get; set;}

    public int NormalProperty {get; set;}
}

class AnotherClass
{
    // The argument passed by could be `JsonConvert.DeserializeObject<ExampleClass>(untrustedData)` .
    public void AnotherMethod(ExampleClass ec)
    {
        while(ec != null)
        {
            Console.WriteLine(ec.ToString());
            ec = ec.ExampleProperty;
        }
    }
}
```

Solution

```
using System;

[Serializable()]
class ExampleClass
{
    [NonSerialized]
    public ExampleClass ExampleProperty {get; set;}

    public int NormalProperty {get; set;}
}

class AnotherClass
{
    // The argument passed by could be ` JsonConvert.DeserializeObject<ExampleClass>(untrustedData)` .
    public void AnotherMethod(ExampleClass ec)
    {
        while(ec != null)
        {
            Console.WriteLine(ec.ToString());
            ec = ec.ExampleProperty;
        }
    }
}
```

CA5363: Do not disable request validation

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA5363
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

The attribute `ValidateInput` is set to `false` for a class or method.

Rule description

Request validation is a feature in ASP.NET that examines HTTP requests and determines whether they contain potentially dangerous content that can lead to injection attacks, including cross-site-scripting.

How to fix violations

Set the `ValidateInput` attribute to `true` or delete it entirely. Alternatively, use `AllowHTMLAttribute` to allow HTML in specific parts of the input.

When to suppress warnings

You can suppress this violation if all the payload in the incoming HTTP request is sourced from a trusted entity and could not be tampered with by an adversary prior to or during transport.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA5363
// The code that's violating the rule is on this line.
#pragma warning restore CA5363
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA5363.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Pseudo-code examples

Violation

The following pseudo-code sample illustrates the pattern detected by this rule. This disables input validation.

```
using System.Web.Mvc;

class TestControllerClass
{
    [ValidateInput(false)]
    public void TestActionMethod()
    {
    }
}
```

Solution

```
using System.Web.Mvc;

class TestControllerClass
{
    [ValidateInput(true)]
    public void TestActionMethod()
    {
    }
}
```

CA5364: Do not use deprecated security protocols

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA5364
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

This rule fires when either of the following conditions are met:

- A deprecated `System.Net.SecurityProtocolType` value was referenced.
- An integer value representing a deprecated value was assigned to a `SecurityProtocolType` variable.

Deprecated values are:

- Ssl3
- Tls
- Tls10
- Tls11

Rule description

Transport Layer Security (TLS) secures communication between computers, most commonly with Hypertext Transfer Protocol Secure (HTTPS). Older protocol versions of TLS are less secure than TLS 1.2 and TLS 1.3 and are more likely to have new vulnerabilities. Avoid older protocol versions to minimize risk. For guidance on identifying and removing deprecated protocol versions, see [Solving the TLS 1.0 Problem, 2nd Edition](#).

How to fix violations

Don't use deprecated TLS protocol versions.

When to suppress warnings

You can suppress this warning if:

- The reference to the deprecated protocol version isn't being used to enable a deprecated version.
- You need to connect to a legacy service that can't be upgraded to use secure TLS configurations.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA5364
// The code that's violating the rule is on this line.
#pragma warning restore CA5364
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA5364.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Pseudo-code examples

Enumeration name violation

```
using System;
using System.Net;

public class ExampleClass
{
    public void ExampleMethod()
    {
        // CA5364 violation
        ServicePointManager.SecurityProtocol = SecurityProtocolType.Tls11 | SecurityProtocolType.Tls12;
    }
}
```

```
Imports System
Imports System.Net

Public Class TestClass
    Public Sub ExampleMethod()
        ' CA5364 violation
        ServicePointManager.SecurityProtocol = SecurityProtocolType.Tls11 Or SecurityProtocolType.Tls12
    End Sub
End Class
```

Integer value violation

```
using System;
using System.Net;

public class ExampleClass
{
    public void ExampleMethod()
    {
        // CA5364 violation
        ServicePointManager.SecurityProtocol = (SecurityProtocolType) 768;      // TLS 1.1
    }
}
```

```
Imports System
Imports System.Net

Public Class TestClass
    Public Sub ExampleMethod()
        ' CA5364 violation
        ServicePointManager.SecurityProtocol = CType(768, SecurityProtocolType)      ' TLS 1.1
    End Sub
End Class
```

Solution

```
using System;
using System.Net;

public class TestClass
{
    public void TestMethod()
    {
        // Let the operating system decide what TLS protocol version to use.
        // See https://docs.microsoft.com/dotnet/framework/network-programming/tls
    }
}
```

```
Imports System
Imports System.Net

Public Class TestClass
    Public Sub ExampleMethod()
        ' Let the operating system decide what TLS protocol version to use.
        ' See https://docs.microsoft.com/dotnet/framework/network-programming/tls
    End Sub
End Class
```

Related rules

[CA5386: Avoid hardcoding SecurityProtocolType value](#)

[CA5397: Do not use deprecated SslProtocols values](#)

[CA5398: Avoid hardcoded SslProtocols values](#)

CA5365: Do Not Disable HTTP Header Checking

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA5365
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

Set `EnableHeaderChecking` to `false`.

Rule description

HTTP header checking enables encoding of the carriage return and newline characters, `\r` and `\n`, that are found in response headers. This encoding can help to avoid injection attacks that exploit an application that echoes untrusted data contained in the header.

How to fix violations

Set `EnableHeaderChecking` to `true`. Or, remove the assignment to `false` because the default value is `true`.

When to suppress warnings

HTTP header continuations rely on headers spanning multiple lines and require new lines in them. If you need to use header continuations, you need to set the `EnableHeaderChecking` property to `false`. There is a performance impact from checking the headers. If you are certain you are already doing the right checks, turning off this feature can improve the performance of your application. Before you disable this feature, be sure you are already taking the right precautions in this area.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA5365
// The code that's violating the rule is on this line.
#pragma warning restore CA5365
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA5365.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Pseudo-code examples

```
using System;
using System.Web.Configuration;

class TestClass
{
    public void TestMethod()
    {
        HttpRuntimeSection httpRuntimeSection = new HttpRuntimeSection();
        httpRuntimeSection.EnableHeaderChecking = false;
    }
}
```

Solution

```
using System;
using System.Web.Configuration;

class TestClass
{
    public void TestMethod()
    {
        HttpRuntimeSection httpRuntimeSection = new HttpRuntimeSection();
        httpRuntimeSection.EnableHeaderChecking = true;
    }
}
```

CA5366: Use XmlReader For DataSet Read XML

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA5366
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

A Document Type Definition (DTD) defines the structure and the legal elements and attributes of an XML document. Referring to a DTD from an external resource could cause potential Denial of Service (DoS) attacks. Most readers cannot disable DTD processing and restrict external references loading except for [System.Xml.XmlReader](#). Using these other readers to load XML by one of the following methods triggers this rule:

- [ReadXml](#)
- [ReadXmlSchema](#)
- [ReadXmlSerializable](#)

Rule description

Using a [System.Data.DataSet](#) to read XML with untrusted data may load dangerous external references, which should be restricted by using an [XmlReader](#) with a secure resolver or with DTD processing disabled.

How to fix violations

Use [XmlReader](#) or its derived classes to read XML.

When to suppress warnings

Suppress a warning from this rule when dealing with a trusted data source.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA5366
// The code that's violating the rule is on this line.
#pragma warning restore CA5366
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA5366.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Pseudo-code examples

Violation

```
using System.Data;
using System.IO;

public class ExampleClass
{
    public void ExampleMethod()
    {
        new DataSet().ReadXml(new FileStream("xmlFilename", FileMode.Open));
    }
}
```

Solution

```
using System.Data;
using System.IO;
using System.Xml;

public class ExampleClass
{
    public void ExampleMethod()
    {
        new DataSet().ReadXml(new XmlTextReader(new FileStream("xmlFilename", FileMode.Open)));
    }
}
```

CA5367: Do not serialize types with pointer fields

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA5367
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

Pointers are not type safe, which means you cannot guarantee the correctness of the memory they point at. So, serializing types with pointer fields is a security risk, as it may allow an attacker to control the pointer.

Rule description

This rule checks whether there's a serializable class with a pointer field or property. Members that can't be serialized can be a pointer, such as static members or fields marked with [System.NonSerializedAttribute](#).

How to fix violations

Don't use pointer types for members in a serializable class or don't serialize the members that are pointers.

When to suppress warnings

Don't take the risk to use pointers in serializable types.

Pseudo-code examples

Violation

```
using System;

[Serializable()]
unsafe class TestClassA
{
    private int* pointer;
}
```

Solution 1

```
using System;

[Serializable()]
unsafe class TestClassA
{
    private int i;
}
```

Solution 2

```
using System;

[Serializable()]
unsafe class TestClassA
{
    private static int* pointer;
}
```

CA5368: Set ViewStateUserKey For Classes Derived From Page

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA5368
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

The `Page.ViewStateUserKey` property is not assigned in `Page.OnInit` or the `Page_Init` method.

Rule description

When designing an ASP.NET Web Form, be mindful of cross-site request forgery (CSRF) attacks. A CSRF attack can send malicious requests from an authenticated user to your ASP.NET Web Form.

One way of protecting against CSRF attacks in ASP.NET Web Form is by setting a page's `ViewStateUserKey` to a string that is unpredictable and unique to a session. For more information, see [Take Advantage of ASP.NET Built-in Features to Fend Off Web Attacks](#).

How to fix violations

Set the `ViewStateUserKey` property to a unpredictable and unique string per session. For example, if you use ASP.NET session state, `HttpSessionState.SessionID` will work.

When to suppress warnings

It's safe to suppress a warning from this rule if:

- The ASP.NET Web Form page does not perform sensitive operations.
- Cross-site request forgery attacks are mitigated in a way that this rule doesn't detect. For example, if the page inherits from a master page that contains CSRF defenses.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA5368
// The code that's violating the rule is on this line.
#pragma warning restore CA5368
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA5368.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Pseudo-code examples

Violation

```
using System;
using System.Web.UI;

class ExampleClass : Page
{
    protected override void OnInit (EventArgs e)
    {
    }
}
```

Solution

```
using System;
using System.Web.UI;

class ExampleClass : Page
{
    protected override void OnInit (EventArgs e)
    {
        // Assuming that your page makes use of ASP.NET session state and the SessionID is stable.
        ViewStateUserKey = Session.SessionID;
    }
}
```

CA5369: Use XmlReader for Deserialize

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA5369
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

Deserializing untrusted XML input with `XmlSerializer.Deserialize` instantiated without an `XmlReader` object can potentially lead to denial of service, information disclosure, and server-side request forgery attacks. These attacks are enabled by untrusted DTD and XML schema processing, which allows for the inclusion of XML bombs and malicious external entities in the XML. Only with `XmlReader` is it possible to disable DTD. Inline XML schema processing as `XmlReader` has the `ProhibitDtd` and `ProcessInlineSchema` property set to `false` by default in .NET Framework version 4.0 and later. The other options such as `Stream`, `TextReader`, and `XmlSerializationReader` cannot disable DTD processing.

Rule description

Processing untrusted DTD and XML schemas may enable loading dangerous external references, which should be restricted by using an `XmlReader` with a secure resolver or with DTD and XML inline schema processing disabled. This rule detects code that uses the `XmlSerializer.Deserialize` method and does not take `XmlReader` as a constructor parameter.

How to fix violations

Do not use `XmlSerializer.Deserialize` overloads other than `Deserialize(XmlReader)`, `Deserialize(XmlReader, String)`, `Deserialize(XmlReader, XmlDeserializationEvents)`, or `Deserialize(XmlReader, String, XmlDeserializationEvents)`.

When to suppress warnings

You can potentially suppress this warning if the parsed XML comes from a trusted source and hence cannot be tampered with.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA5369
// The code that's violating the rule is on this line.
#pragma warning restore CA5369
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA5369.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Pseudo-code examples

Violation

The following pseudo-code sample illustrates the pattern detected by this rule. The type of the first parameter of `XmlSerializer.Deserialize` is not `XmlReader` or a derived class thereof.

```
using System.IO;
using System.Xml.Serialization;
...
new XmlSerializer(typeof(TestClass)).Deserialize(new FileStream("filename", FileMode.Open));
```

Solution

```
using System.IO;
using System.Xml;
using System.Xml.Serialization;
...
new XmlSerializer(typeof(TestClass)).Deserialize(XmlReader.Create (new FileStream("filename",
FileMode.Open)));
```

CA5370: Use XmlReader for validating reader

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA5370
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

Validating untrusted XML input with the `XmlValidatingReader` class instantiated without an `XmlReader` object can potentially lead to denial of service, information disclosure, and server-side request forgery. These attacks are enabled by untrusted DTD and XML schema processing, which allows for the inclusion of XML bombs and malicious external entities in the XML. Only with `XmlReader` is it possible to disable DTD. Inline XML schema processing as `XmlReader` has the `ProhibitDtd` and `ProcessInlineSchema` property set to `false` by default in .NET Framework starting in version 4.0.

Rule description

Processing untrusted DTD and XML schemas may enable loading dangerous external references. This dangerous loading can be restricted by using an `XmlReader` with a secure resolver or with DTD and XML inline schema processing disabled. This rule detects code that uses the `XmlValidatingReader` class without `XmlReader` as a constructor parameter.

How to fix violations

- Use `XmlValidatingReader(XmlReader)` with `ProhibitDtd` and `ProcessInlineSchema` properties set to `false`.
- Starting in .NET Framework 2.0, `XmlValidatingReader` is considered obsolete. You can instantiate a validating reader with `XmlReader.Create`.

When to suppress warnings

You can potentially suppress this warning if the `XmlValidatingReader` is always used to validate XML that comes from a trusted source and hence cannot be tampered with.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA5370
// The code that's violating the rule is on this line.
#pragma warning restore CA5370
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA5370.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Pseudo-code examples

Violation

The following pseudo-code sample illustrates the pattern detected by this rule. The type of the first parameter of `XmlValidatingReader.XmlValidatingReader()` is not `XmlReader`.

```
using System;
using System.IO;
using System.Xml;
...
public void TestMethod(Stream xmlFragment, XmlNodeType fragType, XmlParserContext context)
{
    var obj = new XmlValidatingReader(xmlFragment, fragType, context);
}
```

Solution

```
using System;
using System.Xml;
...
public void TestMethod(XmlReader xmlReader)
{
    var obj = new XmlValidatingReader(xmlReader);
}
```

CA5371: Use XmlReader for schema read

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA5371
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

Processing untrusted XML input with `XmlSchema.Read` instantiated without an `XmlReader` object can potentially lead to denial of service, information disclosure, and server-side request forgery attacks. These attacks are enabled by untrusted DTD and XML schema processing, which allows for the inclusion of XML bombs and malicious external entities in the XML. Only with `XmlReader` is it possible to disable DTD. Inline XML schema processing as `XmlReader` has the `ProhibitDtd` and `ProcessInlineSchema` property set to false by default in .NET Framework starting in version 4.0. The other options such as `Stream`, `TextReader`, and `XmlSerializationReader` cannot disable DTD processing.

Rule description

Processing untrusted DTD and XML schemas may enable loading dangerous external references. Using an `XmlReader` with a secure resolver or with DTD and XML inline schema processing disabled restricts this. This rule detects code that uses the `XmlSchema.Read` method without `XmlReader` as a parameter.

How to fix violations

Use `XmlSchema.Read(XmlReader, *)` overloads.

When to suppress warnings

You can potentially suppress this warning if the `XmlSchema.Read` method is always used to process XML that comes from a trusted source and hence cannot be tampered with.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA5371
// The code that's violating the rule is on this line.
#pragma warning restore CA5371
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA5371.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Pseudo-code examples

Violation

The following pseudo-code sample illustrates the pattern detected by this rule. The type of the first parameter of `XmlSchema.Read` is not `XmlReader`.

```
using System.IO;
using System.Xml.Schema;
...
public void TestMethod(Stream stream, ValidationEventHandler validationEventHandler)
{
    XmlSchema.Read(stream, validationEventHandler);
}
```

Solution

```
using System.IO;
using System.Xml.Schema;
...
public void TestMethod(XmlReader reader, ValidationEventHandler validationEventHandler)
{
    XmlSchema.Read(reader, validationEventHandler);
}
```

CA5372: Use XmlReader for XPathDocument

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA5372
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

Using the `XPathDocument` class instantiated without an `XmlReader` object can potentially lead to denial of service, information disclosure, and server-side request forgery attacks. These attacks are enabled by untrusted DTD and XML schema processing, which allows for the inclusion of XML bombs and malicious external entities in the XML. Only with `XmlReader` is it possible to disable DTD. Inline XML schema processing as `XmlReader` has the `ProhibitDtd` and `ProcessInlineSchema` property set to false by default in .NET Framework starting in version 4.0. The other options such as `Stream`, `TextReader`, and `XmlSerializationReader` cannot disable DTD processing.

Rule description

Processing XML from untrusted data may load dangerous external references, which can be restricted by using an `XmlReader` with a secure resolver or with DTD processing disabled. This rule detects code that uses the `XPathDocument` class and doesn't take `XmlReader` as a constructor parameter.

How to fix violations

Use `XPathDocument(XmlReader, *)` constructors.

When to suppress warnings

You can potentially suppress this warning if the `XPathDocument` object is used to process an XML file that comes from a trusted source and hence cannot be tampered with.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA5372
// The code that's violating the rule is on this line.
#pragma warning restore CA5372
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA5372.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Pseudo-code examples

Violation

The following pseudo-code sample illustrates the pattern detected by this rule. The type of the first parameter of `XPathDocument` is not `XmlReader`.

```
using System.IO;
using System.Xml.XPath;
...
var obj = new XPathDocument(stream);
```

Solution

```
using System.Xml;
using System.Xml.XPath;
...
public void TestMethod(XmlReader reader)
{
    var obj = new XPathDocument(reader);
}
```

CA5373: Do not use obsolete key derivation function

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA5373
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

Cryptographically weak key derivation methods `System.Security.Cryptography.PasswordDeriveBytes` and/or `Rfc2898DeriveBytes.CryptDeriveKey` are used to generate a key.

Rule description

This rule detects the invocation of weak key derivation methods `System.Security.Cryptography.PasswordDeriveBytes` and `Rfc2898DeriveBytes.CryptDeriveKey`. `System.Security.Cryptography.PasswordDeriveBytes` used a weak algorithm PBKDF1. `Rfc2898DeriveBytes.CryptDeriveKey` does not use iteration count and salt from the `Rfc2898DeriveBytes` object, which makes it weak.

How to fix violations

Password-based key derivation should use the PBKDF2 algorithm with SHA-2 hashing. `Rfc2898DeriveBytes.GetBytes` can be used to achieve that.

When to suppress warnings

Suppress the warning if the risk associated with using PBKDF1 is carefully reviewed and accepted.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA5373
// The code that's violating the rule is on this line.
#pragma warning restore CA5373
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA5373.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Pseudo-code examples

Violation

As of the time of this writing, the following pseudo-code sample illustrates the pattern detected by this rule.

```
using System;
using System.Security.Cryptography;
class TestClass
{
    public void TestMethod(Rfc2898DeriveBytes rfc2898DeriveBytes, string aliname, string alghashname, int
keySize, byte[] rgbIV)
    {
        rfc2898DeriveBytes.CryptDeriveKey(aliname, alghashname, keySize, rgbIV);
    }
}
```

Solution

```
using System;
using System.Security.Cryptography;
class TestClass
{
    public void TestMethod(Rfc2898DeriveBytes rfc2898DeriveBytes)
    {
        rfc2898DeriveBytes.GetBytes(1);
    }
}
```

CA5374: Do not use XslTransform

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA5374
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

Instantiating an [System.Xml.Xsl.XslTransform](#), which doesn't restrict potentially dangerous external references or prevent scripts.

Rule description

[XslTransform](#) is vulnerable when operating on untrusted input. An attack could execute arbitrary code.

How to fix violations

Replace [XslTransform](#) with [System.Xml.Xsl.XslCompiledTransform](#). For more guidance, see [/dotnet/standard/data/xml/migrating-from-the-xsltransform-class].

When to suppress warnings

The [XslTransform](#) object, XSLT style sheets, and XML source data are all from trusted sources.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA5374
// The code that's violating the rule is on this line.
#pragma warning restore CA5374
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA5374.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Pseudo-code examples

Violation

At present, the following pseudo-code sample illustrates the pattern detected by this rule.

```
using System;
using System.Xml;
using System.Xml.XPath;
using System.Xml.Xsl;

namespace TestForXslTransform
{
    class Program
    {
        static void Main(string[] args)
        {
            // Create a new XsltTransform object.
            XsltTransform xslt = new XsltTransform();

            // Load the stylesheet.
            xslt.Load("https://server/favorite.xsl");

            // Create a new XPathDocument and load the XML data to be transformed.
            XPathDocument mydata = new XPathDocument("inputdata.xml");

            // Create an XmlTextWriter which outputs to the console.
            XmlWriter writer = new XmlTextWriter(Console.Out);

            // Transform the data and send the output to the console.
            xslt.Transform(mydata, null, writer, null);
        }
    }
}
```

Solution

```
using System.Xml;
using System.Xml.Xsl;

namespace TestForXslTransform
{
    class Program
    {
        static void Main(string[] args)
        {
            // Default XsltSettings constructor disables the XSLT document() function
            // and embedded script blocks.
            XsltSettings settings = new XsltSettings();

            // Execute the transform.
            XslCompiledTransform xslt = new XslCompiledTransform();
            xslt.Load("https://server/favorite.xsl", settings, new XmlUrlResolver());
            xslt.Transform("inputdata.xml", "outputdata.html");
        }
    }
}
```

CA5375: Do not use account shared access signature

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA5375
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

Generating an account Shared Access Signature (SAS) with the `GetSharedAccessSignature` method under the `Microsoft.WindowsAzure.Storage` namespace.

Rule description

An account SAS can delegate access to read, write, and delete operations on blob containers, tables, queues, and file shares that are not permitted with a service SAS. However, it doesn't support container-level policies and has less flexibility and control over the permissions that are granted. If possible, use a service SAS for fine grained access control. For more information, see [Delegate access with a shared access signature](#).

How to fix violations

Use a service SAS instead of an account SAS for fine grained access control and container-level access policy.

When to suppress warnings

It is safe to suppress this rule if you're sure that the permissions of all resources are as restricted as possible.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA5375
// The code that's violating the rule is on this line.
#pragma warning restore CA5375
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA5375.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Pseudo-code examples

Violation

At present, the following pseudo-code sample illustrates the pattern detected by this rule.

```
using System;
using Microsoft.WindowsAzure.Storage;

class ExampleClass
{
    public void ExampleMethod(SharedAccessAccountPolicy policy)
    {
        CloudStorageAccount cloudStorageAccount = new CloudStorageAccount();
        cloudStorageAccount.GetSharedAccessSignature(policy);
    }
}
```

Solution

Instead of account SAS, use service SAS.

```
using System;
using Microsoft.WindowsAzure.Storage;
using Microsoft.WindowsAzure.Storage.File;

class ExampleClass
{
    public void ExampleMethod(StorageCredentials storageCredentials, SharedAccessFilePolicy policy,
SharedAccessFileHeaders headers, string groupPolicyIdentifier, IPAddressOrRange ipAddressOrRange)
    {
        CloudFile cloudFile = new CloudFile(storageCredentials);
        SharedAccessProtocol protocols = SharedAccessProtocol.HttpsOnly;
        cloudFile.GetSharedAccessSignature(policy, headers, groupPolicyIdentifier, protocols,
ipAddressOrRange);
    }
}
```

Related rules

[CA5376: Use SharedAccessProtocol.HttpsOnly](#)

[CA5377: Use container level access policy](#)

CA5376: Use SharedAccessProtocol HttpsOnly

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA5376
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

Using the `GetSharedAccessSignature` method under the `Microsoft.WindowsAzure.Storage` namespace to generate a Shared Access Signature (SAS) with specifying `protocols` as `HttpsOrHttp`.

By default, this rule analyzes the entire codebase, but this is [configurable](#).

Rule description

SAS is a sensitive data which can't be transported in plain text on HTTP.

How to fix violations

Using `HttpsOnly` when generating SAS.

When to suppress warnings

Do not suppress this rule.

Configure code to analyze

Use the following options to configure which parts of your codebase to run this rule on.

- [Exclude specific symbols](#)
- [Exclude specific types and their derived types](#)

You can configure these options for just this rule, for all rules, or for all rules in this category ([Security](#)). For more information, see [Code quality rule configuration options](#).

Exclude specific symbols

You can exclude specific symbols, such as types and methods, from analysis. For example, to specify that the rule should not run on any code within types named `MyType`, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Symbol name only (includes all symbols with the name, regardless of the containing type or namespace).

- Fully qualified names in the symbol's [documentation ID format](#). Each symbol name requires a symbol-kind prefix, such as `M:` for methods, `T:` for types, and `N:` for namespaces.
- `.ctor` for constructors and `.cctor` for static constructors.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType</code>	Matches all symbols named <code>MyType</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType1 MyType2</code>	Matches all symbols named either <code>MyType1</code> or <code>MyType2</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS.MyType.MyMethod(ParamType)</code>	Matches specific method <code>MyMethod</code> with the specified fully qualified signature.
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS1.MyType1.MyMethod1(ParamType) M:NS2.MyType2.MyMethod2(ParamType)</code>	Matches specific methods <code>MyMethod1</code> and <code>MyMethod2</code> with the respective fully qualified signatures.

Exclude specific types and their derived types

You can exclude specific types and their derived types from analysis. For example, to specify that the rule should not run on any methods within types named `MyType` and their derived types, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Type name only (includes all types with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#), with an optional `T:` prefix.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType</code>	Matches all types named <code>MyType</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType1 MyType2</code>	Matches all types named either <code>MyType1</code> or <code>MyType2</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS.MyType</code>	Matches specific type <code>MyType</code> with given fully qualified name and all of its derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS1.MyType1 M:NS2.MyType2</code>	Matches specific types <code>MyType1</code> and <code>MyType2</code> with the respective fully qualified names, and all of their derived types.

Pseudo-code examples

Violation

At present, the following pseudo-code sample illustrates the pattern detected by this rule.

```
using System;
using Microsoft.WindowsAzure.Storage;
using Microsoft.WindowsAzure.Storage.File;

class ExampleClass
{
    public void ExampleMethod(SharedAccessFilePolicy policy, SharedAccessFileHeaders headers, string
groupPolicyIdentifier, IPAddressOrRange ipAddressOrRange)
    {
        CloudFile cloudFile = new CloudFile(null);
        SharedAccessProtocol protocols = SharedAccessProtocol.HttpsOrHttp;
        cloudFile.GetSharedAccessSignature(policy, headers, groupPolicyIdentifier, protocols,
ipAddressOrRange);
    }
}
```

Solution

```
using System;
using Microsoft.WindowsAzure.Storage;
using Microsoft.WindowsAzure.Storage.File;

class ExampleClass
{
    public void ExampleMethod(SharedAccessFilePolicy policy, SharedAccessFileHeaders headers, string
groupPolicyIdentifier, IPAddressOrRange ipAddressOrRange)
    {
        CloudFile cloudFile = new CloudFile(null);
        SharedAccessProtocol protocols = SharedAccessProtocol.HttpsOnly;
        cloudFile.GetSharedAccessSignature(policy, headers, groupPolicyIdentifier, protocols,
ipAddressOrRange);
    }
}
```

Related rules

[CA5375: Do not use account shared access signature](#)

[CA5377: Use container level access policy](#)

CA5377: Use container level access policy

9/20/2022 • 3 minutes to read • [Edit Online](#)

	Value
Rule ID	CA5377
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

Container level policy is not set when generating a service Shared Access Signature (SAS).

By default, this rule analyzes the entire codebase, but this is [configurable](#).

Rule description

A container-level access policy can be modified or revoked at any time. It provides greater flexibility and control over the permissions that are granted. For more information, see [Define a stored access policy](#).

How to fix violations

Specify a valid group policy identifier when generating the service SAS.

When to suppress warnings

It is safe to suppress this rule if you're sure that the permissions of all resources are as restricted as possible.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA5377
// The code that's violating the rule is on this line.
#pragma warning restore CA5377
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA5377.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

Use the following options to configure which parts of your codebase to run this rule on.

- [Exclude specific symbols](#)
- [Exclude specific types and their derived types](#)

You can configure these options for just this rule, for all rules, or for all rules in this category ([Security](#)). For more information, see [Code quality rule configuration options](#).

Exclude specific symbols

You can exclude specific symbols, such as types and methods, from analysis. For example, to specify that the rule should not run on any code within types named `MyType`, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Symbol name only (includes all symbols with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#). Each symbol name requires a symbol-kind prefix, such as `M:` for methods, `T:` for types, and `N:` for namespaces.
- `.ctor` for constructors and `.cctor` for static constructors.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType</code>	Matches all symbols named <code>MyType</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType1 MyType2</code>	Matches all symbols named either <code>MyType1</code> or <code>MyType2</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS.MyType.MyMethod(ParamType)</code>	Matches specific method <code>MyMethod</code> with the specified fully qualified signature.
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS1.MyType1.MyMethod1(ParamType) M:NS2.MyType2.MyMethod2(ParamType)</code>	Matches specific methods <code>MyMethod1</code> and <code>MyMethod2</code> with the respective fully qualified signatures.

Exclude specific types and their derived types

You can exclude specific types and their derived types from analysis. For example, to specify that the rule should not run on any methods within types named `MyType` and their derived types, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Type name only (includes all types with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#), with an optional `T:` prefix.

Examples:

OPTION VALUE	SUMMARY
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType	Matches all types named <code>MyType</code> and all of their derived types.
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType1 MyType2	Matches all types named either <code>MyType1</code> or <code>MyType2</code> and all of their derived types.
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS.MyType	Matches specific type <code>MyType</code> with given fully qualified name and all of its derived types.
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS1.MyType1 M:NS2.MyType2	Matches specific types <code>MyType1</code> and <code>MyType2</code> with the respective fully qualified names, and all of their derived types.

Pseudo-code examples

Violation

At present, the following pseudo-code sample illustrates the pattern detected by this rule.

```
using System;
using Microsoft.WindowsAzure.Storage;
using Microsoft.WindowsAzure.Storage.Blob;

class ExampleClass
{
    public void ExampleMethod(SharedAccessBlobPolicy policy, SharedAccessBlobHeaders headers,
    Nullable<SharedAccessProtocol> protocols, IPAddressOrRange ipAddressOrRange)
    {
        var cloudAppendBlob = new CloudAppendBlob(null);
        string groupPolicyIdentifier = null;
        cloudAppendBlob.GetSharedAccessSignature(policy, headers, groupPolicyIdentifier, protocols,
ipAddressOrRange);
    }
}
```

Solution

```
using System;
using Microsoft.WindowsAzure.Storage;
using Microsoft.WindowsAzure.Storage.Blob;

class ExampleClass
{
    public void ExampleMethod(SharedAccessBlobPolicy policy, SharedAccessBlobHeaders headers,
    Nullable<SharedAccessProtocol> protocols, IPAddressOrRange ipAddressOrRange)
    {
        CloudAppendBlob cloudAppendBlob = new CloudAppendBlob(null);
        string groupPolicyIdentifier = "123";
        cloudAppendBlob.GetSharedAccessSignature(policy, headers, groupPolicyIdentifier, protocols,
ipAddressOrRange);
    }
}
```

Related rules

CA5375: Do not use account shared access signature

CA5376: Use SharedAccessProtocol HttpsOnly

CA5378: Do not disable ServicePointManagerSecurityProtocols

9/20/2022 • 3 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA5378
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

A `AppContext.SetSwitch` method call sets

```
Switch.System.ServiceModel.DisableUsingServicePointManagerSecurityProtocols to true.
```

By default, this rule analyzes the entire codebase, but this is [configurable](#).

Rule description

Setting `Switch.System.ServiceModel.DisableUsingServicePointManagerSecurityProtocols` to `true` limits Windows Communication Framework's (WCF) Transport Layer Security (TLS) connections to using TLS 1.0. That version of TLS will be deprecated. For more information, see [Transport Layer Security \(TLS\) best practices with .NET Framework](#).

How to fix violations

- If your application targets .NET Framework v4.7 or later, you can either remove the `AppContext.SetSwitch` method call, or set the switch's value to `false`.
- If your application targets .NET Framework v4.6.2 or earlier and runs on .NET Framework v4.7 or later, set the switch's value to `false`.
- Otherwise, refer to [Transport Layer Security \(TLS\) best practices with .NET Framework](#) for mitigations.

When to suppress warnings

You can suppress this warning if you need to connect to a legacy service that can't be upgraded to use secure TLS configurations.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA5378
// The code that's violating the rule is on this line.
#pragma warning restore CA5378
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA5378.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

Use the following options to configure which parts of your codebase to run this rule on.

- [Exclude specific symbols](#)
- [Exclude specific types and their derived types](#)

You can configure these options for just this rule, for all rules, or for all rules in this category ([Security](#)). For more information, see [Code quality rule configuration options](#).

Exclude specific symbols

You can exclude specific symbols, such as types and methods, from analysis. For example, to specify that the rule should not run on any code within types named `MyType`, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Symbol name only (includes all symbols with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#). Each symbol name requires a symbol-kind prefix, such as `M:` for methods, `T:` for types, and `N:` for namespaces.
- `.ctor` for constructors and `.cctor` for static constructors.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType</code>	Matches all symbols named <code>MyType</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType1 MyType2</code>	Matches all symbols named either <code>MyType1</code> or <code>MyType2</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS.MyType.MyMethod(ParamType)</code>	Matches specific method <code>MyMethod</code> with the specified fully qualified signature.
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS1.MyType1.MyMethod1(ParamType) M:NS2.MyType2.MyMethod2(ParamType)</code>	Matches specific methods <code>MyMethod1</code> and <code>MyMethod2</code> with the respective fully qualified signatures.

Exclude specific types and their derived types

You can exclude specific types and their derived types from analysis. For example, to specify that the rule should

not run on any methods within types named `MyType` and their derived types, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Type name only (includes all types with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#), with an optional `T:` prefix.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType</code>	Matches all types named <code>MyType</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType1 MyType2</code>	Matches all types named either <code>MyType1</code> or <code>MyType2</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS.MyType</code>	Matches specific type <code>MyType</code> with given fully qualified name and all of its derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS1.MyType1 M:NS2.MyType2</code>	Matches specific types <code>MyType1</code> and <code>MyType2</code> with the respective fully qualified names, and all of their derived types.

Pseudo-code examples

Violation

```
using System;

public class ExampleClass
{
    public void ExampleMethod()
    {
        // CA5378 violation
        ApplicationContext.SetSwitch("Switch.System.ServiceModel.DisableUsingServicePointManagerSecurityProtocols",
true);
    }
}
```

```
Imports System

Public Class ExampleClass
    Public Sub ExampleMethod()
        ' CA5378 violation
        ApplicationContext.SetSwitch("Switch.System.ServiceModel.DisableUsingServicePointManagerSecurityProtocols",
true)
        End Sub
    End Class
```

Solution

```
using System;

public class ExampleClass
{
    public void ExampleMethod()
    {
        ApplicationContext.SetSwitch("Switch.System.ServiceModel.DisableUsingServicePointManagerSecurityProtocols",
false);
    }
}
```

```
Imports System

Public Class ExampleClass
    Public Sub ExampleMethod()
        ApplicationContext.SetSwitch("Switch.System.ServiceModel.DisableUsingServicePointManagerSecurityProtocols",
false)
    End Sub
End Class
```

CA5379: Ensure key derivation function algorithm is sufficiently strong

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA5379
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

Use of one of the following algorithms when instantiating `System.Security.Cryptography.Rfc2898DeriveBytes`:

- `System.Security.Cryptography.MD5`
- `System.Security.Cryptography.SHA1`
- An algorithm that the rule can't determine at compile time

Rule description

The `Rfc2898DeriveBytes` class defaults to using the `SHA1` algorithm. When instantiating an `Rfc2898DeriveBytes` object, you should specify a hash algorithm of `SHA256` or higher. Note that `Rfc2898DeriveBytes.HashAlgorithm` property only has a `get` accessor.

How to fix violations

Because `MD5` or `SHA1` are vulnerable to collisions, use `SHA256` or higher for the `Rfc2898DeriveBytes` class.

Older versions of .NET Framework or .NET Core may not allow you to specify a key derivation function hash algorithm. In such cases, you need to upgrade the target framework version of .NET to use a stronger algorithm.

When to suppress warnings

It is not recommended to suppress this rule except for application compatibility reasons.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA5379
// The code that's violating the rule is on this line.
#pragma warning restore CA5379
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA5379.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Pseudo-code examples

Specify hash algorithm in constructor violation

```
using System.Security.Cryptography;

class ExampleClass
{
    public void ExampleMethod(byte[] password, byte[] salt, int iterations, HashAlgorithmName hashAlgorithm)
    {
        var rfc2898DeriveBytes = new Rfc2898DeriveBytes(password, salt, iterations, HashAlgorithmName.MD5);
    }
}
```

Specify hash algorithm in derived class' constructor violation

```
using System.Security.Cryptography;

class DerivedClass : Rfc2898DeriveBytes
{
    public DerivedClass (byte[] password, byte[] salt, int iterations, HashAlgorithmName hashAlgorithm) :
    base(password, salt, iterations, hashAlgorithm)
    {
    }
}

class ExampleClass
{
    public void ExampleMethod(byte[] password, byte[] salt, int iterations, HashAlgorithmName hashAlgorithm)
    {
        var derivedClass = new DerivedClass(password, salt, iterations, HashAlgorithmName.MD5);
    }
}
```

Set hash algorithm property in derived classes violation

```
using System.Security.Cryptography;

class DerivedClass : Rfc2898DeriveBytes
{
    public DerivedClass (byte[] password, byte[] salt, int iterations, HashAlgorithmName hashAlgorithm) :
    base(password, salt, iterations, hashAlgorithm)
    {
    }

    public HashAlgorithmName HashAlgorithm { get; set; }
}

class ExampleClass
{
    public void ExampleMethod(byte[] password, byte[] salt, int iterations, HashAlgorithmName hashAlgorithm)
    {
        var derivedClass = new DerivedClass(password, salt, iterations, HashAlgorithmName.MD5);
        derivedClass.HashAlgorithm = HashAlgorithmName.SHA256;
    }
}
```

Solution

```
using System.Security.Cryptography;

class ExampleClass
{
    public void ExampleMethod(byte[] password, byte[] salt, int iterations, HashAlgorithmName hashAlgorithm)
    {
        var rfc2898DeriveBytes = new Rfc2898DeriveBytes(password, salt, iterations,
HashAlgorithmName.SHA256);
    }
}
```

CA5380: Do not add certificates to root store

9/20/2022 • 3 minutes to read • [Edit Online](#)

	Value
Rule ID	CA5380
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

Adding certificates to the operating system's trusted root certificates increases the risk of legitimizing untrusted certification authority.

By default, this rule analyzes the entire codebase, but this is [configurable](#).

Rule description

This rule detects code that adds a certificate into the Trusted Root Certification Authorities certificate store. By default, the Trusted Root Certification Authorities certificate store is configured with a set of public CAs that has met the requirements of the Microsoft Root Certificate Program. Since all trusted root certification authorities (CA's) can issue certificates for any domain, an attacker can pick a weak or coercible CA that you install by yourself to target for an attack – and a single vulnerable, malicious or coercible CA undermines the security of the entire system.

How to fix violations

Do not install certificates into the Trusted Root Certification Authorities certificate store.

When to suppress warnings

It is not recommended to suppress this rule.

Configure code to analyze

Use the following options to configure which parts of your codebase to run this rule on.

- [Exclude specific symbols](#)
- [Exclude specific types and their derived types](#)

You can configure these options for just this rule, for all rules, or for all rules in this category ([Security](#)). For more information, see [Code quality rule configuration options](#).

Exclude specific symbols

You can exclude specific symbols, such as types and methods, from analysis. For example, to specify that the rule should not run on any code within types named `MyType`, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Symbol name only (includes all symbols with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#). Each symbol name requires a symbol-kind prefix, such as `M:` for methods, `T:` for types, and `N:` for namespaces.
- `.ctor` for constructors and `.cctor` for static constructors.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType</code>	Matches all symbols named <code>MyType</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType1 MyType2</code>	Matches all symbols named either <code>MyType1</code> or <code>MyType2</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS.MyType.MyMethod(ParamType)</code>	Matches specific method <code>MyMethod</code> with the specified fully qualified signature.
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS1.MyType1.MyMethod1(ParamType) M:NS2.MyType2.MyMethod2(ParamType)</code>	Matches specific methods <code>MyMethod1</code> and <code>MyMethod2</code> with the respective fully qualified signatures.

Exclude specific types and their derived types

You can exclude specific types and their derived types from analysis. For example, to specify that the rule should not run on any methods within types named `MyType` and their derived types, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Type name only (includes all types with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#), with an optional `T:` prefix.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType</code>	Matches all types named <code>MyType</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType1 MyType2</code>	Matches all types named either <code>MyType1</code> or <code>MyType2</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS.MyType</code>	Matches specific type <code>MyType</code> with given fully qualified name and all of its derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS1.MyType1 M:NS2.MyType2</code>	Matches specific types <code>MyType1</code> and <code>MyType2</code> with the respective fully qualified names, and all of their derived types.

Pseudo-code examples

Violation

The following pseudo-code sample illustrates the pattern detected by this rule.

```
using System.Security.Cryptography.X509Certificates;

class TestClass
{
    public void TestMethod()
    {
        var storeName = StoreName.Root;
        var x509Store = new X509Store(storeName);
        x509Store.Add(new X509Certificate2());
    }
}
```

Solution

```
using System.Security.Cryptography.X509Certificates;

class TestClass
{
    public void TestMethod()
    {
        var storeName = StoreName.My;
        var x509Store = new X509Store(storeName);
        x509Store.Add(new X509Certificate2());
    }
}
```

CA5381: Ensure certificates are not added to root store

9/20/2022 • 3 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA5381
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

Adding certificates to the operating system's trusted root certificates increases the risk of legitimizing untrusted certification authority.

By default, this rule analyzes the entire codebase, but this is [configurable](#).

Rule description

This rule detects code that potentially adds a certificate into the Trusted Root Certification Authorities certificate store. By default, the Trusted Root Certification Authorities certificate store is configured with a set of public certification authorities (CAs) that has met the requirements of the Microsoft Root Certificate Program. Since all trusted root CAs can issue certificates for any domain, an attacker can pick a weak or coercible CA that you install by yourself to target for an attack – and a single vulnerable, malicious or coercible CA undermines the security of the entire system.

How to fix violations

Do not install certificates into the Trusted Root Certification Authorities certificate store.

When to suppress warnings

It is not recommended to suppress this rule.

Configure code to analyze

Use the following options to configure which parts of your codebase to run this rule on.

- [Exclude specific symbols](#)
- [Exclude specific types and their derived types](#)

You can configure these options for just this rule, for all rules, or for all rules in this category ([Security](#)). For more information, see [Code quality rule configuration options](#).

Exclude specific symbols

You can exclude specific symbols, such as types and methods, from analysis. For example, to specify that the rule should not run on any code within types named `MyType`, add the following key-value pair to an `.editorconfig` file

in your project:

```
dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Symbol name only (includes all symbols with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#). Each symbol name requires a symbol-kind prefix, such as `M:` for methods, `T:` for types, and `N:` for namespaces.
- `.ctor` for constructors and `.cctor` for static constructors.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType</code>	Matches all symbols named <code>MyType</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType1 MyType2</code>	Matches all symbols named either <code>MyType1</code> or <code>MyType2</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS.MyType.MyMethod(ParamType)</code>	Matches specific method <code>MyMethod</code> with the specified fully qualified signature.
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS1.MyType1.MyMethod1(ParamType) M:NS2.MyType2.MyMethod2(ParamType)</code>	Matches specific methods <code>MyMethod1</code> and <code>MyMethod2</code> with the respective fully qualified signatures.

Exclude specific types and their derived types

You can exclude specific types and their derived types from analysis. For example, to specify that the rule should not run on any methods within types named `MyType` and their derived types, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Type name only (includes all types with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#), with an optional `T:` prefix.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType</code>	Matches all types named <code>MyType</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType1 MyType2</code>	Matches all types named either <code>MyType1</code> or <code>MyType2</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS.MyType</code>	Matches specific type <code>MyType</code> with given fully qualified name and all of its derived types.

OPTION VALUE	SUMMARY
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS1.MyType1 M:NS2.MyType2	Matches specific types <code>MyType1</code> and <code>MyType2</code> with the respective fully qualified names, and all of their derived types.

Pseudo-code examples

Violation

The following pseudo-code sample illustrates the pattern detected by this rule.

```
using System;
using System.Security.Cryptography.X509Certificates;

class TestClass
{
    public void TestMethod()
    {
        var storeName = StoreName.Root;
        Random r = new Random();

        if (r.Next(6) == 4)
        {
            storeName = StoreName.My;
        }

        var x509Store = new X509Store(storeName);
        x509Store.Add(new X509Certificate2());
    }
}
```

Solution

```
using System.Security.Cryptography.X509Certificates;

class TestClass
{
    public void TestMethod()
    {
        var storeName = StoreName.My;
        var x509Store = new X509Store(storeName);
        x509Store.Add(new X509Certificate2());
    }
}
```

CA5382: Use secure cookies in ASP.NET Core

9/20/2022 • 3 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA5382
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

The `Microsoft.AspNetCore.Http.CookieOptions.Secure` property is set as `false` when invoking `Microsoft.AspNetCore.Http.IResponseCookies.Append`. For now, this rule only looks at the `Microsoft.AspNetCore.Http.Internal.ResponseCookies` class, which is one of the implementations of `IResponseCookies`.

This rule is similar to [CA5383](#), but analysis can determine that the `Secure` property is definitely `false` or not set.

By default, this rule analyzes the entire codebase, but this is [configurable](#).

Rule description

Applications available over HTTPS must use secure cookies, which indicate to the browser that the cookie should only be transmitted using Transport Layer Security (TLS).

How to fix violations

Set `Secure` property as `true`.

When to suppress warnings

- If cookies are configured to be secure by default, such as using `Microsoft.AspNetCore.CookiePolicy.CookiePolicyMiddleware` in `Startup.Configure`:

```
public class Startup
{
    public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
    {
        app.UseCookiePolicy(
            new CookiePolicyOptions
            {
                Secure = CookieSecurePolicy.Always
            });
    }
}
```

- If you're sure there's no sensitive data in the cookies.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA5382
// The code that's violating the rule is on this line.
#pragma warning restore CA5382
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA5382.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

Use the following options to configure which parts of your codebase to run this rule on.

- [Exclude specific symbols](#)
- [Exclude specific types and their derived types](#)

You can configure these options for just this rule, for all rules, or for all rules in this category ([Security](#)). For more information, see [Code quality rule configuration options](#).

Exclude specific symbols

You can exclude specific symbols, such as types and methods, from analysis. For example, to specify that the rule should not run on any code within types named `MyType`, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Symbol name only (includes all symbols with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#). Each symbol name requires a symbol-kind prefix, such as `M:` for methods, `T:` for types, and `N:` for namespaces.
- `.ctor` for constructors and `.cctor` for static constructors.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType</code>	Matches all symbols named <code>MyType</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType1 MyType2</code>	Matches all symbols named either <code>MyType1</code> or <code>MyType2</code> .

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS.MyType.MyMethod(ParamType)</code>	Matches specific method <code>MyMethod</code> with the specified fully qualified signature.
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS1.MyType1.MyMethod1(ParamType) M:NS2.MyType2.MyMethod2(ParamType)</code>	Matches specific methods <code>MyMethod1</code> and <code>MyMethod2</code> with the respective fully qualified signatures.

Exclude specific types and their derived types

You can exclude specific types and their derived types from analysis. For example, to specify that the rule should not run on any methods within types named `MyType` and their derived types, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Type name only (includes all types with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#), with an optional `T:` prefix.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType</code>	Matches all types named <code>MyType</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType1 MyType2</code>	Matches all types named either <code>MyType1</code> or <code>MyType2</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS.MyType</code>	Matches specific type <code>MyType</code> with given fully qualified name and all of its derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS1.MyType1 M:NS2.MyType2</code>	Matches specific types <code>MyType1</code> and <code>MyType2</code> with the respective fully qualified names, and all of their derived types.

Example

The following snippet illustrates the pattern detected by this rule.

Violation:

```
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Http.Internal;

class ExampleClass
{
    public void ExampleMethod(string key, string value)
    {
        var cookieOptions = new CookieOptions();
        cookieOptions.Secure = false;
        var responseCookies = new ResponseCookies(null, null);
        responseCookies.Append(key, value, cookieOptions);
    }
}
```

Solution:

```
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Http.Internal;

class ExampleClass
{
    public void ExampleMethod(string key, string value)
    {
        var cookieOptions = new CookieOptions();
        cookieOptions.Secure = true;
        var responseCookies = new ResponseCookies(null, null);
        responseCookies.Append(key, value, cookieOptions);
    }
}
```

CA5383: Ensure use secure cookies in ASP.NET Core

9/20/2022 • 3 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA5383
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

The `Microsoft.AspNetCore.Http.CookieOptions.Secure` property may be set as `false` when invoking `Microsoft.AspNetCore.Http.IResponseCookies.Append`. For now, this rule only looks at the `Microsoft.AspNetCore.Http.Internal.ResponseCookies` class, which is one of the implementations of `IResponseCookies`.

This rule is similar to [CA5382](#), but analysis can't determine that the `Secure` property is definitely `false` or not set.

By default, this rule analyzes the entire codebase, but this is [configurable](#).

Rule description

Applications available over HTTPS must use secure cookies, which indicate to the browser that the cookie should only be transmitted using Transport Layer Security (TLS).

How to fix violations

Set `Secure` property as `true` under all circumstances.

When to suppress warnings

- If cookies are configured to be secure by default, such as using `Microsoft.AspNetCore.CookiePolicy.CookiePolicyMiddleware` in `Startup.Configure`:

```
public class Startup
{
    public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
    {
        app.UseCookiePolicy(
            new CookiePolicyOptions
            {
                Secure = CookieSecurePolicy.Always
            });
    }
}
```

- If you're sure there's no sensitive data in the cookies.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA5383
// The code that's violating the rule is on this line.
#pragma warning restore CA5383
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA5383.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

Use the following options to configure which parts of your codebase to run this rule on.

- [Exclude specific symbols](#)
- [Exclude specific types and their derived types](#)

You can configure these options for just this rule, for all rules, or for all rules in this category ([Security](#)). For more information, see [Code quality rule configuration options](#).

Exclude specific symbols

You can exclude specific symbols, such as types and methods, from analysis. For example, to specify that the rule should not run on any code within types named `MyType`, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Symbol name only (includes all symbols with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#). Each symbol name requires a symbol-kind prefix, such as `M:` for methods, `T:` for types, and `N:` for namespaces.
- `.ctor` for constructors and `.cctor` for static constructors.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType</code>	Matches all symbols named <code>MyType</code> .

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType1 MyType2</code>	Matches all symbols named either <code>MyType1</code> or <code>MyType2</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS.MyType.MyMethod(ParamType)</code>	Matches specific method <code>MyMethod</code> with the specified fully qualified signature.
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS1.MyType1.MyMethod1(ParamType) M:NS2.MyType2.MyMethod2(ParamType)</code>	Matches specific methods <code>MyMethod1</code> and <code>MyMethod2</code> with the respective fully qualified signatures.

Exclude specific types and their derived types

You can exclude specific types and their derived types from analysis. For example, to specify that the rule should not run on any methods within types named `MyType` and their derived types, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Type name only (includes all types with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#), with an optional `T:` prefix.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType</code>	Matches all types named <code>MyType</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType1 MyType2</code>	Matches all types named either <code>MyType1</code> or <code>MyType2</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS.MyType</code>	Matches specific type <code>MyType</code> with given fully qualified name and all of its derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS1.MyType1 M:NS2.MyType2</code>	Matches specific types <code>MyType1</code> and <code>MyType2</code> with the respective fully qualified names, and all of their derived types.

Example

The following snippet illustrates the pattern detected by this rule.

Violation:

```
using System;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Http.Internal;

class ExampleClass
{
    public void ExampleMethod(string key, string value)
    {
        var cookieOptions = new CookieOptions();
        cookieOptions.Secure = false;
        Random r = new Random();

        if (r.Next(6) == 4)
        {
            cookieOptions.Secure = true;
        }

        var responseCookies = new ResponseCookies(null, null);
        responseCookies.Append(key, value, cookieOptions);
    }
}
```

Solution:

```
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Http.Internal;

class ExampleClass
{
    public void ExampleMethod(string key, string value)
    {
        var cookieOptions = new CookieOptions();
        cookieOptions.Secure = true;
        var responseCookies = new ResponseCookies(null, null);
        responseCookies.Append(key, value, cookieOptions);
    }
}
```

CA5384: Do not use digital signature algorithm (DSA)

9/20/2022 • 3 minutes to read • [Edit Online](#)

	Value
Rule ID	CA5384
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

Using DSA in one of the following ways:

- Returning or instantiating derived classes of [System.Security.Cryptography.DSA](#)
- Using [System.Security.Cryptography.AsymmetricAlgorithm.Create](#) or [System.Security.Cryptography.CryptoConfig.CreateFromName](#) to create a DSA object.

By default, this rule analyzes the entire codebase, but this is [configurable](#).

Rule description

DSA is a weak asymmetric encryption algorithm.

How to fix violations

Switch to an RSA with at least 2048 key size, ECDH or ECDsa algorithm instead.

When to suppress warnings

It is not recommended to suppress this rule unless for compatibility with legacy applications and data.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA5384
// The code that's violating the rule is on this line.
#pragma warning restore CA5384
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA5384.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

Use the following options to configure which parts of your codebase to run this rule on.

- [Exclude specific symbols](#)
- [Exclude specific types and their derived types](#)

You can configure these options for just this rule, for all rules, or for all rules in this category ([Security](#)). For more information, see [Code quality rule configuration options](#).

Exclude specific symbols

You can exclude specific symbols, such as types and methods, from analysis. For example, to specify that the rule should not run on any code within types named `MyType`, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Symbol name only (includes all symbols with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#). Each symbol name requires a symbol-kind prefix, such as `M:` for methods, `T:` for types, and `N:` for namespaces.
- `.ctor` for constructors and `.cctor` for static constructors.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType</code>	Matches all symbols named <code>MyType</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType1 MyType2</code>	Matches all symbols named either <code>MyType1</code> or <code>MyType2</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS.MyType.MyMethod(ParamType)</code>	Matches specific method <code>MyMethod</code> with the specified fully qualified signature.
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS1.MyType1.MyMethod1(ParamType) M:NS2.MyType2.MyMethod2(ParamType)</code>	Matches specific methods <code>MyMethod1</code> and <code>MyMethod2</code> with the respective fully qualified signatures.

Exclude specific types and their derived types

You can exclude specific types and their derived types from analysis. For example, to specify that the rule should not run on any methods within types named `MyType` and their derived types, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Type name only (includes all types with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#), with an optional `T:` prefix.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType</code>	Matches all types named <code>MyType</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType1 MyType2</code>	Matches all types named either <code>MyType1</code> or <code>MyType2</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS.MyType</code>	Matches specific type <code>MyType</code> with given fully qualified name and all of its derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS1.MyType1 M:NS2.MyType2</code>	Matches specific types <code>MyType1</code> and <code>MyType2</code> with the respective fully qualified names, and all of their derived types.

Example

The following code snippet illustrates the pattern detected by this rule.

Violation:

```
using System.Security.Cryptography;

class ExampleClass
{
    public void ExampleMethod()
    {
        DSACng dsaCng = new DSACng();
    }
}
```

Solution:

```
using System.Security.Cryptography;

class ExampleClass
{
    public void ExampleMethod()
    {
        AsymmetricAlgorithm asymmetricAlgorithm = AsymmetricAlgorithm.Create("ECDsa");
    }
}
```

CA5385: Use Rivest–Shamir–Adleman (RSA) algorithm with sufficient key size

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA5385
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

Using asymmetric encryption algorithm RSA with key size less than 2048 in one of the following ways:

- Instantiating any descendant classes of `System.Security.Cryptography.RSA` and specifying the `KeySize` parameter as less than 2048.
- Returning any object whose type is descendant of `System.Security.Cryptography.RSA`.
- Using `System.Security.Cryptography.AsymmetricAlgorithm.Create` without parameter which would create RSA with the default key size 1024.
- Using `System.Security.Cryptography.AsymmetricAlgorithm.Create` and specifying the `a1gName` parameter as `RSA` with the default key size 1024.
- Using `System.Security.Cryptography.CryptoConfig.CreateFromName` and specifying the `name` parameter as `RSA` with the default key size 1024.
- Using `System.Security.Cryptography.CryptoConfig.CreateFromName` and specifying the `name` parameter as `RSA` and specifying the key size as smaller than 2048 explicitly by `args`.

Rule description

An RSA key smaller than 2048 bits is more vulnerable to brute force attacks.

How to fix violations

Switch to an RSA with at least 2048 key size, ECDH or ECDsa algorithm instead.

When to suppress warnings

It is not recommended to suppress this rule unless for compatibility with legacy applications and data.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA5385
// The code that's violating the rule is on this line.
#pragma warning restore CA5385
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA5385.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Example

The following code snippet illustrates the pattern detected by this rule.

Violation:

```
using System.Security.Cryptography;

class ExampleClass
{
    public void ExampleMethod()
    {
        RSACng rsaCng = new RSACng(1024);
    }
}
```

Solution:

```
using System.Security.Cryptography;

class ExampleClass
{
    public void ExampleMethod()
    {
        RSACng rsaCng = new RSACng(2048);
    }
}
```

CA5386: Avoid hardcoding SecurityProtocolType value

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA5386
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

This rule fires when either of the following conditions are met:

- A safe but hardcoded `System.Net.SecurityProtocolType` value was referenced.
- An integer value representing a safe protocol version was assigned to a `SecurityProtocolType` variable.

Safe values are:

- `Tls12`
- `Tls13`

Rule description

Transport Layer Security (TLS) secures communication between computers, most commonly with Hypertext Transfer Protocol Secure (HTTPS). Protocol versions TLS 1.0 and TLS 1.1 are deprecated, while TLS 1.2 and TLS 1.3 are current. In the future, TLS 1.2 and TLS 1.3 may be deprecated. To ensure that your application remains secure, avoid hardcoding a protocol version and target at least .NET Framework v4.7.1. For more information, see [Transport Layer Security \(TLS\) best practices with .NET Framework](#).

How to fix violations

Don't hardcode TLS protocol versions.

When to suppress warnings

You can suppress this warning if your application targets .NET Framework v4.6.2 or earlier and may run on a computer that has insecure defaults.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA5386
// The code that's violating the rule is on this line.
#pragma warning restore CA5386
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA5386.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Pseudo-code examples

Enumeration name violation

```
using System;
using System.Net;

public class ExampleClass
{
    public void ExampleMethod()
    {
        // CA5386 violation
        ServicePointManager.SecurityProtocol = SecurityProtocolType.Tls12;
    }
}
```

```
Imports System
Imports System.Net

Public Class TestClass
    Public Sub ExampleMethod()
        ' CA5386 violation
        ServicePointManager.SecurityProtocol = SecurityProtocolType.Tls12
    End Sub
End Class
```

Integer value violation

```
using System;
using System.Net;

public class ExampleClass
{
    public void ExampleMethod()
    {
        // CA5386 violation
        ServicePointManager.SecurityProtocol = (SecurityProtocolType) 3072;      // TLS 1.2
    }
}
```

```
Imports System
Imports System.Net

Public Class TestClass
    Public Sub ExampleMethod()
        ' CA5386 violation
        ServicePointManager.SecurityProtocol = CType(3072, SecurityProtocolType)      ' TLS 1.2
    End Sub
End Class
```

Solution

```
using System;
using System.Net;

public class TestClass
{
    public void TestMethod()
    {
        // Let the operating system decide what TLS protocol version to use.
        // See https://docs.microsoft.com/dotnet/framework/network-programming/tls
    }
}
```

```
Imports System
Imports System.Net

Public Class TestClass
    Public Sub ExampleMethod()
        ' Let the operating system decide what TLS protocol version to use.
        ' See https://docs.microsoft.com/dotnet/framework/network-programming/tls
    End Sub
End Class
```

Related rules

[CA5364: Do not use deprecated security protocols](#)

[CA5397: Do not use deprecated SslProtocols values](#)

[CA5398: Avoid hardcoded SslProtocols values](#)

CA5387: Do not use weak key derivation function with insufficient iteration count

9/20/2022 • 3 minutes to read • [Edit Online](#)

	Value
Rule ID	CA5387
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

Using `System.Security.Cryptography.Rfc2898DeriveBytes.GetBytes` with the default iteration count or specifying an iteration count of less than 100,000.

By default, this rule analyzes the entire codebase, but this is [configurable](#).

Rule description

This rule checks if a cryptographic key was generated by `Rfc2898DeriveBytes` with an iteration count of less than 100,000. A higher iteration count can help mitigate against dictionary attacks that try to guess the generated cryptographic key.

This rule is similar to [CA5388](#), but analysis determines that the iteration count is less than 100,000.

How to fix violations

Set the iteration count greater than or equal with 100,000 before calling `GetBytes`.

When to suppress warnings

It's safe to suppress a warning if you need to use a smaller iteration count for compatibility with existing data.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA5387
// The code that's violating the rule is on this line.
#pragma warning restore CA5387
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA5387.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

Use the following options to configure which parts of your codebase to run this rule on.

- [Exclude specific symbols](#)
- [Exclude specific types and their derived types](#)

You can configure these options for just this rule, for all rules, or for all rules in this category ([Security](#)). For more information, see [Code quality rule configuration options](#).

Exclude specific symbols

You can exclude specific symbols, such as types and methods, from analysis. For example, to specify that the rule should not run on any code within types named `MyType`, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Symbol name only (includes all symbols with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#). Each symbol name requires a symbol-kind prefix, such as `M:` for methods, `T:` for types, and `N:` for namespaces.
- `.ctor` for constructors and `.cctor` for static constructors.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType</code>	Matches all symbols named <code>MyType</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType1 MyType2</code>	Matches all symbols named either <code>MyType1</code> or <code>MyType2</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS.MyType.MyMethod(ParamType)</code>	Matches specific method <code>MyMethod</code> with the specified fully qualified signature.
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS1.MyType1.MyMethod1(ParamType) M:NS2.MyType2.MyMethod2(ParamType)</code>	Matches specific methods <code>MyMethod1</code> and <code>MyMethod2</code> with the respective fully qualified signatures.

Exclude specific types and their derived types

You can exclude specific types and their derived types from analysis. For example, to specify that the rule should not run on any methods within types named `MyType` and their derived types, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Type name only (includes all types with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#), with an optional `T:` prefix.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType</code>	Matches all types named <code>MyType</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType1 MyType2</code>	Matches all types named either <code>MyType1</code> or <code>MyType2</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS.MyType</code>	Matches specific type <code>MyType</code> with given fully qualified name and all of its derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS1.MyType1 M:NS2.MyType2</code>	Matches specific types <code>MyType1</code> and <code>MyType2</code> with the respective fully qualified names, and all of their derived types.

Pseudo-code examples

Default iteration count violation

```
using System.Security.Cryptography;

class ExampleClass
{
    public void ExampleMethod(string password, byte[] salt, int cb)
    {
        var rfc2898DeriveBytes = new Rfc2898DeriveBytes(password, salt);
        rfc2898DeriveBytes.GetBytes(cb);
    }
}
```

Specify iteration count in constructor violation

```
using System.Security.Cryptography;

class ExampleClass
{
    public void ExampleMethod(string password, byte[] salt, int cb)
    {
        var rfc2898DeriveBytes = new Rfc2898DeriveBytes(password, salt, 100);
        rfc2898DeriveBytes.GetBytes(cb);
    }
}
```

Specify iteration count by property assignment violation

```
using System.Security.Cryptography;

class ExampleClass
{
    public void ExampleMethod(string password, byte[] salt, int cb)
    {
        var rfc2898DeriveBytes = new Rfc2898DeriveBytes(password, salt);
        rfc2898DeriveBytes.IterationCount = 100;
        rfc2898DeriveBytes.GetBytes(cb);
    }
}
```

Solution

```
using System.Security.Cryptography;

class ExampleClass
{
    public void ExampleMethod(string password, byte[] salt, int cb)
    {
        var rfc2898DeriveBytes = new Rfc2898DeriveBytes(password, salt);
        rfc2898DeriveBytes.IterationCount = 100000;
        rfc2898DeriveBytes.GetBytes(cb);
    }
}
```

CA5388: Ensure sufficient iteration count when using weak key derivation function

9/20/2022 • 3 minutes to read • [Edit Online](#)

	Value
Rule ID	CA5388
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

Iteration count may be smaller than 100,000 when deriving cryptographic key by [System.Security.Cryptography.Rfc2898DeriveBytes.GetBytes](#).

By default, this rule analyzes the entire codebase, but this is [configurable](#).

Rule description

This rule checks if a cryptographic key was generated by [Rfc2898DeriveBytes](#) with an iteration count that may be less than 100,000. A higher iteration count can help mitigate against dictionary attacks that try to guess the generated cryptographic key.

This rule is similar to [CA5387](#), but analysis can't determine if the iteration count is less than 100,000.

How to fix violations

Set the iteration count greater than or equal with 100k before calling [GetBytes](#) explicitly.

When to suppress warnings

It's safe to suppress warnings from this rule if:

- You need to use a smaller iteration count for compatibility with existing data.
- You're sure that the iteration count is set above 100,000.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA5388
// The code that's violating the rule is on this line.
#pragma warning restore CA5388
```

To disable the rule for a file, folder, or project, set its severity to [none](#) in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA5388.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

Use the following options to configure which parts of your codebase to run this rule on.

- [Exclude specific symbols](#)
- [Exclude specific types and their derived types](#)

You can configure these options for just this rule, for all rules, or for all rules in this category ([Security](#)). For more information, see [Code quality rule configuration options](#).

Exclude specific symbols

You can exclude specific symbols, such as types and methods, from analysis. For example, to specify that the rule should not run on any code within types named `MyType`, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Symbol name only (includes all symbols with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#). Each symbol name requires a symbol-kind prefix, such as `M:` for methods, `T:` for types, and `N:` for namespaces.
- `.ctor` for constructors and `.cctor` for static constructors.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType</code>	Matches all symbols named <code>MyType</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType1 MyType2</code>	Matches all symbols named either <code>MyType1</code> or <code>MyType2</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS.MyType.MyMethod(ParamType)</code>	Matches specific method <code>MyMethod</code> with the specified fully qualified signature.
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS1.MyType1.MyMethod1(ParamType) M:NS2.MyType2.MyMethod2(ParamType)</code>	Matches specific methods <code>MyMethod1</code> and <code>MyMethod2</code> with the respective fully qualified signatures.

Exclude specific types and their derived types

You can exclude specific types and their derived types from analysis. For example, to specify that the rule should not run on any methods within types named `MyType` and their derived types, add the following key-value pair

to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Type name only (includes all types with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#), with an optional `T:` prefix.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType</code>	Matches all types named <code>MyType</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType1 MyType2</code>	Matches all types named either <code>MyType1</code> or <code>MyType2</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS.MyType</code>	Matches specific type <code>MyType</code> with given fully qualified name and all of its derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS1.MyType1 M:NS2.MyType2</code>	Matches specific types <code>MyType1</code> and <code>MyType2</code> with the respective fully qualified names, and all of their derived types.

Pseudo-code examples

Violation

```
using System;
using System.Security.Cryptography;

class ExampleClass
{
    public void ExampleMethod(string password, byte[] salt, int cb)
    {
        var iterations = 100;
        Random r = new Random();

        if (r.Next(6) == 4)
        {
            iterations = 100000;
        }

        var rfc2898DeriveBytes = new Rfc2898DeriveBytes(password, salt, iterations);
        rfc2898DeriveBytes.GetBytes(cb);
    }
}
```

Solution

```
using System.Security.Cryptography;

class ExampleClass
{
    public void ExampleMethod(string password, byte[] salt, int cb)
    {
        var rfc2898DeriveBytes = new Rfc2898DeriveBytes(password, salt);
        rfc2898DeriveBytes.IterationCount = 100000;
        rfc2898DeriveBytes.GetBytes(cb);
    }
}
```

CA5389: Do not add archive item's path to the target file system path

9/20/2022 • 4 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA5389
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

A non-sanitized source file path is used as the target file path in one of these parameters:

- parameter `destinationFileName` of method `ZipFileExtensions.ExtractToFile`
- parameter `path` of method `File.Open`
- parameter `path` of method `File.OpenWrite`
- parameter `path` of method `File.Create`
- parameter `path` of constructor for `FileStream`
- parameter `fileName` of constructor for `FileInfo`

By default, this rule analyzes the entire codebase, but this is [configurable](#).

Rule description

File path can be relative and can lead to file system access outside of the expected file system target path, leading to malicious config changes and remote code execution via lay-and-wait technique.

How to fix violations

Do not use the source file path to construct the target file path, or make sure that the last character on the extraction path is the directory separator character.

When to suppress warnings

You can suppress this warning if the source path always comes from a trusted source.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA5389
// The code that's violating the rule is on this line.
#pragma warning restore CA5389
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA5389.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

Use the following options to configure which parts of your codebase to run this rule on.

- [Exclude specific symbols](#)
- [Exclude specific types and their derived types](#)

You can configure these options for just this rule, for all rules, or for all rules in this category ([Security](#)). For more information, see [Code quality rule configuration options](#).

Exclude specific symbols

You can exclude specific symbols, such as types and methods, from analysis. For example, to specify that the rule should not run on any code within types named `MyType`, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Symbol name only (includes all symbols with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#). Each symbol name requires a symbol-kind prefix, such as `M:` for methods, `T:` for types, and `N:` for namespaces.
- `.ctor` for constructors and `.cctor` for static constructors.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType</code>	Matches all symbols named <code>MyType</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType1 MyType2</code>	Matches all symbols named either <code>MyType1</code> or <code>MyType2</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS.MyType.MyMethod(ParamType)</code>	Matches specific method <code>MyMethod</code> with the specified fully qualified signature.
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS1.MyType1.MyMethod1(ParamType) M:NS2.MyType2.MyMethod2(ParamType)</code>	Matches specific methods <code>MyMethod1</code> and <code>MyMethod2</code> with the respective fully qualified signatures.

Exclude specific types and their derived types

You can exclude specific types and their derived types from analysis. For example, to specify that the rule should

not run on any methods within types named `MyType` and their derived types, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Type name only (includes all types with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#), with an optional `T:` prefix.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType</code>	Matches all types named <code>MyType</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType1 MyType2</code>	Matches all types named either <code>MyType1</code> or <code>MyType2</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS.MyType</code>	Matches specific type <code>MyType</code> with given fully qualified name and all of its derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS1.MyType1 M:NS2.MyType2</code>	Matches specific types <code>MyType1</code> and <code>MyType2</code> with the respective fully qualified names, and all of their derived types.

Example

The following code snippet illustrates the pattern detected by this rule.

Violation:

```
using System.IO.Compression;

class TestClass
{
    public void TestMethod(ZipArchiveEntry zipArchiveEntry)
    {
        zipArchiveEntry.ExtractToFile(zipArchiveEntry.FullName);
    }
}
```

Solution:

```
using System;
using System.IO;
using System.IO.Compression;

class Program
{
    static void Main(string[] args)
    {
        string zipPath = @".\result.zip";

        Console.WriteLine("Provide path where to extract the zip file:");
        string extractPath = Console.ReadLine();

        // Normalizes the path.
        extractPath = Path.GetFullPath(extractPath);

        // Ensures that the last character on the extraction path
        // is the directory separator char.
        // Without this, a malicious zip file could try to traverse outside of the expected
        // extraction path.
        if (!extractPath.EndsWith(Path.DirectorySeparatorChar.ToString(), StringComparison.OrdinalIgnoreCase))
            extractPath += Path.DirectorySeparatorChar;

        using (ZipArchive archive = ZipFile.OpenRead(zipPath))
        {
            foreach (ZipArchiveEntry entry in archive.Entries)
            {
                if (entry.FullName.EndsWith(".txt", StringComparison.OrdinalIgnoreCase))
                {
                    // Gets the full path to ensure that relative segments are removed.
                    string destinationPath = Path.GetFullPath(Path.Combine(extractPath, entry.FullName));

                    // Ordinal match is safest, case-sensitive volumes can be mounted within volumes that
                    // are case-insensitive.
                    if (destinationPath.StartsWith(extractPath, StringComparison.OrdinalIgnoreCase))
                        entry.ExtractToFile(destinationPath);
                }
            }
        }
    }
}
```

CA5390: Do not hard-code encryption key

9/20/2022 • 3 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA5390
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

The `key` parameter of the `System.Security.Cryptography.AesCcm` or `System.Security.Cryptography.AesGcm` constructor, `System.Security.Cryptography.SymmetricAlgorithm.Key` property, or the `rgbKey` parameter of the `System.Security.Cryptography.SymmetricAlgorithm.CreateEncryptor` or `System.Security.Cryptography.SymmetricAlgorithm.CreateDecryptor` method is hard-coded by one of the following:

- Byte array.
- `System.Convert.FromBase64String`.
- All the overloads of `System.Text.Encoding.GetBytes`.

By default, this rule analyzes the entire codebase, but this is [configurable](#).

Rule description

For a symmetric algorithm to be successful, the secret key must be known only to the sender and the receiver. When a key is hard-coded, it is easily discovered. Even with compiled binaries, it is easy for malicious users to extract it. Once the private key is compromised, the cipher text can be decrypted directly and is not protected anymore.

How to fix violations

- Consider redesigning your application to use a secure key management system, such as Azure Key Vault.
- Keep credentials and keys in a secure location separate from your source code.

When to suppress warnings

Do not suppress a warning from this rule.

Configure code to analyze

Use the following options to configure which parts of your codebase to run this rule on.

- [Exclude specific symbols](#)
- [Exclude specific types and their derived types](#)

You can configure these options for just this rule, for all rules, or for all rules in this category ([Security](#)). For more information, see [Code quality rule configuration options](#).

Exclude specific symbols

You can exclude specific symbols, such as types and methods, from analysis. For example, to specify that the rule should not run on any code within types named `MyType`, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Symbol name only (includes all symbols with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#). Each symbol name requires a symbol-kind prefix, such as `M:` for methods, `T:` for types, and `N:` for namespaces.
- `.ctor` for constructors and `.cctor` for static constructors.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType</code>	Matches all symbols named <code>MyType</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType1 MyType2</code>	Matches all symbols named either <code>MyType1</code> or <code>MyType2</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS.MyType.MyMethod(ParamType)</code>	Matches specific method <code>MyMethod</code> with the specified fully qualified signature.
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS1.MyType1.MyMethod1(ParamType) M:NS2.MyType2.MyMethod2(ParamType)</code>	Matches specific methods <code>MyMethod1</code> and <code>MyMethod2</code> with the respective fully qualified signatures.

Exclude specific types and their derived types

You can exclude specific types and their derived types from analysis. For example, to specify that the rule should not run on any methods within types named `MyType` and their derived types, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Type name only (includes all types with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#), with an optional `T:` prefix.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType</code>	Matches all types named <code>MyType</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType1 MyType2</code>	Matches all types named either <code>MyType1</code> or <code>MyType2</code> and all of their derived types.

OPTION VALUE	SUMMARY
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS.MyType	Matches specific type <code>MyType</code> with given fully qualified name and all of its derived types.
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS1.MyType1 M:NS2.MyType2	Matches specific types <code>MyType1</code> and <code>MyType2</code> with the respective fully qualified names, and all of their derived types.

Pseudo-code examples

Hard-coded byte array violation

```
using System;
using System.Security.Cryptography;

class ExampleClass
{
    public void ExampleMethod(byte[] someOtherBytesForIV)
    {
        byte[] rgbKey = new byte[] {1, 2, 3};
        SymmetricAlgorithm rijndael = SymmetricAlgorithm.Create();
        rijndael.CreateEncryptor(rgbKey, someOtherBytesForIV);
    }
}
```

Hard-coded Convert.FromBase64String violation

```
using System;
using System.Security.Cryptography;

class ExampleClass
{
    public void ExampleMethod(byte[] someOtherBytesForIV)
    {
        byte[] key = Convert.FromBase64String("AAAAAaazaoensuth");
        SymmetricAlgorithm rijndael = SymmetricAlgorithm.Create();
        rijndael.CreateEncryptor(key, someOtherBytesForIV);
    }
}
```

Hard-coded Encoding.GetBytes violation

```
using System.Text;
using System.Security.Cryptography;

class ExampleClass
{
    public void ExampleMethod(byte[] someOtherBytesForIV)
    {
        byte[] key = Encoding.ASCII.GetBytes("AAAAAaazaoensuth");
        SymmetricAlgorithm rijndael = SymmetricAlgorithm.Create();
        rijndael.CreateEncryptor(key, someOtherBytesForIV);
    }
}
```

Solution

```
using System.Text;
using System.Security.Cryptography;

class ExampleClass
{
    public void ExampleMethod(char[] chars, byte[] someOtherBytesForIV)
    {
        byte[] key = Encoding.ASCII.GetBytes(chars);
        SymmetricAlgorithm rijndael = SymmetricAlgorithm.Create();
        rijndael.CreateEncryptor(key, someOtherBytesForIV);
    }
}
```

CA5391: Use antiforgery tokens in ASP.NET Core MVC controllers

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA5391
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

Actions that result in modifying operations don't have an antiforgery token attribute. Or, using a global antiforgery token filter without calling expected anti forgery token functions.

Rule description

Handling a `POST`, `PUT`, `PATCH`, or `DELETE` request without validating an antiforgery token may be vulnerable to cross-site request forgery attacks. A cross-site request forgery attack can send malicious requests from an authenticated user to your ASP.NET Core MVC controller.

How to fix violations

- Mark the modifying action with a valid antiforgery token attribute:
 - `Microsoft.AspNetCore.Mvc.ValidateAntiForgeryTokenAttribute`.
 - Attribute whose name is like `%Validate%Anti_forgery%Attribute`.
- Add the valid forgery token attribute into the global filter with `Microsoft.AspNetCore.Mvc.Filters.FilterCollection.Add`.
- Add any custom or Mvc-provided antiforgery filter class that calls `Validate` on any class that implements the `Microsoft.AspNetCore.Antiforgery.IAntiforgery` interface.

When to suppress warnings

It's safe to suppress this rule if solutions other than using antiforgery token attributes are adopted to mitigate CSRF vulnerabilities. For more information, see [Prevent Cross-Site Request Forgery \(XSRF/CSRF\) attacks in ASP.NET Core](#).

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA5391
// The code that's violating the rule is on this line.
#pragma warning restore CA5391
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA5391.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

You can configure whether the rule applies only to derived classes of `Microsoft.AspNetCore.Mvc.Controller` in your codebase. For example, to specify that the rule should not run on any code within derived types of `ControllerBase`, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CA5391.exclude_aspnet_core_mvc_controllerbase = true
```

Pseudo-code examples

Without anti forgery token attribute violation

```
using Microsoft.AspNetCore.Mvc;

class ExampleController : Controller
{
    [HttpDelete]
    public IActionResult ExampleAction (string actionPerformed)
    {
        return null;
    }

    [ValidateAntiForgeryToken]
    [HttpDelete]
    public IActionResult AnotherAction (string actionPerformed)
    {
        return null;
    }
}
```

Without valid global anti forgery filter

```

using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Filters;

class ExampleController : Controller
{
    [ValidateAntiForgeryToken]
    [HttpDelete]
    public IActionResult AnotherAction (string actionName)
    {
        return null;
    }

    [HttpDelete]
    public IActionResult ExampleAction (string actionName)
    {
        return null;
    }
}

class FilterClass : IAsyncAuthorizationFilter
{
    public Task OnAuthorizationAsync (AuthorizationFilterContext context)
    {
        return null;
    }
}

class BlahClass
{
    public static void BlahMethod ()
    {
        FilterCollection filterCollection = new FilterCollection ();
        filterCollection.Add(typeof(FilterClass));
    }
}

```

Marked with an anti forgery token attribute solution

```

using Microsoft.AspNetCore.Mvc;

class ExampleController : Controller
{
    [ValidateAntiForgeryToken]
    [HttpDelete]
    public IActionResult ExampleAction ()
    {
        return null;
    }

    [ValidateAntiForgeryToken]
    [HttpDelete]
    public IActionResult AnotherAction ()
    {
        return null;
    }
}

```

Using a valid global anti forgery filter

```
using System.Threading.Tasks;
using Microsoft.AspNetCore.Antiforgery;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Filters;

class ExampleController : Controller
{
    [ValidateAntiForgeryToken]
    [HttpDelete]
    public IActionResult AnotherAction()
    {
        return null;
    }

    [HttpDelete]
    public IActionResult ExampleAction()
    {
        return null;
    }
}

class FilterClass : IAsyncAuthorizationFilter
{
    private readonly IAntiforgery antiforgery;

    public FilterClass(IAntiforgery antiforgery)
    {
        this.antiforgery = antiforgery;
    }

    public Task OnAuthorizationAsync(AuthorizationFilterContext context)
    {
        return antiforgery.ValidateRequestAsync(context.HttpContext);
    }
}

class BlahClass
{
    public static void BlahMethod()
    {
        FilterCollection filterCollection = new FilterCollection();
        filterCollection.Add(typeof(FilterClass));
    }
}
```

CA5392: Use DefaultDllImportSearchPaths attribute for P/Invokes

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA5392
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

The [DefaultDllImportSearchPathsAttribute](#) is not specified for a [Platform Invoke \(P/Invoke\)](#) function.

Rule description

By default, P/Invoke functions using [DllImportAttribute](#) probe a number of directories, including the current working directory for the library to load. This can be a security issue for certain applications, leading to DLL hijacking.

For example, if a malicious DLL with the same name as the imported one is placed under the current working directory, which will be searched firstly by default, then the malicious DLL could be loaded.

For more information, see [Load Library Safely](#).

How to fix violations

Use [DefaultDllImportSearchPathsAttribute](#) to specify the DLL search path explicitly for the assembly or the method.

When to suppress warnings

It's safe to suppress this rule if:

- You're sure the loaded assembly is what you want. For example, your application runs on a trusted server and you completely trust the files.
- The imported assembly is a commonly used system assembly, like user32.dll, and the search path strategy follows the [Known DLLs mechanism](#).

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA5392
// The code that's violating the rule is on this line.
#pragma warning restore CA5392
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA5392.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Pseudo-code examples

```
using System;
using System.Runtime.InteropServices;

class ExampleClass
{
    [DllImport("The3rdAssembly.dll")]
    public static extern int MessageBox(IntPtr hWnd, String text, String caption, uint type);

    public void ExampleMethod()
    {
        MessageBox(new IntPtr(0), "Hello World!", "Hello Dialog", 0);
    }
}
```

Solution

```
using System;
using System.Runtime.InteropServices;

class ExampleClass
{
    [DllImport("The3rdAssembly.dll")]
    [DefaultDllImportSearchPaths(DllImportSearchPath.UserDirectories)]
    public static extern int MessageBox(IntPtr hWnd, String text, String caption, uint type);

    public void ExampleMethod()
    {
        MessageBox(new IntPtr(0), "Hello World!", "Hello Dialog", 0);
    }
}
```

Related rules

[CA5393: Do not use unsafe DllImportSearchPath value](#)

CA5393: Do not use unsafe DllImportSearchPath value

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA5393
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

Using one of the unsafe values of `<xref:System.Runtime.InteropServices.DllImportSearchPath? displayProperty=fullName>`:

- `AssemblyDirectory`
- `UseDllDirectoryForDependencies`
- `ApplicationDirectory`
- `LegacyBehavior`

Rule description

There could be a malicious DLL in the default DLL search directories and assembly directories. Or, depending on where your application is run from, there could be a malicious DLL in the application's directory.

For more information, see [Load Library Safely](#).

How to fix violations

Use safe values of `DllImportSearchPath` to specify an explicit search path instead:

- `SafeDirectories`
- `System32`
- `UserDirectories`

When to suppress warnings

It's safe to suppress this rule if:

- You're sure the loaded assembly is what you want.
- The imported assembly is a commonly used system assembly, like `user32.dll`, and the search path strategy follows the [Known DLLs mechanism](#).

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA5393
// The code that's violating the rule is on this line.
#pragma warning restore CA5393
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.cs,vb]
dotnet_diagnostic.CA5393.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.cs,vb]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

Use the following option to configure which parts of your codebase to run this rule on.

- [Unsafe DllImportSearchPath bits](#)

You can configure this option for just this rule, for all rules, or for all rules in this category ([Security](#)). For more information, see [Code quality rule configuration options](#).

Unsafe DllImportSearchPath bits

You can configure which value of [DllImportSearchPath](#) is unsafe for the analysis. For example, to specify that the code should not use `AssemblyDirectory`, `UseDllImportSearchPaths(DllImportSearchPath.AssemblyDirectory)` or `ApplicationDirectory`, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CA5393.unsafe_DllImportSearchPath_bits = 770
```

You should specify the integer value of a bitwise combination of the enumeration's values.

Pseudo-code examples

```
using System;
using System.Runtime.InteropServices;

class ExampleClass
{
    [DllImport("The3rdAssembly.dll")]
    [DefaultDllImportSearchPaths(DllImportSearchPath.AssemblyDirectory)]
    public static extern int MessageBox(IntPtr hWnd, String text, String caption, uint type);

    public void ExampleMethod()
    {
        MessageBox(new IntPtr(0), "Hello World!", "Hello Dialog", 0);
    }
}
```

Solution

```
using System;
using System.Runtime.InteropServices;

class ExampleClass
{
    [DllImport("The3rdAssembly.dll")]
    [DefaultDllImportSearchPaths(DefaultDllImportSearchPath.UserDirectories)]
    public static extern int MessageBox(IntPtr hWnd, String text, String caption, uint type);

    public void ExampleMethod()
    {
        MessageBox(new IntPtr(0), "Hello World!", "Hello Dialog", 0);
    }
}
```

Related rules

[CA5392: Use DefaultDllImportSearchPaths attribute for P/Invokes](#)

CA5394: Do not use insecure randomness

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA5394
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

One of the methods of `System.Random` is invoked.

Rule description

Using a cryptographically weak pseudo-random number generator may allow an attacker to predict what security-sensitive value will be generated.

How to fix violations

If you need an unpredictable value for security, use a cryptographically strong random number generator like `System.Security.Cryptography.RandomNumberGenerator` or `System.Security.Cryptography.RNGCryptoServiceProvider`.

When to suppress warnings

It's safe to suppress warnings from this rule if you're sure that the weak pseudo-random numbers aren't used in a security-sensitive manner.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA5394
// The code that's violating the rule is on this line.
#pragma warning restore CA5394
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA5394.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Pseudo-code examples

Violation

```
using System;

class ExampleClass
{
    public void ExampleMethod(Random random)
    {
        var sensitiveVariable = random.Next();
    }
}
```

Solution

```
using System;
using System.Security.Cryptography;

class ExampleClass
{
    public void ExampleMethod(int toExclusive)
    {
        var sensitiveVariable = RandomNumberGenerator.GetInt32(toExclusive);
    }
}
```

CA5395: Miss HttpVerb attribute for action methods

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA5395
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

Not specifying the kind of HTTP request explicitly for action methods.

Rule description

All the action methods that create, edit, delete, or otherwise modify data needs to be protected with the antiforgery attribute from cross-site request forgery attacks. Performing a GET operation should be a safe operation that has no side effects and doesn't modify your persisted data.

How to fix violations

Mark the action methods with `Httpverb` attribute.

When to suppress warnings

It's safe to suppress warnings from this rule if:

- You're sure that no modifying operation is taking place in the action method. Or, it's not an action method at all.
- Solutions other than using antiforgery token attributes are adopted to mitigate CSRF vulnerabilities. For more information, see [Prevent Cross-Site Request Forgery \(XSRF/CSRF\) attacks in ASP.NET Core](#).

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA5395
// The code that's violating the rule is on this line.
#pragma warning restore CA5395
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA5395.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Pseudo-code examples

Violation

```
using Microsoft.AspNetCore.Mvc;

[ValidateAntiForgeryToken]
class BlahController : Controller
{
}

class ExampleController : Controller
{
    public IActionResult ExampleAction()
    {
        return null;
    }
}
```

Solution

```
using Microsoft.AspNetCore.Mvc;

[ValidateAntiForgeryToken]
class BlahController : Controller
{
}

class ExampleController : Controller
{
    [HttpGet]
    public IActionResult ExampleAction()
    {
        return null;
    }
}
```

CA5396: Set HttpOnly to true for HttpCookie

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA5396
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

`System.Web.HttpCookie.HttpOnly` is set to `false`. The default value of this property is `false`.

Rule description

As a defense in depth measure, ensure security sensitive HTTP cookies are marked as `HttpOnly`. This indicates web browsers should disallow scripts from accessing the cookies. Injected malicious scripts are a common way of stealing cookies.

How to fix violations

Set `System.Web.HttpCookie.HttpOnly` to `true`.

When to suppress warnings

- If the global value of `HttpOnly` is set, such as in the following example:

```
<system.web>
  ...
  <httpCookies httpOnlyCookies="true" requireSSL="true" />
</system.web>
```

- If you're sure there's no sensitive data in the cookies.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA5396
// The code that's violating the rule is on this line.
#pragma warning restore CA5396
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA5396.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Example

Violation:

```
using System.Web;

class ExampleClass
{
    public void ExampleMethod()
    {
        HttpCookie httpCookie = new HttpCookie("cookieName");
        httpCookie.HttpOnly = false;
    }
}
```

Solution:

```
using System.Web;

class ExampleClass
{
    public void ExampleMethod()
    {
        HttpCookie httpCookie = new HttpCookie("cookieName");
        httpCookie.HttpOnly = true;
    }
}
```

CA5397: Do not use deprecated SslProtocols values

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA5397
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

This rule fires when either of the following conditions are met:

- A deprecated `System.Security.Authentication.SslProtocols` value was referenced.
- An integer value representing a deprecated value was either assigned to a `SslProtocols` variable, used as a `SslProtocols` return value, or used as a `SslProtocols` argument.

Deprecated values are:

- Ssl2
- Ssl3
- Tls
- Tls10
- Tls11

Rule description

Transport Layer Security (TLS) secures communication between computers, most commonly with Hypertext Transfer Protocol Secure (HTTPS). Older protocol versions of TLS are less secure than TLS 1.2 and TLS 1.3 and are more likely to have new vulnerabilities. Avoid older protocol versions to minimize risk. For guidance on identifying and removing deprecated protocol versions, see [Solving the TLS 1.0 Problem, 2nd Edition](#).

How to fix violations

Don't use deprecated TLS protocol versions.

When to suppress warnings

You can suppress this warning if:

- The reference to the deprecated protocol version isn't being used to enable a deprecated version.
- You need to connect to a legacy service that can't be upgraded to use secure TLS configurations.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA5397
// The code that's violating the rule is on this line.
#pragma warning restore CA5397
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA5397.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Pseudo-code examples

Enumeration name violation

```
using System;
using System.Security.Authentication;

public class ExampleClass
{
    public void ExampleMethod()
    {
        // CA5397 violation for using Tls11
        SslProtocols protocols = SslProtocols.Tls11 | SslProtocols.Tls12;
    }
}
```

```
Imports System
Imports System.Security.Authentication

Public Class TestClass
    Public Sub ExampleMethod()
        ' CA5397 violation for using Tls11
        Dim sslProtocols As SslProtocols = SslProtocols.Tls11 Or SslProtocols.Tls12
    End Sub
End Class
```

Integer value violation

```
using System;
using System.Security.Authentication;

public class ExampleClass
{
    public void ExampleMethod()
    {
        // CA5397 violation
        SslProtocols sslProtocols = (SslProtocols) 768;      // TLS 1.1
    }
}
```

```

Imports System
Imports System.Security.Authentication

Public Class TestClass
    Public Sub ExampleMethod()
        ' CA5397 violation
        Dim sslProtocols As SslProtocols = CType(768, SslProtocols)      ' TLS 1.1
    End Sub
End Class

```

Solution

```

using System;
using System.Security.Authentication;

public class TestClass
{
    public void Method()
    {
        // Let the operating system decide what TLS protocol version to use.
        // See https://docs.microsoft.com/dotnet/framework/network-programming/tls
        SslProtocols sslProtocols = SslProtocols.None;
    }
}

```

```

Imports System
Imports System.Security.Authentication

Public Class TestClass
    Public Sub ExampleMethod()
        ' Let the operating system decide what TLS protocol version to use.
        ' See https://docs.microsoft.com/dotnet/framework/network-programming/tls
        Dim sslProtocols As SslProtocols = SslProtocols.None
    End Sub
End Class

```

Related rules

[CA5364: Do not use deprecated security protocols](#)

[CA5386: Avoid hardcoding SecurityProtocolType value](#)

[CA5398: Avoid hardcoded SslProtocols values](#)

CA5398: Avoid hardcoded SslProtocols values

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA5398
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

This rule fires when either of the following conditions are met:

- A safe but hardcoded `System.Security.Authentication.SslProtocols` value was referenced.
- An integer value representing a safe protocol version was either assigned to a `SslProtocols` variable, used as a `SslProtocols` return value, or used as a `SslProtocols` argument.

Safe values are:

- `Tls12`
- `Tls13`

Rule description

Transport Layer Security (TLS) secures communication between computers, most commonly with Hypertext Transfer Protocol Secure (HTTPS). Protocol versions TLS 1.0 and TLS 1.1 are deprecated, while TLS 1.2 and TLS 1.3 are current. In the future, TLS 1.2 and TLS 1.3 may be deprecated. To ensure that your application remains secure, avoid hardcoding a protocol version. For more information, see [Transport Layer Security \(TLS\) best practices with .NET Framework](#).

How to fix violations

Don't hardcode TLS protocol versions.

When to suppress warnings

It's safe to suppress a warning if you need to connect to a legacy service that can't be upgraded to use future TLS protocol versions.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA5398
// The code that's violating the rule is on this line.
#pragma warning restore CA5398
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA5398.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Pseudo-code examples

Enumeration name violation

```
using System;
using System.Security.Authentication;

public class ExampleClass
{
    public void ExampleMethod()
    {
        // CA5398 violation
        SslProtocols sslProtocols = SslProtocols.Tls12;
    }
}
```

```
Imports System
Imports System.Security.Authentication

Public Class TestClass
    Public Function ExampleMethod() As SslProtocols
        ' CA5398 violation
        Return SslProtocols.Tls12
    End Function
End Class
```

Integer value violation

```
using System;
using System.Security.Authentication;

public class ExampleClass
{
    public SslProtocols ExampleMethod()
    {
        // CA5398 violation
        return (SslProtocols) 3072;      // TLS 1.2
    }
}
```

```

Imports System
Imports System.Security.Authentication

Public Class TestClass
    Public Function ExampleMethod() As SslProtocols
        ' CA5398 violation
        Return CType(3072, SslProtocols) ' TLS 1.2
    End Function
End Class

```

Solution

```

using System;
using System.Security.Authentication;

public class TestClass
{
    public void Method()
    {
        // Let the operating system decide what TLS protocol version to use.
        // See https://docs.microsoft.com/dotnet/framework/network-programming/tls
        SslProtocols sslProtocols = SslProtocols.None;
    }
}

```

```

Imports System
Imports System.Security.Authentication

Public Class TestClass
    Public Sub ExampleMethod()
        ' Let the operating system decide what TLS protocol version to use.
        ' See https://docs.microsoft.com/dotnet/framework/network-programming/tls
        Dim sslProtocols As SslProtocols = SslProtocols.None
    End Sub
End Class

```

Related rules

[CA5364: Do not use deprecated security protocols](#)

[CA5386: Avoid hardcoding SecurityProtocolType value](#)

[CA5397: Do not use deprecated SslProtocols values](#)

CA5399: Enable HttpClient certificate revocation list check

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA5399
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

Using `System.Net.Http.HttpClient` while providing a platform specific handler (`System.Net.Http.WinHttpHandler` or `System.Net.Http.HttpClientHandler`) whose `CheckCertificateRevocationList` property is not set to `true` will allow revoked certificates to be accepted by the `HttpClient` as valid.

This rule is similar to [CA5400](#), but analysis can determine that the `CheckCertificateRevocationList` property is definitely `false` or not set.

Rule description

A revoked certificate isn't trusted anymore. It could be used by attackers passing some malicious data or stealing sensitive data in HTTPS communication.

How to fix violations

Set the `System.Net.Http.HttpClientHandler.CheckCertificateRevocationList` property to `true` explicitly. If the `CheckCertificateRevocationList` property is unavailable, you need to upgrade your target framework.

When to suppress warnings

Do not suppress this rule.

Configure code to analyze

Use the following options to configure which parts of your codebase to run this rule on.

- [Exclude specific symbols](#)
- [Exclude specific types and their derived types](#)

You can configure these options for just this rule, for all rules, or for all rules in this category ([Security](#)). For more information, see [Code quality rule configuration options](#).

Exclude specific symbols

You can exclude specific symbols, such as types and methods, from analysis. For example, to specify that the rule should not run on any code within types named `MyType`, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Symbol name only (includes all symbols with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#). Each symbol name requires a symbol-kind prefix, such as `M:` for methods, `T:` for types, and `N:` for namespaces.
- `.ctor` for constructors and `.cctor` for static constructors.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType</code>	Matches all symbols named <code>MyType</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType1 MyType2</code>	Matches all symbols named either <code>MyType1</code> or <code>MyType2</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS.MyType.MyMethod(ParamType)</code>	Matches specific method <code>MyMethod</code> with the specified fully qualified signature.
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS1.MyType1.MyMethod1(ParamType) M:NS2.MyType2.MyMethod2(ParamType)</code>	Matches specific methods <code>MyMethod1</code> and <code>MyMethod2</code> with the respective fully qualified signatures.

Exclude specific types and their derived types

You can exclude specific types and their derived types from analysis. For example, to specify that the rule should not run on any methods within types named `MyType` and their derived types, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Type name only (includes all types with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#), with an optional `T:` prefix.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType</code>	Matches all types named <code>MyType</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType1 MyType2</code>	Matches all types named either <code>MyType1</code> or <code>MyType2</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS.MyType</code>	Matches specific type <code>MyType</code> with given fully qualified name and all of its derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS1.MyType1 M:NS2.MyType2</code>	Matches specific types <code>MyType1</code> and <code>MyType2</code> with the respective fully qualified names, and all of their derived types.

Pseudo-code examples

```
using System.Net.Http;

class ExampleClass
{
    void ExampleMethod()
    {
        WinHttpHandler winHttpHandler = new WinHttpHandler();
        winHttpHandler.CheckCertificateRevocationList = false;
        HttpClient httpClient = new HttpClient(winHttpHandler);
    }
}
```

Solution

```
using System.Net.Http;

class ExampleClass
{
    void ExampleMethod()
    {
        WinHttpHandler winHttpHandler = new WinHttpHandler();
        winHttpHandler.CheckCertificateRevocationList = true;
        HttpClient httpClient = new HttpClient(winHttpHandler);
    }
}
```

CA5400: Ensure HttpClient certificate revocation list check is not disabled

9/20/2022 • 3 minutes to read • [Edit Online](#)

	Value
Rule ID	CA5400
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

Using `System.Net.Http.HttpClient` while providing a platform specific handler (`System.Net.Http.WinHttpHandler` or `System.Net.Http.HttpClientHandler`) whose `CheckCertificateRevocationList` property is possibly set to `false` will allow revoked certificates to be accepted by the `HttpClient` as valid.

This rule is similar to [CA5399](#), but analysis can't determine that the `CheckCertificateRevocationList` property is definitely `false` or not set.

Rule description

A revoked certificate isn't trusted anymore. It could be used by attackers passing some malicious data or stealing sensitive data in HTTPS communication.

How to fix violations

Set the `System.Net.Http.HttpClientHandler.CheckCertificateRevocationList` property to `true` explicitly. If the `CheckCertificateRevocationList` property is unavailable, you need to upgrade your target framework.

When to suppress warnings

It's safe to suppress this rule if you're sure that the `CheckCertificateRevocationList` property is set correctly.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA5400
// The code that's violating the rule is on this line.
#pragma warning restore CA5400
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA5400.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

Use the following options to configure which parts of your codebase to run this rule on.

- [Exclude specific symbols](#)
- [Exclude specific types and their derived types](#)

You can configure this option for just this rule, for all rules, or for all rules in this category ([Security](#)). For more information, see [Code quality rule configuration options](#).

Exclude specific symbols

You can exclude specific symbols, such as types and methods, from analysis. For example, to specify that the rule should not run on any code within types named `MyType`, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Symbol name only (includes all symbols with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#). Each symbol name requires a symbol-kind prefix, such as `M:` for methods, `T:` for types, and `N:` for namespaces.
- `.ctor` for constructors and `.cctor` for static constructors.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType</code>	Matches all symbols named <code>MyType</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = MyType1 MyType2</code>	Matches all symbols named either <code>MyType1</code> or <code>MyType2</code> .
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS.MyType.MyMethod(ParamType)</code>	Matches specific method <code>MyMethod</code> with the specified fully qualified signature.
<code>dotnet_code_quality.CAXXXX.excluded_symbol_names = M:NS1.MyType1.MyMethod1(ParamType) M:NS2.MyType2.MyMethod2(ParamType)</code>	Matches specific methods <code>MyMethod1</code> and <code>MyMethod2</code> with the respective fully qualified signatures.

Exclude specific types and their derived types

You can exclude specific types and their derived types from analysis. For example, to specify that the rule should not run on any methods within types named `MyType` and their derived types, add the following key-value pair

to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType
```

Allowed symbol name formats in the option value (separated by `|`):

- Type name only (includes all types with the name, regardless of the containing type or namespace).
- Fully qualified names in the symbol's [documentation ID format](#), with an optional `T:` prefix.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType</code>	Matches all types named <code>MyType</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = MyType1 MyType2</code>	Matches all types named either <code>MyType1</code> or <code>MyType2</code> and all of their derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS.MyType</code>	Matches specific type <code>MyType</code> with given fully qualified name and all of its derived types.
<code>dotnet_code_quality.CAXXXX.excluded_type_names_with_derived_types = M:NS1.MyType1 M:NS2.MyType2</code>	Matches specific types <code>MyType1</code> and <code>MyType2</code> with the respective fully qualified names, and all of their derived types.

Pseudo-code examples

```
using System;
using System.Net.Http;

class ExampleClass
{
    void ExampleMethod(bool checkCertificateRevocationList)
    {
        WinHttpHandler winHttpHandler = new WinHttpHandler();
        winHttpHandler.CheckCertificateRevocationList = checkCertificateRevocationList;
        Random r = new Random();

        if (r.Next(6) == 4)
        {
            winHttpHandler.CheckCertificateRevocationList = true;
        }

        HttpClient httpClient = new HttpClient(winHttpHandler);
    }
}
```

Solution

```
using System.Net.Http;

class ExampleClass
{
    void ExampleMethod()
    {
        WinHttpHandler winHttpHandler = new WinHttpHandler();
        winHttpHandler.CheckCertificateRevocationList = true;
        HttpClient httpClient = new HttpClient(winHttpHandler);
    }
}
```

CA5401: Do not use CreateEncryptor with non-default IV

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA5401
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

Using `System.Security.Cryptography.SymmetricAlgorithm.CreateEncryptor` with non-default `rgbIV`.

Rule description

Symmetric encryption should always use a non-repeatable initialization vector to prevent dictionary attacks.

This rule is similar to [CA5402](#), but analysis determines that the initialization vector is definitely the default.

How to fix violations

Use the default `rgbIV` value, that is, use the overload of the

`System.Security.Cryptography.SymmetricAlgorithm.CreateEncryptor` which doesn't have any parameter.

When to suppress warnings

It's safe to suppress a warning from this rule if:

- The `rgbIV` parameter was generated by `System.Security.Cryptography.SymmetricAlgorithm.GenerateIV`.
- You're sure that the `rgbIV` is really random and non-repeatable.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA5401
// The code that's violating the rule is on this line.
#pragma warning restore CA5401
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA5401.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Pseudo-code examples

```
using System.Security.Cryptography;

class ExampleClass
{
    public void ExampleMethod(byte[] rgbIV)
    {
        AesCng aesCng = new AesCng();
        aesCng.IV = rgbIV;
        aesCng.CreateEncryptor();
    }
}
```

Solution

```
using System.Security.Cryptography;

class ExampleClass
{
    public void ExampleMethod()
    {
        AesCng aesCng = new AesCng();
        aesCng.CreateEncryptor();
    }
}
```

CA5402: Use CreateEncryptor with the default IV

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA5402
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

The `rgbIV` could be non-default when using `System.Security.Cryptography.SymmetricAlgorithm.CreateEncryptor`.

Rule description

Symmetric encryption should always use a non-repeatable initialization vector to prevent dictionary attacks.

This rule is similar to [CA5401](#), but analysis can't determine that the initialization vector is definitely the default.

How to fix violations

Use the default `rgbIV` value explicitly, that is, use the overload of the `System.Security.Cryptography.SymmetricAlgorithm.CreateEncryptor` which doesn't have any parameter.

When to suppress warnings

It's safe to suppress a warning from this rule if:

- The `rgbIV` parameter was generated by `System.Security.Cryptography.SymmetricAlgorithm.GenerateIV`.
- You're sure that the `rgbIV` parameter is really random and non-repeatable.
- You're sure that the initialization vector is used.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA5402
// The code that's violating the rule is on this line.
#pragma warning restore CA5402
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA5402.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Pseudo-code examples

```
using System;
using System.Security.Cryptography;

class ExampleClass
{
    public void ExampleMethod(byte[] rgbIV)
    {
        AesCng aesCng = new AesCng();
        Random r = new Random();

        if (r.Next(6) == 4)
        {
            aesCng.IV = rgbIV;
        }

        aesCng.CreateEncryptor();
    }
}
```

Solution

```
using System.Security.Cryptography;

class ExampleClass
{
    public void ExampleMethod()
    {
        AesCng aesCng = new AesCng();
        aesCng.CreateEncryptor();
    }
}
```

CA5403: Do not hard-code certificate

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA5403
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

The `data` or `rawData` parameter of a `X509Certificate` or `X509Certificate2` constructor is hard-coded by one of the following:

- Byte array.
- Char array.
- `System.Convert.FromBase64String(String)`.
- All the overloads of `System.Text.Encoding.GetBytes`.

Rule description

A hard-coded certificate's private key is easily discovered. Even with compiled binaries, it is easy for malicious users to extract a hard-coded certificate's private key. Once the private key is compromised, an attacker can impersonate that certificate, and any resources or operations protected by that certificate will be available to the attacker.

How to fix violations

- Consider redesigning your application to use a secure key management system, such as Azure Key Vault.
- Keep credentials and certificates in a secure location separate from your source code.

When to suppress warnings

It's safe to suppress a warning from this rule if the hard-coded data doesn't contain the certificate's private key. For example, the data is from a `.cer` file. Hard-coding public certificate information may still create a challenge for rotating certificates as they expire or get revoked.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA5403
// The code that's violating the rule is on this line.
#pragma warning restore CA5403
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA5403.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Pseudo-code examples

Hard-coded by byte array

```
using System.IO;
using System.Security.Cryptography.X509Certificates;

class ExampleClass
{
    public void ExampleMethod(string path)
    {
        byte[] bytes = new byte[] {1, 2, 3};
        File.WriteAllBytes(path, bytes);
        new X509Certificate2(path);
    }
}
```

Hard-coded by char array

```
using System.IO;
using System.Security.Cryptography.X509Certificates;
using System.Text;

class ExampleClass
{
    public void ExampleMethod(byte[] bytes, string path)
    {
        char[] chars = new char[] { '1', '2', '3' };
        Encoding.ASCII.GetBytes(chars, 0, 3, bytes, 0);
        File.WriteAllBytes(path, bytes);
        new X509Certificate2(path);
    }
}
```

Hard-coded by FromBase64String

```
using System;
using System.IO;
using System.Security.Cryptography.X509Certificates;

class ExampleClass
{
    public void ExampleMethod(string path)
    {
        byte[] bytes = Convert.FromBase64String("AAAAAAazaoensuth");
        File.WriteAllBytes(path, bytes);
        new X509Certificate2(path);
    }
}
```

Hard-coded by GetBytes

```
using System;
using System.IO;
using System.Security.Cryptography.X509Certificates;
using System.Text;

class ExampleClass
{
    public void ExampleMethod(string path)
    {
        byte[] bytes = Encoding.ASCII.GetBytes("AAAAAAazaoensuth");
        File.WriteAllBytes(path, bytes);
        new X509Certificate2(path);
    }
}
```

Solution

```
using System.IO;
using System.Security.Cryptography.X509Certificates;

class ExampleClass
{
    public void ExampleMethod(string path)
    {
        new X509Certificate2("Certificate.cer");
    }
}
```

CA5404: Do not disable token validation checks

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA5404
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

Setting `TokenValidationParameters` properties `RequireExpirationTime`, `ValidateAudience`, `ValidateIssuer`, or `ValidateLifetime` to `false`.

Rule description

Token validation checks ensure that while validating tokens, all aspects are analyzed and verified. Turning off validation can lead to security holes by allowing untrusted tokens to make it through validation.

More details about best practices for token validation can be found on the [library's wiki](#).

How to fix violations

Set `TokenValidationParameters` properties `RequireExpirationTime`, `ValidateAudience`, `ValidateIssuer`, or `ValidateLifetime` to `true`. Or, remove the assignment to `false` because the default value is `true`.

When to suppress warnings

In the vast majority of cases, this validation is essential to ensure the security of the consuming app. However, there are some cases where this validation is not needed, especially in non-standard token types. Before you disable this validation, be sure you have fully thought through the security implications. For information about the trade-offs, see the [token validation library's wiki](#).

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA5404
// The code that's violating the rule is on this line.
#pragma warning restore CA5404
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA5404.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Pseudo-code examples

```
using System;
using Microsoft.IdentityModel.Tokens;

class TestClass
{
    public void TestMethod()
    {
        TokenValidationParameters parameters = new TokenValidationParameters();
        parameters.RequireExpirationTime = false;
        parameters.ValidateAudience = false;
        parameters.ValidateIssuer = false;
        parameters.ValidateLifetime = false;
    }
}
```

Solution

```
using System;
using Microsoft.IdentityModel.Tokens;

class TestClass
{
    public void TestMethod()
    {
        TokenValidationParameters parameters = new TokenValidationParameters();
        parameters.RequireExpirationTime = true;
        parameters.ValidateAudience = true;
        parameters.ValidateIssuer = true;
        parameters.ValidateLifetime = true;
    }
}
```

CA5405: Do not always skip token validation in delegates

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA5405
Category	Security
Fix is breaking or non-breaking	Non-breaking

Cause

The callback assigned to [AudienceValidator](#) or [LifetimeValidator](#) always returns `true`.

Rule description

By setting critical `TokenValidationParameter` validation delegates to always return `true`, important authentication safeguards are disabled. Disabling safeguards can lead to incorrect validation of tokens from any issuer or expired tokens.

For more information about best practices for token validation, see the [library's wiki](#).

How to fix violations

- Improve the logic of the delegate so not all code paths return `true`, which effectively disables that type of validation.
- Throw `SecurityTokenInvalidAudienceException` or `SecurityTokenInvalidLifetimeException` in failure cases when you want to fail validation and have other cases pass by returning `true`.

When to suppress warnings

In some specific cases where you're utilizing the delegate for additional logging and it's for token types where the specific type of validation is not needed, it may make sense to suppress this warning. Before you disable this validation, be sure you have fully thought through the security implications. For information about the trade-offs, see the [token validation library's wiki](#).

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA5405
// The code that's violating the rule is on this line.
#pragma warning restore CA5405
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA5405.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Security.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Pseudo-code examples

Violation

```
using System;
using Microsoft.IdentityModel.Tokens;

class TestClass
{
    public void TestMethod()
    {
        TokenValidationParameters parameters = new TokenValidationParameters();
        parameters.AudienceValidator = (audiences, token, tvp) => { return true; };
    }
}
```

Solution

```
using System;
using Microsoft.IdentityModel.Tokens;

class TestClass
{
    public void TestMethod()
    {
        TokenValidationParameters parameters = new TokenValidationParameters();
        parameters.AudienceValidator = (audiences, token, tvp) =>
        {
            // Implement your own custom audience validation
            if (PerformCustomAudienceValidation(audiences, token))
                return true;
            else
                return false;
        };
    }
}
```

Usage rules

9/20/2022 • 6 minutes to read • [Edit Online](#)

Usage rules support proper usage of .NET.

In this section

RULE	DESCRIPTION
CA1801: Review unused parameters	A method signature includes a parameter that is not used in the method body.
CA1816: Call GC.SuppressFinalize correctly	A method that is an implementation of Dispose does not call <code>GC.SuppressFinalize</code> ; or a method that is not an implementation of <code>Dispose</code> calls <code>GC.SuppressFinalize</code> ; or a method calls <code>GC.SuppressFinalize</code> and passes something other than <code>this</code> (<code>Me</code> in Visual Basic).
CA2200: Rethrow to preserve stack details	An exception is rethrown and the exception is explicitly specified in the throw statement. If an exception is rethrown by specifying the exception in the throw statement, the list of method calls between the original method that threw the exception and the current method is lost.
CA2201: Do not raise reserved exception types	This makes the original error hard to detect and debug.
CA2207: Initialize value type static fields inline	A value type declares an explicit static constructor. To fix a violation of this rule, initialize all static data when it is declared and remove the static constructor.
CA2208: Instantiate argument exceptions correctly	A call is made to the default (parameterless) constructor of an exception type that is or derives from <code>ArgumentException</code> , or an incorrect string argument is passed to a parameterized constructor of an exception type that is or derives from <code>ArgumentException</code> .
CA2211: Non-constant fields should not be visible	Static fields that are not constants or read-only are not thread-safe. Access to such a field must be carefully controlled and requires advanced programming techniques for synchronizing access to the class object.
CA2213: Disposable fields should be disposed	A type that implements <code>System.IDisposable</code> declares fields that are of types that also implement <code>IDisposable</code> . The <code>Dispose</code> method of the field is not called by the <code>Dispose</code> method of the declaring type.
CA2214: Do not call overridable methods in constructors	When a constructor calls a virtual method, it is possible that the constructor for the instance that invokes the method has not executed.
CA2215: Dispose methods should call base class dispose	If a type inherits from a disposable type, it must call the <code>Dispose</code> method of the base type from its own <code>Dispose</code> method.

RULE	DESCRIPTION
CA2216: Disposable types should declare finalizer	A type that implements System.IDisposable , and has fields that suggest the use of unmanaged resources, does not implement a finalizer as described by <code>Object.Finalize</code> .
CA2217: Do not mark enums with FlagsAttribute	An externally visible enumeration is marked with <code>FlagsAttribute</code> , and it has one or more values that are not powers of two or a combination of the other defined values on the enumeration.
CA2218: Override GetHashCode on overriding Equals	A public type overrides System.Object.Equals but does not override System.Object.GetHashCode .
CA2219: Do not raise exceptions in exception clauses	When an exception is raised in a finally or fault clause, the new exception hides the active exception. When an exception is raised in a filter clause, the run time silently catches the exception. This makes the original error hard to detect and debug.
CA2224: Override equals on overloading operator equals	A public type implements the equality operator but doesn't override System.Object.Equals .
CA2225: Operator overloads have named alternates	An operator overload was detected, and the expected named alternative method was not found. The named alternative member provides access to the same functionality as the operator, and is provided for developers who program in languages that do not support overloaded operators.
CA2226: Operators should have symmetrical overloads	A type implements the equality or inequality operator, and does not implement the opposite operator.
CA2227: Collection properties should be read only	A writable collection property allows a user to replace the collection with a different collection. A read-only property stops the collection from being replaced but still allows the individual members to be set.
CA2229: Implement serialization constructors	To fix a violation of this rule, implement the serialization constructor. For a sealed class, make the constructor private; otherwise, make it protected.
CA2231: Overload operator equals on overriding ValueType.Equals	A value type overrides <code>Object.Equals</code> but does not implement the equality operator.
CA2234: Pass System.Uri objects instead of strings	A call is made to a method that has a string parameter whose name contains "uri", "URI", "urn", "URN", "url", or "URL". The declaring type of the method contains a corresponding method overload that has a System.Uri parameter.
CA2235: Mark all non-serializable fields	An instance field of a type that is not serializable is declared in a type that is serializable.

RULE	DESCRIPTION
CA2237: Mark ISerializable types with SerializableAttribute	To be recognized by the common language runtime as serializable, types must be marked with the <code>SerializableAttribute</code> attribute even if the type uses a custom serialization routine through implementation of the <code>ISerializable</code> interface.
CA2241: Provide correct arguments to formatting methods	The format argument passed to <code>String.Format</code> does not contain a format item that corresponds to each object argument, or vice versa.
CA2242: Test for NaN correctly	This expression tests a value against <code>Single.Nan</code> or <code>Double.Nan</code> . Use <code>Single.IsNaN(Single)</code> or <code>Double.IsNaN(Double)</code> to test the value.
CA2243: Attribute string literals should parse correctly	An attribute's string literal parameter does not parse correctly for a URL, a GUID, or a version.
CA2244: Do not duplicate indexed element initializations	An object initializer has more than one indexed element initializer with the same constant index. All but the last initializer are redundant.
CA2245: Do not assign a property to itself	A property was accidentally assigned to itself.
CA2246: Do not assign a symbol and its member in the same statement	Assigning a symbol and its member, that is, a field or a property, in the same statement is not recommended. It is not clear if the member access was intended to use the symbol's old value prior to the assignment or the new value from the assignment in this statement.
CA2247: Argument passed to TaskCompletionSource constructor should be TaskCreationOptions enum instead of TaskContinuationOptions enum	TaskCompletionSource has constructors that take <code>TaskCreationOptions</code> that control the underlying <code>Task</code> , and constructors that take object state that's stored in the task. Accidentally passing a <code>TaskContinuationOptions</code> instead of a <code>TaskCreationOptions</code> will result in the call treating the options as state.
CA2248: Provide correct 'enum' argument to 'Enum.HasFlag'	The enum type passed as an argument to the <code>HasFlag</code> method call is different from the calling enum type.
CA2249: Consider using String.Contains instead of String.IndexOf	Calls to <code>string.IndexOf</code> where the result is used to check for the presence or absence of a substring can be replaced by <code>string.Contains</code> .
CA2250: Use <code>ThrowIfCancellationRequested</code>	<code>ThrowIfCancellationRequested</code> automatically checks whether the token has been canceled, and throws an <code>OperationCanceledException</code> if it has.
CA2251: Use <code>string.Equals</code> over <code>string.Compare</code>	It is both clearer and likely faster to use <code>String.Equals</code> instead of comparing the result of <code>String.Compare</code> to zero.
CA2252: Opt in to preview features	Opt in to preview features before using preview APIs.
CA2253: Named placeholders should not be numeric values	Named placeholders in the logging message template should not be comprised of only numeric characters.

RULE	DESCRIPTION
CA2254: Template should be a static expression	The logging message template should not vary between calls.
CA2255: The <code>ModuleInitializer</code> attribute should not be used in libraries	Module initializers are intended to be used by application code to ensure an application's components are initialized before the application code begins executing.
CA2256: All members declared in parent interfaces must have an implementation in a <code>DynamicInterfaceCastableImplementation-attributed</code> interface	Types attributed with <code>DynamicInterfaceCastableImplementationAttribute</code> act as an interface implementation for a type that implements the <code>IDynamicInterfaceCastable</code> type. As a result, it must provide an implementation of all of the members defined in the inherited interfaces, because the type that implements <code>IDynamicInterfaceCastable</code> will not provide them otherwise.
CA2257: Members defined on an interface with 'DynamicInterfaceCastableImplementationAttribute' should be 'static'	Since a type that implements <code>IDynamicInterfaceCastable</code> may not implement a dynamic interface in metadata, calls to an instance interface member that is not an explicit implementation defined on this type are likely to fail at run time. Mark new interface members <code>static</code> to avoid run-time errors.
CA2258: Providing a 'DynamicInterfaceCastableImplementation' interface in Visual Basic is unsupported	Providing a functional <code>DynamicInterfaceCastableImplementationAttribute</code> -attributed interface requires the Default Interface Members feature, which is unsupported in Visual Basic.

CA1801: Review unused parameters

9/20/2022 • 3 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA1801
Category	Usage
Fix is breaking or non-breaking	<p>Non-breaking - If the member is not visible outside the assembly, regardless of the change you make.</p> <p>Non-breaking - If you change the member to use the parameter within its body.</p> <p>Breaking - If you remove the parameter and it is visible outside the assembly.</p>

Cause

A method signature includes a parameter that's not used in the method body.

This rule does not examine the following kinds of methods:

- Methods referenced by a delegate.
- Methods used as event handlers.
- Serialization constructors (see [guidelines](#)).
- Serialization [GetObjectData](#) methods.
- Methods declared with the `abstract` (`MustOverride` in Visual Basic) modifier.
- Methods declared with the `virtual` (`Overridable` in Visual Basic) modifier.
- Methods declared with the `override` (`Overrides` in Visual Basic) modifier.
- Methods declared with the `extern` (`Declare` statement in Visual Basic) modifier.

This rule does not flag parameters that are named with the [discard](#) symbol, for example, `_`, `_1`, and `_2`. This reduces warning noise on parameters that are needed for signature requirements, for example, a method used as a delegate, a parameter with special attributes, or a parameter whose value is implicitly accessed at run time by a framework but is not referenced in code.

NOTE

This rule has been deprecated in favor of [IDE0060](#). For information about how to enforce the IDE0060 analyzer at build, see [code-style analysis](#).

Rule description

Review parameters in non-virtual methods that are not used in the method body to make sure no incorrectness

exists around failure to access them. Unused parameters incur maintenance and performance costs.

Sometimes, a violation of this rule can point to an implementation bug in the method. For example, the parameter should have been used in the method body. Suppress warnings of this rule if the parameter must exist because of backward compatibility.

How to fix violations

To fix a violation of this rule, remove the unused parameter (a breaking change), or use the parameter in the method body (a non-breaking change).

When to suppress warnings

It is safe to suppress a warning from this rule:

- In previously shipped code for which the fix would be a breaking change.
- For the `this` parameter in a custom extension method for `Microsoft.VisualStudio.TestTools.UnitTesting.Assert`. The functions in the `Assert` class are static, so there's no need to access the `this` parameter in the method body.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1801
// The code that's violating the rule is on this line.
#pragma warning restore CA1801
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1801.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Usage.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

Use the following option to configure which parts of your codebase to run this rule on.

- [Include specific API surfaces](#)

You can configure this option for just this rule, for all rules, or for all rules in this category (Performance). For more information, see [Code quality rule configuration options](#).

Include specific API surfaces

You can configure which parts of your codebase to run this rule on, based on their accessibility. For example, to specify that the rule should run only against the non-public API surface, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXX.api_surface = private, internal
```

By default, the CA1801 rule applies to all API surfaces (public, internal, and private).

Example

The following example shows two methods. One method violates the rule and the other method satisfies the rule.

```
// This method violates the rule.
public static string GetSomething(int first, int second)
{
    return first.ToString(CultureInfo.InvariantCulture);
}

// This method satisfies the rule.
public static string GetSomethingElse(int first)
{
    return first.ToString(CultureInfo.InvariantCulture);
}
```

Related rules

- [CA1812: Avoid uninstantiated internal classes](#)
- [CA2229: Implement serialization constructors](#)

CA1816: Call GC.SuppressFinalize correctly

9/20/2022 • 3 minutes to read • [Edit Online](#)

	Value
Rule ID	CA1816
Category	Usage
Fix is breaking or non-breaking	Non-breaking

Cause

Violations of this rule can be caused by:

- In an unsealed class, a method that's an implementation of [IDisposable.Dispose](#) and doesn't call [GC.SuppressFinalize](#).
- A method that's not an implementation of [IDisposable.Dispose](#) and calls [GC.SuppressFinalize](#).
- A method that calls [GC.SuppressFinalize](#) and passes something other than `this` (C#) or `Me` (Visual Basic).

Rule description

The [IDisposable.Dispose](#) method lets users release resources at any time before the object becomes available for garbage collection. If the [IDisposable.Dispose](#) method is called, it frees resources of the object. This makes finalization unnecessary. [IDisposable.Dispose](#) should call [GC.SuppressFinalize](#) so the garbage collector doesn't call the finalizer of the object.

To prevent derived types with finalizers from having to reimplement [IDisposable](#) and to call it, unsealed types without finalizers should still call [GC.SuppressFinalize](#).

How to fix violations

To fix a violation of this rule:

- If the method is an implementation of [Dispose](#), add a call to [GC.SuppressFinalize](#).
- If the method is not an implementation of [Dispose](#), either remove the call to [GC.SuppressFinalize](#) or move it to the type's [Dispose](#) implementation.
- Change all calls to [GC.SuppressFinalize](#) to pass `this` (C#) or `Me` (Visual Basic).
- If the type is not meant to be overridden, mark it as `sealed`.

When to suppress warnings

Only suppress a warning from this rule if you're deliberately using [GC.SuppressFinalize](#) to control the lifetime of other objects. Don't suppress a warning from this rule if an implementation of [Dispose](#) doesn't call [GC.SuppressFinalize](#). In this situation, failing to suppress finalization degrades performance and provides no benefits.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA1816
// The code that's violating the rule is on this line.
#pragma warning restore CA1816
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA1816.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Usage.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Example that violates CA1816

This code shows a method that calls `GC.SuppressFinalize`, but doesn't pass [this \(C#\)](#) or [Me \(Visual Basic\)](#). As a result, this code violates rule CA1816.

```
Public Class DatabaseConnector
    Implements IDisposable

    Private _Connection As New SqlConnection

    Public Sub Dispose() Implements IDisposable.Dispose
        Dispose(True)
        ' Violates rules
        GC.SuppressFinalize(True)
    End Sub

    Protected Overridable Sub Dispose(ByVal disposing As Boolean)
        If disposing Then
            If _Connection IsNot Nothing Then
                _Connection.Dispose()
                _Connection = Nothing
            End If
        End If
    End Sub

End Class
```

```

public class DatabaseConnector : IDisposable
{
    private SqlConnection _Connection = new SqlConnection();

    public void Dispose()
    {
        Dispose(true);
        GC.SuppressFinalize(true); // Violates rule
    }

    protected virtual void Dispose(bool disposing)
    {
        if (disposing)
        {
            if (_Connection != null)
            {
                _Connection.Dispose();
                _Connection = null;
            }
        }
    }
}

```

Example that satisfies CA1816

This example shows a method that correctly calls [GC.SuppressFinalize](#) by passing [this \(C#\)](#) or [Me \(Visual Basic\)](#).

```

Public Class DatabaseConnector
    Implements IDisposable

    Private _Connection As New SqlConnection

    Public Sub Dispose() Implements IDisposable.Dispose
        Dispose(True)
        GC.SuppressFinalize(Me)
    End Sub

    Protected Overridable Sub Dispose(ByVal disposing As Boolean)
        If disposing Then
            If _Connection IsNot Nothing Then
                _Connection.Dispose()
                _Connection = Nothing
            End If
        End If
    End Sub

End Class

```

```
public class DatabaseConnector : IDisposable
{
    private SqlConnection _Connection = new SqlConnection();

    public void Dispose()
    {
        Dispose(true);
        GC.SuppressFinalize(this);
    }

    protected virtual void Dispose(bool disposing)
    {
        if (disposing)
        {
            if (_Connection != null)
            {
                _Connection.Dispose();
                _Connection = null;
            }
        }
    }
}
```

Related rules

- [CA2215: Dispose methods should call base class dispose](#)
- [CA2216: Disposable types should declare finalizer](#)

See also

- [Dispose pattern](#)

CA2200: Rethrow to preserve stack details

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA2200
Category	Usage
Fix is breaking or non-breaking	Non-breaking

Cause

An exception is rethrown and the exception is explicitly specified in the `throw` statement.

Rule description

Once an exception is thrown, part of the information it carries is the stack trace. The stack trace is a list of the method call hierarchy that starts with the method that throws the exception and ends with the method that catches the exception. If an exception is re-thrown by specifying the exception in the `throw` statement, the stack trace is restarted at the current method and the list of method calls between the original method that threw the exception and the current method is lost. To keep the original stack trace information with the exception, use the `throw` statement without specifying the exception.

How to fix violations

To fix a violation of this rule, rethrow the exception without specifying the exception explicitly.

When to suppress warnings

Do not suppress a warning from this rule.

Example

The following example shows a method, `CatchAndRethrowExplicitly`, which violates the rule and a method, `CatchAndRethrowImplicitly`, which satisfies the rule.

```
class TestsRethrow
{
    static void Main2200()
    {
        TestsRethrow testRethrow = new TestsRethrow();
        testRethrow.CatchException();
    }

    void CatchException()
    {
        try
        {
            CatchAndRethrowExplicitly();
        }
        catch (ArithmeticeException e)
        {
            Console.WriteLine("Explicitly specified:{0}{1}",
                Environment.NewLine, e.StackTrace);
        }

        try
        {
            CatchAndRethrowImplicitly();
        }
        catch (ArithmeticeException e)
        {
            Console.WriteLine("{0}Implicitly specified:{0}{1}",
                Environment.NewLine, e.StackTrace);
        }
    }

    void CatchAndRethrowExplicitly()
    {
        try
        {
            ThrowException();
        }
        catch (ArithmeticeException e)
        {
            // Violates the rule.
            throw e;
        }
    }

    void CatchAndRethrowImplicitly()
    {
        try
        {
            ThrowException();
        }
        catch (ArithmeticeException)
        {
            // Satisfies the rule.
            throw;
        }
    }

    void ThrowException()
    {
        throw new ArithmeticeException("illegal expression");
    }
}
```

```

Imports System

Namespace ca2200

    Class TestsRethrow

        Shared Sub Main2200()
            Dim testRethrow As New TestsRethrow()
            testRethrow.CatchException()
        End Sub

        Sub CatchException()

            Try
                CatchAndRethrowExplicitly()
            Catch e As ArithmeticException
                Console.WriteLine("Explicitly specified:{0}{1}",
                    Environment.NewLine, e.StackTrace)
            End Try

            Try
                CatchAndRethrowImplicitly()
            Catch e As ArithmeticException
                Console.WriteLine("{0}Implicitly specified:{0}{1}",
                    Environment.NewLine, e.StackTrace)
            End Try

        End Sub

        Sub CatchAndRethrowExplicitly()

            Try
                ThrowException()
            Catch e As ArithmeticException

                ' Violates the rule.
                Throw e
            End Try

        End Sub

        Sub CatchAndRethrowImplicitly()

            Try
                ThrowException()
            Catch e As ArithmeticException

                ' Satisfies the rule.
                Throw
            End Try

        End Sub

        Sub ThrowException()
            Throw New ArithmeticException("illegal expression")
        End Sub

    End Class

End Namespace

```

CA2201: Do not raise reserved exception types

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA2201
Category	Usage
Fix is breaking or non-breaking	Breaking

Cause

A method raises an exception type that is too general or that is reserved by the runtime.

Rule description

The following exception types are too general to provide sufficient information to the user:

- [System.Exception](#)
- [System.ApplicationException](#)
- [System.SystemException](#)

The following exception types are reserved and should be thrown only by the common language runtime:

- [System.AccessViolationException](#)
- [System.ExecutionEngineException](#)
- [System.IndexOutOfRangeException](#)
- [System.NullReferenceException](#)
- [System.OutOfMemoryException](#)
- [System.Runtime.InteropServices.COMException](#)
- [System.Runtime.InteropServices.ExternalException](#)
- [System.Runtime.InteropServices.SEHException](#)
- [System.StackOverflowException](#)

Do Not Throw General Exceptions

If you throw a general exception type, such as [Exception](#) or [SystemException](#) in a library or framework, it forces consumers to catch all exceptions, including unknown exceptions that they do not know how to handle.

Instead, either throw a more derived type that already exists in the framework, or create your own type that derives from [Exception](#).

Throw Specific Exceptions

The following table shows parameters and which exceptions to throw when you validate the parameter,

including the value parameter in the set accessor of a property:

PARAMETER DESCRIPTION	EXCEPTION
null reference	System.ArgumentNullException
Outside the allowed range of values (such as an index for a collection or list)	System.ArgumentOutOfRangeException
Invalid enum value	System.ComponentModel.InvalidEnumArgumentException
Contains a format that does not meet the parameter specifications of a method (such as the format string for <code>ToString(String)</code>)	System.FormatException
Otherwise invalid	System.ArgumentException

When an operation is invalid for the current state of an object throw [System.InvalidOperationException](#)

When an operation is performed on an object that has been disposed throw [System.ObjectDisposedException](#)

When an operation is not supported (such as in an overridden `Stream.Write` in a Stream opened for reading) throw [System.NotSupportedException](#)

When a conversion would result in an overflow (such as in an explicit cast operator overload) throw [System.OverflowException](#)

For all other situations, consider creating your own type that derives from [Exception](#) and throw that.

How to fix violations

To fix a violation of this rule, change the type of the thrown exception to a specific type that is not one of the reserved types.

When to suppress warnings

Do not suppress a warning from this rule.

Related rules

- [CA1031: Do not catch general exception types](#)

CA2207: Initialize value type static fields inline

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA2207
Category	Usage
Fix is breaking or non-breaking	Non-breaking

Cause

A value-type declares an explicit static constructor.

Rule description

When a value-type is declared, it undergoes a default initialization where all value-type fields are set to zero and all reference-type fields are set to `null` (`Nothing` in Visual Basic). An explicit static constructor is only guaranteed to run before an instance constructor or static member of the type is called. Therefore, if the type is created without calling an instance constructor, the static constructor is not guaranteed to run.

If all static data is initialized inline and no explicit static constructor is declared, the C# and Visual Basic compilers add the `beforefieldinit` flag to the MSIL class definition. The compilers also add a private static constructor that contains the static initialization code. This private static constructor is guaranteed to run before any static fields of the type are accessed.

How to fix violations

To fix a violation of this rule initialize all static data when it is declared and remove the static constructor.

When to suppress warnings

Do not suppress a warning from this rule.

Related rules

[CA1810: Initialize reference type static fields inline](#)

CA2208: Instantiate argument exceptions correctly

9/20/2022 • 3 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA2208
Category	Usage
Fix is breaking or non-breaking	Non-breaking

Cause

When a method has a parameter, and it throws an exception type that is, or derives from, [ArgumentException](#), it is expected to call a constructor accepting a `paramName` parameter correctly. Possible causes include the following situations:

- A call is made to the default (parameterless) constructor of an exception type that is, or derives from, [ArgumentException](#) that has a constructor accepting a `paramName` parameter.
- An incorrect string argument is passed to a parameterized constructor of an exception type that is, or derives from, [ArgumentException](#).
- One of the parameters' names is passed for the `message` argument of the constructor of exception type that is, or derives from, [ArgumentException](#).

Rule description

Instead of calling the default constructor, call one of the constructor overloads that allows a more meaningful exception message to be provided. The exception message should target the developer and clearly explain the error condition and how to correct or avoid the exception.

The signatures of the one and two string constructors of [ArgumentException](#) and its derived types are not consistent with respect to the position `message` and `paramName` parameters. Make sure these constructors are called with the correct string arguments. The signatures are as follows:

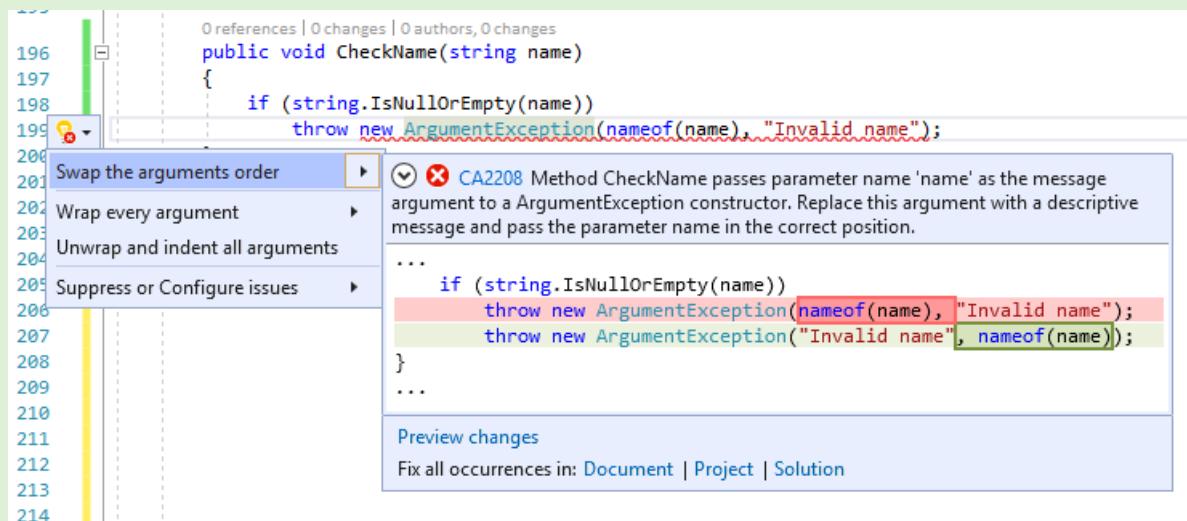
- `ArgumentException(string message)`
- `ArgumentException(string message, string paramName)`
- `ArgumentNullException(string paramName)`
- `ArgumentNullException(string paramName, string message)`
- `ArgumentOutOfRangeException(string paramName)`
- `ArgumentOutOfRangeException(string paramName, string message)`
- `DuplicateWaitObjectException(string parameterName)`
- `DuplicateWaitObjectException(string parameterName, string message)`

How to fix violations

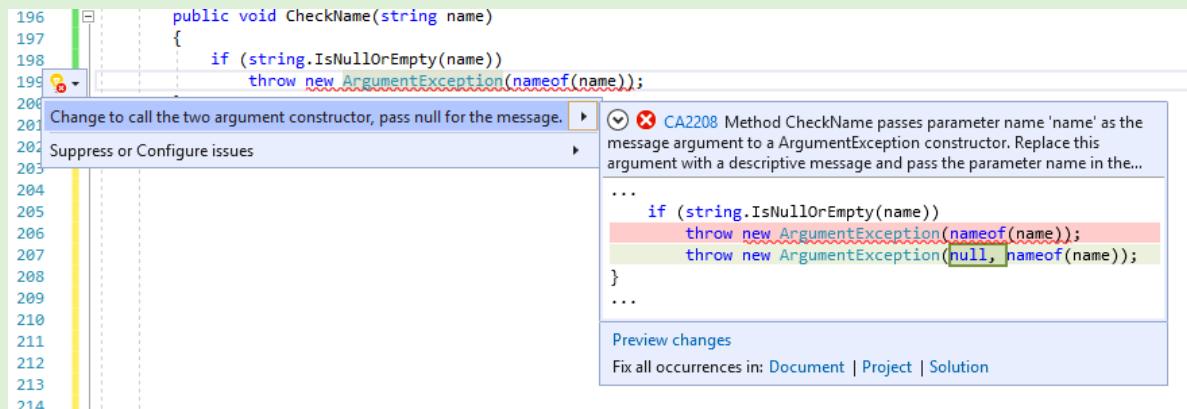
To fix a violation of this rule, call a constructor that takes a message, a parameter name, or both, and make sure the arguments are proper for the type of [ArgumentException](#) being called.

TIP

A code fix is available in Visual Studio for incorrectly positioned parameter names. To use it, position the cursor on the warning row and press **Ctrl+.** (period). Choose **Swap the arguments order** from the list of options that's presented.



If a parameter name instead of a message is passed to the `ArgumentException(String)` method, the fixer provides the option to switch to the two-argument constructor instead.



When to suppress warnings

It's safe to suppress a warning from this rule only if a parameterized constructor is called with the correct string arguments.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA2208
// The code that's violating the rule is on this line.
#pragma warning restore CA2208
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA2208.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Usage.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

Use the following option to configure which parts of your codebase to run this rule on.

- [Include specific API surfaces](#)

You can configure this option for just this rule, for all rules, or for all rules in this category (Design). For more information, see [Code quality rule configuration options](#).

Include specific API surfaces

You can configure which parts of your codebase to run this rule on, based on their accessibility. For example, to specify that the rule should run only against the non-public API surface, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXX.api_surface = private, internal
```

By default, the CA2208 rule applies to all API surfaces (public, internal, and private).

Example

The following code shows a constructor that incorrectly instantiates an instance of `ArgumentNullException`.

```
public class Book
{
    public Book(string title)
    {
        Title = title ??
            throw new ArgumentNullException("All books must have a title.", nameof(title));
    }

    public string Title { get; }
}
```

```

Public Class Book

    Private ReadOnly _Title As String

    Public Sub New(ByVal title As String)
        ' Violates this rule (constructor arguments are switched)
        If (title Is Nothing) Then
            Throw New ArgumentNullException("title cannot be a null reference (Nothing in Visual Basic)",
"title")
        End If
        _Title = title
    End Sub

    Public ReadOnly Property Title()
        Get
            Return _Title
        End Get
    End Property

End Class

```

The following code fixes the previous violation by switching the constructor arguments.

```

public class Book
{
    public Book(string title)
    {
        Title = title ??
            throw new ArgumentNullException(nameof(title), "All books must have a title.");
    }

    public string Title { get; }
}

```

```

Public Class Book

    Private ReadOnly _Title As String

    Public Sub New(ByVal title As String)
        If (title Is Nothing) Then
            Throw New ArgumentNullException("title", "title cannot be a null reference (Nothing in Visual
Basic)")
        End If

        _Title = title
    End Sub

    Public ReadOnly Property Title()
        Get
            Return _Title
        End Get
    End Property

End Class

```

Related rules

- [CA1507: Use nameof in place of string](#)

CA2211: Non-constant fields should not be visible

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA2211
Category	Usage
Fix is breaking or non-breaking	Breaking

Cause

A public or protected static field is not constant nor is it read-only.

Rule description

Static fields that are neither constants nor read-only are not thread-safe. Access to such a field must be carefully controlled and requires advanced programming techniques for synchronizing access to the class object. Because these are difficult skills to learn, and testing such an object poses its own challenges, static fields are best used to store data that does not change. This rule applies to libraries; applications should not expose any fields.

How to fix violations

To fix a violation of this rule, make the static field constant or read-only. If this is not possible, redesign the type to use an alternative mechanism such as a thread-safe property that manages thread-safe access to the underlying field. Realize that issues such as lock contention and deadlocks might affect the performance and behavior of the library.

When to suppress warnings

It is safe to suppress a warning from this rule if you are developing an application and therefore have full control over access to the type that contains the static field. Library designers should not suppress a warning from this rule; using non-constant static fields can make using the library difficult for developers to use correctly.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA2211
// The code that's violating the rule is on this line.
#pragma warning restore CA2211
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA2211.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Usage.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Example

The following example shows a type that violates this rule.

```
Imports System

Namespace ca2211

    Public Class SomeStaticFields
        ' Violates rule: AvoidNonConstantStatic;
        ' the field is public and not a literal.
        Public Shared publicField As DateTime = DateTime.Now

        ' Satisfies rule: AvoidNonConstantStatic.
        Public Shared ReadOnly literalField As DateTime = DateTime.Now

        ' Satisfies rule: NonConstantFieldsShouldNotBeVisible;
        ' the field is private.
        Private Shared privateField As DateTime = DateTime.Now
    End Class
End Namespace
```

```
public class SomeStaticFields
{
    // Violates rule: AvoidNonConstantStatic;
    // the field is public and not a literal.
    static public DateTime publicField = DateTime.Now;

    // Satisfies rule: AvoidNonConstantStatic.
    public static readonly DateTime literalField = DateTime.Now;

    // Satisfies rule: NonConstantFieldsShouldNotBeVisible;
    // the field is private.
    static DateTime privateField = DateTime.Now;
}
```

CA2213: Disposable fields should be disposed

9/20/2022 • 3 minutes to read • [Edit Online](#)

	Value
Rule ID	CA2213
Category	Usage
Fix is breaking or non-breaking	Non-breaking

Cause

A type that implements [System.IDisposable](#) declares fields that are of types that also implement [IDisposable](#). The [Dispose](#) method of the field is not called by the [Dispose](#) method of the declaring type.

Rule description

A type is responsible for disposing of all its unmanaged resources. Rule CA2213 checks to see whether a disposable type (that is, one that implements [IDisposable](#)) `T` declares a field `F` that is an instance of a disposable type `FT`. For each field `F` that's assigned a locally created object within the methods or initializers of the containing type `T`, the rule attempts to locate a call to `FT.Dispose`. The rule searches the methods called by `T.Dispose` and one level lower (that is, the methods called by the methods called by `T.Dispose`).

Note

Other than the [special cases](#), rule CA2213 fires only for fields that are assigned a locally created disposable object within the containing type's methods and initializers. If the object is created or assigned outside of type `T`, the rule does not fire. This reduces noise for cases where the containing type doesn't own the responsibility for disposing of the object.

Special cases

Rule CA2213 can also fire for fields of the following types even if the object they're assigned isn't created locally:

- [System.IO.Stream](#)
- [System.IO.TextReader](#)
- [System.IO.TextWriter](#)
- [System.Resources.IResourceReader](#)

Passing an object of one of these types to a constructor and then assigning it to a field indicates a *dispose ownership transfer* to the newly constructed type. That is, the newly constructed type is now responsible for disposing of the object. If the object is not disposed, a violation of CA2213 occurs.

How to fix violations

To fix a violation of this rule, call [Dispose](#) on fields that are of types that implement [IDisposable](#).

When to suppress warnings

It's safe to suppress a warning from this rule if:

- The flagged type is not responsible for releasing the resource held by the field (that is, the type does not have *dispose ownership*)
- The call to [Dispose](#) occurs at a deeper calling level than the rule checks
- the dispose ownership of the field(s) is not held by the containing type.

SUPPRESS A WARNING

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA2213
// The code that's violating the rule is on this line.
#pragma warning restore CA2213
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.cs,vb]
dotnet_diagnostic.CA2213.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.cs,vb]
dotnet_analyzer_diagnostic.category-Usage.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

EXAMPLE

The following snippet shows a type `TypeA` that implements [IDisposable](#).

```
public class TypeA : IDisposable
{
    protected virtual void Dispose(bool disposing)
    {
        if (disposing)
        {
            // Dispose managed resources
        }

        // Free native resources
    }

    public void Dispose()
    {
        Dispose(true);
        GC.SuppressFinalize(this);
    }

    // Disposable types implement a finalizer.
    ~TypeA()
    {
        Dispose(false);
    }
}
```

The following snippet shows a type `TypeB` that violates rule CA2213 by declaring a field `aFieldOfADisposableType` as a disposable type (`TypeA`) and not calling `Dispose` on the field.

```
public class TypeB : IDisposable
{
    // Assume this type has some unmanaged resources.
    TypeA aFieldOfADisposableType = new TypeA();
    private bool disposed = false;

    protected virtual void Dispose(bool disposing)
    {
        if (!disposed)
        {
            // Dispose of resources held by this instance.

            // Violates rule: DisposableFieldsShouldBeDisposed.
            // Should call aFieldOfADisposableType.Dispose();

            disposed = true;
            // Suppress finalization of this disposed instance.
            if (disposing)
            {
                GC.SuppressFinalize(this);
            }
        }
    }

    public void Dispose()
    {
        if (!disposed)
        {
            // Dispose of resources held by this instance.
            Dispose(true);
        }
    }

    // Disposable types implement a finalizer.
    ~TypeB()
    {
        Dispose(false);
    }
}
```

To fix the violation, call `Dispose()` on the disposable field:

```
protected virtual void Dispose(bool disposing)
{
    if (!disposed)
    {
        // Dispose of resources held by this instance.
        aFieldOfADisposableType.Dispose();

        disposed = true;

        // Suppress finalization of this disposed instance.
        if (disposing)
        {
            GC.SuppressFinalize(this);
        }
    }
}
```

See also

- [System.IDisposable](#)
- [Dispose pattern](#)

CA2214: Do not call overridable methods in constructors

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA2214
Category	Usage
Fix is breaking or non-breaking	Non-breaking

Cause

The constructor of an unsealed type calls a virtual method defined in its class.

Rule description

When a virtual method is called, the actual type that executes the method is not selected until run time. When a constructor calls a virtual method, it's possible that the constructor for the instance that invokes the method has not executed. This could lead to errors or unexpected behavior, if an overridden virtual method relies on initialization and other configuration in the constructor.

How to fix violations

To fix a violation of this rule, do not call a type's virtual methods from within the type's constructors.

When to suppress warnings

Do not suppress a warning from this rule. The constructor should be redesigned to eliminate the call to the virtual method.

Example

The following example demonstrates the effect of violating this rule. The test application creates an instance of `DerivedType`, which causes its base class (`BadlyConstructedType`) constructor to execute. `BadlyConstructedType`'s constructor incorrectly calls the virtual method `DoSomething`. As the output shows, `DerivedType.DoSomething()` executes before `DerivedType`'s constructor executes.

```
public class BadlyConstructedType
{
    protected string initialized = "No";

    public BadlyConstructedType()
    {
        Console.WriteLine("Calling base ctor.");
        // Violates rule: DoNotCallOverridableMethodsInConstructors.
        DoSomething();
    }
    // This will be overridden in the derived type.
    public virtual void DoSomething()
    {
        Console.WriteLine("Base DoSomething");
    }
}

public class DerivedType : BadlyConstructedType
{
    public DerivedType()
    {
        Console.WriteLine("Calling derived ctor.");
        initialized = "Yes";
    }
    public override void DoSomething()
    {
        Console.WriteLine("Derived DoSomething is called - initialized ? {0}", initialized);
    }
}

public class TestBadlyConstructedType
{
    public static void Main2214()
    {
        DerivedType derivedInstance = new DerivedType();
    }
}
```

```

Imports System

Namespace ca2214

    Public Class BadlyConstructedType
        Protected initialized As String = "No"

        Public Sub New()
            Console.WriteLine("Calling base ctor.")
            ' Violates rule: DoNotCallOverridableMethodsInConstructors.
            DoSomething()
        End Sub 'New

        ' This will be overridden in the derived type.
        Public Overridable Sub DoSomething()
            Console.WriteLine("Base DoSomething")
        End Sub 'DoSomething
    End Class 'BadlyConstructedType


    Public Class DerivedType
        Inherits BadlyConstructedType

        Public Sub New()
            Console.WriteLine("Calling derived ctor.")
            initialized = "Yes"
        End Sub 'New

        Public Overrides Sub DoSomething()
            Console.WriteLine("Derived DoSomething is called - initialized ? {0}", initialized)
        End Sub 'DoSomething
    End Class 'DerivedType


    Public Class TestBadlyConstructedType

        Public Shared Sub Main2214()
            Dim derivedInstance As New DerivedType()
        End Sub 'Main
    End Class
End Namespace

```

This example produces the following output:

```

Calling base ctor.
Derived DoSomething is called - initialized ? No
Calling derived ctor.

```

CA2215: Dispose methods should call base class dispose

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA2215
Category	Usage
Fix is breaking or non-breaking	Non-breaking

Cause

A type that implements [System.IDisposable](#) inherits from a type that also implements [IDisposable](#). The [Dispose](#) method of the inheriting type does not call the [Dispose](#) method of the parent type.

Rule description

If a type inherits from a disposable type, it must call the [Dispose](#) method of the base type from within its own [Dispose](#) method. Calling the base type `Dispose` method ensures that any resources created by the base type are released.

How to fix violations

To fix a violation of this rule, call `base.Dispose` in your [Dispose](#) method.

When to suppress warnings

It is safe to suppress a warning from this rule if the call to `base.Dispose` occurs at a deeper calling level than the rule checks.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA2215
// The code that's violating the rule is on this line.
#pragma warning restore CA2215
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA2215.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Usage.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Example

The following example shows two types, `TypeA` that implements `IDisposable`, and `TypeB` that inherits from type `TypeA` and correctly calls its `Dispose` method.

```
Namespace ca2215

    Public Class TypeA
        Implements IDisposable

            Protected Overridable Overloads Sub Dispose(disposing As Boolean)
                If disposing Then
                    ' dispose managed resources
                End If

                ' free native resources
            End Sub

            Public Overloads Sub Dispose() Implements IDisposable.Dispose
                Dispose(True)
                GC.SuppressFinalize(Me)
            End Sub

            ' Disposable types implement a finalizer.
            Protected Overrides Sub Finalize()
                Dispose(False)
                MyBase.Finalize()
            End Sub

        End Class

        Public Class TypeB
            Inherits TypeA

            Protected Overrides Sub Dispose(disposing As Boolean)
                If Not disposing Then
                    MyBase.Dispose(False)
                End If
            End Sub

        End Class
    End Namespace
```

```
using System;

namespace ca2215
{
    public class TypeA : IDisposable
    {
        protected virtual void Dispose(bool disposing)
        {
            if (disposing)
            {
                // Dispose managed resources
            }

            // Free native resources
        }

        public void Dispose()
        {
            Dispose(true);
            GC.SuppressFinalize(this);
        }

        // Disposable types implement a finalizer.
        ~TypeA()
        {
            Dispose(false);
        }
    }

    public class TypeB : TypeA
    {
        protected override void Dispose(bool disposing)
        {
            if (!disposing)
            {
                base.Dispose(false);
            }
        }
    }
}
```

See also

- [System.IDisposable](#)
- [Dispose Pattern](#)

CA2216: Disposable types should declare finalizer

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA2216
Category	Usage
Fix is breaking or non-breaking	Non-breaking

Cause

A type that implements [System.IDisposable](#), and has fields that suggest the use of unmanaged resources, does not implement a finalizer as described by [System.Object.Finalize](#).

Rule description

A violation of this rule is reported if the disposable type contains fields of the following types:

- [System.IntPtr](#)
- [System.UIntPtr](#)
- [System.Runtime.InteropServices.HandleRef](#)

How to fix violations

To fix a violation of this rule, implement a finalizer that calls your [Dispose](#) method.

When to suppress warnings

It is safe to suppress a warning from this rule if the type does not implement [IDisposable](#) for the purpose of releasing unmanaged resources.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA2216
// The code that's violating the rule is on this line.
#pragma warning restore CA2216
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA2216.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Usage.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Example

The following example shows a type that violates this rule.

```
public class DisposeMissingFinalize : IDisposable
{
    private bool disposed = false;
    private IntPtr unmanagedResource;

    [DllImport("native.dll")]
    private static extern IntPtr AllocateUnmanagedResource();

    [DllImport("native.dll")]
    private static extern void FreeUnmanagedResource(IntPtr p);

    DisposeMissingFinalize()
    {
        unmanagedResource = AllocateUnmanagedResource();
    }

    protected virtual void Dispose(bool disposing)
    {
        if (!disposed)
        {
            // Dispose of resources held by this instance.
            FreeUnmanagedResource(unmanagedResource);
            disposed = true;

            // Suppress finalization of this disposed instance.
            if (disposing)
            {
                GC.SuppressFinalize(this);
            }
        }
    }

    public void Dispose()
    {
        Dispose(true);
    }

    // Disposable types with unmanaged resources implement a finalizer.
    // Uncomment the following code to satisfy rule:
    // DisposableTypesShouldDeclareFinalizer
    // ~TypeA()
    // {
    //     Dispose(false);
    // }
}
```

Related rules

[CA1816: Call GC.SuppressFinalize correctly](#)

See also

- [System.IDisposable](#)

- [System.IntPtr](#)
- [System.Runtime.InteropServices.HandleRef](#)
- [System.UIntPtr](#)
- [System.Object.Finalize](#)
- [Dispose Pattern](#)

CA2217: Do not mark enums with FlagsAttribute

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA2217
Category	Usage
Fix is breaking or non-breaking	Non-breaking

Cause

An enumeration is marked with [FlagsAttribute](#) and it has one or more values that are not powers of two or a combination of the other defined values on the enumeration.

By default, this rule only looks at externally visible enumerations, but this is [configurable](#).

Rule description

An enumeration should have [FlagsAttribute](#) present only if each value defined in the enumeration is a power of two or a combination of defined values.

How to fix violations

To fix a violation of this rule, remove [FlagsAttribute](#) from the enumeration.

When to suppress warnings

Do not suppress a warning from this rule.

Configure code to analyze

Use the following option to configure which parts of your codebase to run this rule on.

- [Include specific API surfaces](#)

You can configure this option for just this rule, for all rules, or for all rules in this category ([Usage](#)). For more information, see [Code quality rule configuration options](#).

Include specific API surfaces

You can configure which parts of your codebase to run this rule on, based on their accessibility. For example, to specify that the rule should run only against the non-public API surface, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.api_surface = private, internal
```

Examples

The following code shows an enumeration, `Color`, that contains the value 3. 3 is not a power of two, or a combination of any of the defined values. The `Color` enumeration shouldn't be marked with [FlagsAttribute](#).

```
// Violates this rule
[FlagsAttribute]
public enum Color
{
    None = 0,
    Red = 1,
    Orange = 3,
    Yellow = 4
}
```

```
Imports System

Namespace Samples

    ' Violates this rule
    <FlagsAttribute()> _
    Public Enum Color

        None = 0
        Red = 1
        Orange = 3
        Yellow = 4

    End Enum
End Namespace
```

The following code shows an enumeration, `Days`, that meets the requirements for being marked with [FlagsAttribute](#):

```
[FlagsAttribute]
public enum Days
{
    None = 0,
    Monday = 1,
    Tuesday = 2,
    Wednesday = 4,
    Thursday = 8,
    Friday = 16,
    All = Monday | Tuesday | Wednesday | Thursday | Friday
}
```

```
Imports System
Namespace Samples

    <FlagsAttribute()> _
    Public Enum Days

        None = 0
        Monday = 1
        Tuesday = 2
        Wednesday = 4
        Thursday = 8
        Friday = 16
        All = Monday Or Tuesday Or Wednesday Or Thursday Or Friday

    End Enum
End Namespace
```

Related rules

[CA1027: Mark enums with FlagsAttribute](#)

See also

- [System.FlagsAttribute](#)

CA2218: Override GetHashCode on overriding Equals

9/20/2022 • 2 minutes to read • [Edit Online](#)

ITEM	VALUE
RuleId	CA2218
Category	Usage
Breaking change	Non-breaking

Cause

A public type overrides [System.Object.Equals](#) but does not override [System.Object.GetHashCode](#).

Rule description

[GetHashCode](#) returns a value, based on the current instance, that's suited for hashing algorithms and data structures such as hash tables. Two objects that are the same type and are equal must return the same hash code to ensure that instances of the following types work correctly:

- [System.Collections.Hashtable](#)
- [System.Collections.SortedList](#)
- [System.Collections.Generic.Dictionary< TKey, TValue >](#)
- [System.Collections.Generic.SortedDictionary< TKey, TValue >](#)
- [System.Collections.Generic.SortedList< TKey, TValue >](#)
- [System.Collections.Specialized.HybridDictionary](#)
- [System.Collections.Specialized.ListDictionary](#)
- [System.Collections.Specialized.OrderedDictionary](#)
- Types that implement [System.Collections.Generic.IEqualityComparer< T >](#)

NOTE

This rule only applies to Visual Basic code. The C# compiler generates a separate warning, [CS0659](#).

How to fix violations

To fix a violation of this rule, provide an implementation of [GetHashCode](#). For a pair of objects of the same type, ensure that the implementation returns the same value if your implementation of [Equals](#) returns `true` for the pair.

When to suppress warnings

Do not suppress a warning from this rule.

Class example

The following example shows a class (reference type) that violates this rule.

```
' This class violates the rule.
Public Class Point

    Public Property X As Integer
    Public Property Y As Integer

    Public Sub New(x As Integer, y As Integer)
        Me.X = x
        Me.Y = y
    End Sub

    Public Overrides Function Equals(obj As Object) As Boolean

        If obj = Nothing Then
            Return False
        End If

        If [GetType]() <> obj.GetType() Then
            Return False
        End If

        Dim pt As Point = CType(obj, Point)

        Return X = pt.X AndAlso Y = pt.Y
    End Function

End Class
```

The following example fixes the violation by overriding [GetHashCode\(\)](#).

```

' This class satisfies the rule.
Public Class Point

    Public Property X As Integer
    Public Property Y As Integer

    Public Sub New(x As Integer, y As Integer)
        Me.X = x
        Me.Y = y
    End Sub

    Public Overrides Function GetHashCode() As Integer
        Return X Or Y
    End Function

    Public Overrides Function Equals(obj As Object) As Boolean

        If obj = Nothing Then
            Return False
        End If

        If [GetType]() <> obj.GetType() Then
            Return False
        End If

        Dim pt As Point = CType(obj, Point)

        Return Equals(pt)
    End Function

    Public Overloads Function Equals(pt As Point) As Boolean
        Return X = pt.X AndAlso Y = pt.Y
    End Function

    Public Shared Operator =(pt1 As Point, pt2 As Point) As Boolean
        Return pt1.Equals(pt2)
    End Operator

    Public Shared Operator <>(pt1 As Point, pt2 As Point) As Boolean
        Return Not pt1.Equals(pt2)
    End Operator
End Class

```

Related rules

- [CA1046: Do not overload operator equals on reference types](#)
- [CA2224: Override equals on overloading operator equals](#)
- [CA2225: Operator overloads have named alternates](#)
- [CA2226: Operators should have symmetrical overloads](#)
- [CA2231: Overload operator equals on overriding ValueType.Equals](#)

See also

- [CS0659](#)
- [System.Object.Equals](#)
- [System.Object.GetHashCode](#)
- [System.Collections.Hashtable](#)
- [Equality Operators](#)

CA2219: Do not raise exceptions in exception clauses

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA2219
Category	Usage
Fix is breaking or non-breaking	Non-breaking, Breaking

Cause

An exception is thrown from a `finally`, filter, or fault clause.

Rule description

When an exception is raised in an exception clause, it greatly increases the difficulty of debugging.

When an exception is raised in a `finally` or fault clause, the new exception hides the active exception, if present. This makes the original error hard to detect and debug.

When an exception is raised in a filter clause, the runtime silently catches the exception, and causes the filter to evaluate to false. There is no way to tell the difference between the filter evaluating to false and an exception being thrown from a filter. This makes it hard to detect and debug errors in the filter's logic.

How to fix violations

To fix this violation of this rule, do not explicitly raise an exception from a `finally`, filter, or fault clause.

When to suppress warnings

Do not suppress a warning for this rule. There are no scenarios under which an exception raised in an exception clause provides a benefit to the executing code.

Related rules

[CA1065: Do not raise exceptions in unexpected locations](#)

See also

- [Design rules](#)

CA2224: Override Equals on overloading operator equals

9/20/2022 • 2 minutes to read • [Edit Online](#)

ITEM	VALUE
RuleId	CA2224
Category	Usage
Breaking change	Non-breaking

Cause

A public type implements the equality operator but doesn't override [System.Object.Equals](#).

Rule description

The equality operator is intended to be a syntactically convenient way to access the functionality of the [Equals](#) method. If you implement the equality operator, its logic must be identical to that of [Equals](#).

NOTE

This rule only applies to Visual Basic code. The C# compiler generates a separate warning, [CS0660](#).

How to fix violations

To fix a violation of this rule, you should either remove the implementation of the equality operator, or override [Equals](#) and have the two methods return the same values. If the equality operator does not introduce inconsistent behavior, you can fix the violation by providing an implementation of [Equals](#) that calls the [Equals](#) method in the base class.

When to suppress warnings

It is safe to suppress a warning from this rule if the equality operator returns the same value as the inherited implementation of [Equals](#). The examples in this article include a type that could safely suppress a warning from this rule.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA2224
// The code that's violating the rule is on this line.
#pragma warning restore CA2224
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA2224.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Usage.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Example

The following example shows a class (reference type) that violates this rule.

```
' This class violates the rule.
Public Class Point

    Public Property X As Integer
    Public Property Y As Integer

    Public Sub New(x As Integer, y As Integer)
        Me.X = x
        Me.Y = y
    End Sub

    Public Overrides Function GetHashCode() As Integer
        Return HashCode.Combine(X, Y)
    End Function

    Public Shared Operator =(pt1 As Point, pt2 As Point) As Boolean
        If pt1 Is Nothing OrElse pt2 Is Nothing Then
            Return False
        End If

        If pt1.GetType() <> pt2.GetType() Then
            Return False
        End If

        Return pt1.X = pt2.X AndAlso pt1.Y = pt2.Y
    End Operator

    Public Shared Operator <>(pt1 As Point, pt2 As Point) As Boolean
        Return Not pt1 = pt2
    End Operator

End Class
```

The following example fixes the violation by overriding `System.Object.Equals`.

```

' This class satisfies the rule.
Public Class Point

    Public Property X As Integer
    Public Property Y As Integer

    Public Sub New(x As Integer, y As Integer)
        Me.X = x
        Me.Y = y
    End Sub

    Public Overrides Function GetHashCode() As Integer
        Return HashCode.Combine(X, Y)
    End Function

    Public Overrides Function Equals(obj As Object) As Boolean

        If obj = Nothing Then
            Return False
        End If

        If [GetType]() <> obj.GetType() Then
            Return False
        End If

        Dim pt As Point = CType(obj, Point)

        Return X = pt.X AndAlso Y = pt.Y
    End Function

    Public Shared Operator =(pt1 As Point, pt2 As Point) As Boolean
        ' Object.Equals calls Point.Equals(Object).
        Return Object.Equals(pt1, pt2)
    End Operator

    Public Shared Operator <>(pt1 As Point, pt2 As Point) As Boolean
        ' Object.Equals calls Point.Equals(Object).
        Return Not Object.Equals(pt1, pt2)
    End Operator
End Class

```

Related rules

- [CA1046: Do not overload operator equals on reference types](#)
- [CA2218: Override GetHashCode on overriding Equals](#)
- [CA2225: Operator overloads have named alternates](#)
- [CA2226: Operators should have symmetrical overloads](#)
- [CA2231: Overload operator equals on overriding ValueType.Equals](#)

See also

- [CS0660](#)

CA2225: Operator overloads have named alternates

9/20/2022 • 3 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA2225
Category	Usage
Fix is breaking or non-breaking	Non-breaking

Cause

An operator overload was detected and the expected named alternative method was not found.

By default, this rule only looks at externally visible types, but this is [configurable](#).

Rule description

Operator overloading allows the use of symbols to represent computations for a type. For example, a type that overloads the plus symbol `+` for addition would typically have an alternative member named `Add`. The named alternative member provides access to the same functionality as the operator. It's provided for developers who program in languages that do not support overloaded operators.

This rule examines:

- Implicit and explicit cast operators in a type by checking for methods named `To<typename>` and `From<typename>`.
- The operators listed in the following table:

C#	VISUAL BASIC	C++	ALTERNATE METHOD NAME
<code>+(binary)</code>	<code>+</code>	<code>+(binary)</code>	<code>Add</code>
<code>+=</code>	<code>+=</code>	<code>+=</code>	<code>Add</code>
<code>&</code>	<code>And</code>	<code>&</code>	<code>BitwiseAnd</code>
<code>&=</code>	<code>And=</code>	<code>&=</code>	<code>BitwiseAnd</code>
<code> </code>	<code>Or</code>	<code> </code>	<code>BitwiseOr</code>
<code> =</code>	<code>Or=</code>	<code> =</code>	<code>BitwiseOr</code>
<code>--</code>	N/A	<code>--</code>	<code>Decrement</code>
<code>/</code>	<code>/</code>	<code>/</code>	<code>Divide</code>

C#	VISUAL BASIC	C++	ALTERNATE METHOD NAME
/=	/=	/=	Divide
==	=	==	Equals
^	Xor	^	Xor
^=	Xor=	^=	Xor
>	>	>	CompareTo or Compare
>=	>=	>=	CompareTo or Compare
++	N/A	++	Increment
!=	<>	!=	Equals
<<	<<	<<	LeftShift
<<=	<<=	<<=	LeftShift
<	<	<	CompareTo or Compare
<=	<=	<=	CompareTo or Compare
&&	N/A	&&	LogicalAnd
	N/A		LogicalOr
!	N/A	!	LogicalNot
%	Mod	%	Mod or Remainder
%=	N/A	%=	Mod
* (binary)	*	*	Multiply
*=	N/A	*=	Multiply
~	Not	~	OnesComplement
>>	>>	>>	RightShift
=	N/A	>>=	RightShift
- (binary)	- (binary)	- (binary)	Subtract
-=	N/A	-=	Subtract
true	IsTrue	N/A	IsTrue (Property)

C#	VISUAL BASIC	C++	ALTERNATE METHOD NAME
- (unary)	N/A	-	Negate
+ (unary)	N/A	+	Plus
false	IsFalse	False	IsTrue (Property)

*N/A means the operator cannot be overloaded in the selected language.

NOTE

In C#, when a binary operator is overloaded, the corresponding assignment operator, if any, is also implicitly overloaded.

How to fix violations

To fix a violation of this rule, implement the alternative method for the operator. Name it using the recommended alternative name.

When to suppress warnings

Do not suppress a warning from this rule if you're implementing a shared library. Applications can ignore a warning from this rule.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA2225
// The code that's violating the rule is on this line.
#pragma warning restore CA2225
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA2225.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Usage.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

Use the following option to configure which parts of your codebase to run this rule on.

- [Include specific API surfaces](#)

You can configure this option for just this rule, for all rules, or for all rules in this category ([Usage](#)). For more

information, see [Code quality rule configuration options](#).

Include specific API surfaces

You can configure which parts of your codebase to run this rule on, based on their accessibility. For example, to specify that the rule should run only against the non-public API surface, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXX.api_surface = private, internal
```

Example

The following example defines a structure that violates this rule. To correct the example, add a public `Add(int x, int y)` method to the structure.

```
public struct Point
{
    private int x, y;

    public Point(int x, int y)
    {
        this.x = x;
        this.y = y;
    }

    public override string ToString()
    {
        return String.Format("({0},{1})", x, y);
    }

    // Violates rule: OperatorOverloadsHaveNamedAlternates.
    public static Point operator +(Point a, Point b)
    {
        return new Point(a.x + b.x, a.y + b.y);
    }

    public int X { get { return x; } }
    public int Y { get { return x; } }
}
```

Related rules

- [CA1046: Do not overload operator equals on reference types](#)
- [CA2226: Operators should have symmetrical overloads](#)
- [CA2231: Overload operator equals on overriding ValueType.Equals](#)

CA2226: Operators should have symmetrical overloads

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA2226
Category	Usage
Fix is breaking or non-breaking	Non-breaking

Cause

A type implements the equality or inequality operator and does not implement the opposite operator.

By default, this rule only looks at externally visible types, but this is [configurable](#).

Rule description

There are no circumstances where either equality or inequality is applicable to instances of a type, and the opposite operator is undefined. Types typically implement the inequality operator by returning the negated value of the equality operator.

The C# compiler issues an error for violations of this rule.

How to fix violations

To fix a violation of this rule, implement both the equality and inequality operators, or remove the one that's present.

When to suppress warnings

Do not suppress a warning from this rule. If you do, your type will not work in a manner that's consistent with .NET.

Configure code to analyze

Use the following option to configure which parts of your codebase to run this rule on.

- [Include specific API surfaces](#)

You can configure this option for just this rule, for all rules, or for all rules in this category ([Usage](#)). For more information, see [Code quality rule configuration options](#).

Include specific API surfaces

You can configure which parts of your codebase to run this rule on, based on their accessibility. For example, to specify that the rule should run only against the non-public API surface, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXX.api_surface = private, internal
```

Related rules

- [CA1046: Do not overload operator equals on reference types](#)
- [CA2225: Operator overloads have named alternates](#)
- [CA2231: Overload operator equals on overriding ValueType.Equals](#)

CA2227: Collection properties should be read only

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA2227
Category	Usage
Fix is breaking or non-breaking	Breaking

Cause

An externally visible, writable property is of a type that implements [System.Collections.ICollection](#). This rule ignores arrays, indexers (properties with the name 'Item'), immutable collections, readonly collections, and permission sets.

Rule description

A writable collection property allows a user to replace the collection with a completely different collection. A read-only or [init-only](#) property stops the collection from being replaced, but still allows the individual members to be set. If replacing the collection is a goal, the preferred design pattern is to include a method to remove all the elements from the collection, and a method to repopulate the collection. See the [Clear](#) and [AddRange](#) methods of the [System.Collections.ArrayList](#) class for an example of this pattern.

Both binary and XML serialization support read-only properties that are collections. The [System.Xml.Serialization.XmlSerializer](#) class has specific requirements for types that implement [ICollection](#) and [System.Collections.IEnumerable](#) in order to be serializable.

How to fix violations

To fix a violation of this rule, make the property read-only or [init-only](#). If the design requires it, add methods to clear and repopulate the collection.

When to suppress warnings

You can suppress the warning if the property is part of a [Data Transfer Object \(DTO\)](#) class.

Otherwise, do not suppress warnings from this rule.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA2227
// The code that's violating the rule is on this line.
#pragma warning restore CA2227
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA2227.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Usage.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Example

The following example shows a type with a writable collection property and shows how the collection can be replaced directly. Additionally, it shows the preferred manner of replacing a read-only collection property using `Clear` and `AddRange` methods.

```
public class WritableCollection
{
    public ArrayList SomeStrings
    {
        get;

        // This set accessor violates rule CA2227.
        // To fix the code, remove this set accessor or change it to init.
        set;
    }

    public WritableCollection()
    {
        SomeStrings = new ArrayList(new string[] { "one", "two", "three" });
    }
}

class ReplaceWritableCollection
{
    static void Main2227()
    {
        ArrayList newCollection = new ArrayList(new string[] { "a", "new", "collection" });

        WritableCollection collection = new WritableCollection();

        // This line of code demonstrates how the entire collection
        // can be replaced by a property that's not read only.
        collection.SomeStrings = newCollection;

        // If the intent is to replace an entire collection,
        // implement and/or use the Clear() and AddRange() methods instead.
        collection.SomeStrings.Clear();
        collection.SomeStrings.AddRange(newCollection);
    }
}
```

```

Public Class WritableCollection

    ' This property violates rule CA2227.
    ' To fix the code, add the ReadOnly modifier to the property:
    ' ReadOnly Property SomeStrings As ArrayList
    Property SomeStrings As ArrayList

    Sub New()
        SomeStrings = New ArrayList(New String() {"one", "two", "three"})
    End Sub

End Class

Class ViolatingVersusPreferred

    Shared Sub Main2227()
        Dim newCollection As New ArrayList(New String() {"a", "new", "collection"})

        Dim collection As New WritableCollection()

        ' This line of code demonstrates how the entire collection
        ' can be replaced by a property that's not read only.
        collection.SomeStrings = newCollection

        ' If the intent is to replace an entire collection,
        ' implement and/or use the Clear() and AddRange() methods instead.
        collection.SomeStrings.Clear()
        collection.SomeStrings.AddRange(newCollection)
    End Sub

End Class

```

Related rules

- [CA1819: Properties should not return arrays](#)

See also

- [Properties in C#](#)

CA2229: Implement serialization constructors

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA2229
Category	Usage
Fix is breaking or non-breaking	Non-breaking

Cause

The type implements the `System.Runtime.Serialization.ISerializable` interface, is not a delegate or interface, and one of the following conditions is true:

- The type does not have a constructor that takes a `SerializationInfo` object and a `StreamingContext` object (the signature of the serialization constructor).
- The type is unsealed and the access modifier for its serialization constructor is not protected (family).
- The type is sealed and the access modifier for its serialization constructor is not private.

Rule description

This rule is relevant for types that support custom serialization. A type supports custom serialization if it implements the `ISerializable` interface. The serialization constructor is required to deserialize, or recreate, objects that have been serialized using the `ISerializable.GetObjectData` method.

How to fix violations

To fix a violation of this rule, implement the serialization constructor. For a sealed class, make the constructor private; otherwise, make it protected.

When to suppress warnings

Do not suppress a violation of the rule. The type will not be deserializable, and will not function in many scenarios.

Example

The following example shows a type that satisfies the rule.

```
[Serializable]
public class SerializationConstructorsRequired : ISerializable
{
    private int n1;

    // This is a regular constructor.
    public SerializationConstructorsRequired()
    {
        n1 = -1;
    }
    // This is the serialization constructor.
    // Satisfies rule: ImplementSerializationConstructors.

    protected SerializationConstructorsRequired(
        SerializationInfo info,
        StreamingContext context)
    {
        n1 = (int)info.GetValue(nameof(n1), typeof(int));
    }

    // The following method serializes the instance.
    void ISerializable.GetObjectData(SerializationInfo info,
        StreamingContext context)
    {
        info.AddValue(nameof(n1), n1);
    }
}
```

Related rules

[CA2237: Mark ISerializable types with SerializableAttribute](#)

See also

- [System.Runtime.Serialization.ISerializable](#)
- [System.Runtime.Serialization.SerializationInfo](#)
- [System.Runtime.Serialization.StreamingContext](#)

CA2231: Overload operator equals on overriding ValueType.Equals

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA2231
Category	Usage
Fix is breaking or non-breaking	Non-breaking

Cause

A value type overrides [System.Object.Equals](#) but does not implement the equality operator.

By default, this rule only looks at externally visible types, but this is [configurable](#).

Rule description

In most programming languages, there is no default implementation of the equality operator (==) for value types. If your programming language supports operator overloads, you should consider implementing the equality operator. Its behavior should be identical to that of [Equals](#).

You cannot use the default equality operator in an overloaded implementation of the equality operator. Doing so will cause a stack overflow. To implement the equality operator, use the Object.Equals method in your implementation. For example:

```
If (Object.ReferenceEquals(left, Nothing)) Then
    Return Object.ReferenceEquals(right, Nothing)
Else
    Return left.Equals(right)
End If
```

```
if (Object.ReferenceEquals(left, null))
    return Object.ReferenceEquals(right, null);
return left.Equals(right);
```

How to fix violations

To fix a violation of this rule, implement the equality operator.

When to suppress warnings

It is safe to suppress a warning from this rule; however, we recommend that you provide the equality operator if possible.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA2231
// The code that's violating the rule is on this line.
#pragma warning restore CA2231
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA2231.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Usage.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

Use the following option to configure which parts of your codebase to run this rule on.

- [Include specific API surfaces](#)

You can configure this option for just this rule, for all rules, or for all rules in this category ([Usage](#)). For more information, see [Code quality rule configuration options](#).

Include specific API surfaces

You can configure which parts of your codebase to run this rule on, based on their accessibility. For example, to specify that the rule should run only against the non-public API surface, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXXX.api_surface = private, internal
```

Example

The following example defines a type that violates this rule:

```
public struct PointWithoutHash
{
    private int x, y;

    public PointWithoutHash(int x, int y)
    {
        this.x = x;
        this.y = y;
    }

    public override string ToString()
    {
        return String.Format("({0},{1})", x, y);
    }

    public int X { get { return x; } }

    public int Y { get { return x; } }

    // Violates rule: OverrideGetHashCodeOnOverridingEquals.
    // Violates rule: OverrideOperatorEqualsOnOverridingValueTypeEquals.
    public override bool Equals(object obj)
    {
        if (obj.GetType() != typeof(PointWithoutHash))
            return false;

        PointWithoutHash p = (PointWithoutHash)obj;
        return ((this.x == p.x) && (this.y == p.y));
    }
}
```

Related rules

- [CA1046: Do not overload operator equals on reference types](#)
- [CA2225: Operator overloads have named alternates](#)
- [CA2226: Operators should have symmetrical overloads](#)

See also

- [System.Object.Equals](#)

CA2234: Pass System.Uri objects instead of strings

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA2234
Category	Usage
Fix is breaking or non-breaking	Non-breaking

Cause

A call is made to a method that has a string parameter whose name contains "uri", "Uri", "urn", "Urn", "url", or "Url" and the declaring type of the method contains a corresponding method overload that has a [System.Uri](#) parameter.

By default, this rule only looks at externally visible methods and types, but this is [configurable](#).

Rule description

A parameter name is split into tokens based on the camel casing convention, and then each token is checked to see whether it equals "uri", "Uri", "urn", "Urn", "url", or "Url". If there is a match, the parameter is assumed to represent a uniform resource identifier (URI). A string representation of a URI is prone to parsing and encoding errors, and can lead to security vulnerabilities. The [Uri](#) class provides these services in a safe and secure manner. When there is a choice between two overloads that differ only regarding the representation of a URI, the user should choose the overload that takes a [Uri](#) argument.

How to fix violations

To fix a violation of this rule, call the overload that takes the [Uri](#) argument.

When to suppress warnings

It is safe to suppress a warning from this rule if the string parameter does not represent a URI.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA2234
// The code that's violating the rule is on this line.
#pragma warning restore CA2234
```

To disable the rule for a file, folder, or project, set its severity to [none](#) in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA2234.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Usage.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Configure code to analyze

Use the following option to configure which parts of your codebase to run this rule on.

- [Include specific API surfaces](#)

You can configure this option for just this rule, for all rules, or for all rules in this category ([Usage](#)). For more information, see [Code quality rule configuration options](#).

Include specific API surfaces

You can configure which parts of your codebase to run this rule on, based on their accessibility. For example, to specify that the rule should run only against the non-public API surface, add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CAXXX.api_surface = private, internal
```

Example

The following example shows a method, `ErrorProne`, that violates the rule and a method, `SaferWay`, that correctly calls the `Uri` overload:

```

Imports System

Namespace ca2234

    Class History

        Friend Sub AddToHistory(uriString As String)
        End Sub

        Friend Sub AddToHistory(uriType As Uri)
        End Sub

    End Class

    Public Class Browser

        Dim uriHistory As New History()

        Sub ErrorProne()
            uriHistory.AddToHistory("http://www.adventure-works.com")
        End Sub

        Sub SaferWay()
            Try
                Dim newUri As New Uri("http://www.adventure-works.com")
                uriHistory.AddToHistory(newUri)
            Catch uriException As UriFormatException
            End Try
        End Sub

    End Class

End Namespace

```

```

class History
{
    internal void AddToHistory(string uriString) { }
    internal void AddToHistory(Uri uriType) { }
}

public class Browser
{
    History uriHistory = new History();

    public void ErrorProne()
    {
        uriHistory.AddToHistory("http://www.adventure-works.com");
    }

    public void SaferWay()
    {
        try
        {
            Uri newUri = new Uri("http://www.adventure-works.com");
            uriHistory.AddToHistory(newUri);
        }
        catch (UriFormatException) { }
    }
}

```

Related rules

- CA1056: URI properties should not be strings

- CA1054: URI parameters should not be strings
- CA1055: URI return values should not be strings

CA2235: Mark all non-serializable fields

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA2235
Category	Usage
Fix is breaking or non-breaking	Non-breaking

Cause

An instance field of a type that is not serializable is declared in a type that is serializable.

Rule description

A serializable type is one that is marked with the [System.SerializableAttribute](#) attribute. When the type is serialized, a [System.Runtime.Serialization.SerializationException](#) exception is thrown if the type contains an instance field of a type that's not serializable *and* doesn't implement the [System.Runtime.Serialization.ISerializable](#) interface.

TIP

CA2235 does not fire for instance fields of types that implement [ISerializable](#) because they provide their own serialization logic.

How to fix violations

To fix a violation of this rule, apply the [System.NonSerializedAttribute](#) attribute to the field that is not serializable.

When to suppress warnings

Only suppress a warning from this rule if a [System.Runtime.Serialization.ISerializationSurrogate](#) type is declared that allows instances of the field to be serialized and deserialized.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA2235
// The code that's violating the rule is on this line.
#pragma warning restore CA2235
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA2235.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Usage.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Example

The following example shows two types: one that violates the rule and one that satisfies the rule.

```
public class Mouse
{
    int buttons;
    string scanTypeValue;

    public int NumberOfButtons
    {
        get { return buttons; }
    }

    public string ScanType
    {
        get { return scanTypeValue; }
    }

    public Mouse(int numberOfButtons, string scanType)
    {
        buttons = numberOfButtons;
        scanTypeValue = scanType;
    }
}

[Serializable]
public class InputDevices1
{
    // Violates MarkAllNonSerializableFields.
    Mouse opticalMouse;

    public InputDevices1()
    {
        opticalMouse = new Mouse(5, "optical");
    }
}

[Serializable]
public class InputDevices2
{
    // Satisfies MarkAllNonSerializableFields.
    [NonSerialized]
    Mouse opticalMouse;

    public InputDevices2()
    {
        opticalMouse = new Mouse(5, "optical");
    }
}
```

```

Imports System
Imports System.Runtime.Serialization

Namespace ca2235

    Public Class Mouse

        ReadOnly Property NumberOfButtons As Integer

        ReadOnly Property ScanType As String

        Sub New(numberOfButtons As Integer, scanType As String)
            Me.NumberOfButtons = numberOfButtons
            Me.ScanType = scanType
        End Sub

    End Class

    <SerializableAttribute>
    Public Class InputDevices1

        ' Violates MarkAllNonSerializableFields.
        Dim opticalMouse As Mouse

        Sub New()
            opticalMouse = New Mouse(5, "optical")
        End Sub

    End Class

    <SerializableAttribute>
    Public Class InputDevices2

        ' Satisfies MarkAllNonSerializableFields.
        <NonSerializedAttribute>
        Dim opticalMouse As Mouse

        Sub New()
            opticalMouse = New Mouse(5, "optical")
        End Sub

    End Class

End Namespace

```

Remarks

Rule CA2235 does not analyze types that implement the [ISerializable](#) interface (unless they are also marked with the [SerializableAttribute](#) attribute). This is because rule CA2237 already recommends marking types that implement the [ISerializable](#) interface with the [SerializableAttribute](#) attribute.

Related rules

- [CA2229: Implement serialization constructors](#)
- [CA2237: Mark ISerializable types with SerializableAttribute](#)

CA2237: Mark ISerializable types with SerializableAttribute

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA2237
Category	Usage
Fix is breaking or non-breaking	Non-breaking

Cause

An externally visible type implements the [System.Runtime.Serialization.ISerializable](#) interface and the type is not marked with the [System.SerializableAttribute](#) attribute. The rule ignores derived types whose base type is not serializable.

Rule description

To be recognized by the common language runtime as serializable, types must be marked with the [SerializableAttribute](#) attribute even if the type uses a custom serialization routine through implementation of the [ISerializable](#) interface.

How to fix violations

To fix a violation of this rule, apply the [SerializableAttribute](#) attribute to the type.

When to suppress warnings

Do not suppress a warning from this rule for exception classes, because they must be serializable to work correctly across application domains.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA2237
// The code that's violating the rule is on this line.
#pragma warning restore CA2237
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.cs,vb]
dotnet_diagnostic.CA2237.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Usage.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Example

The following example shows a type that violates the rule. Uncomment the `SerializableAttribute` attribute line to satisfy the rule.

```
Imports System
Imports System.Runtime.Serialization
Imports System.Security.Permissions

Namespace ca2237

    ' <SerializableAttribute> _
    Public Class BaseType
        Implements ISerializable

        Dim baseValue As Integer

        Sub New()
            baseValue = 3
        End Sub

        Protected Sub New(
            info As SerializationInfo, context As StreamingContext)

            baseValue = info.GetInt32("baseValue")

        End Sub

        <SecurityPermissionAttribute(SecurityAction.Demand,
            SerializationFormatter:=True)>
        Overridable Sub GetObjectData(
            info As SerializationInfo, context As StreamingContext) _
            Implements ISerializable.GetObjectData

            info.AddValue("baseValue", baseValue)

        End Sub

    End Class

End Namespace
```

```
// [SerializableAttribute]
public class BaseType : ISerializable
{
    int baseValue;

    public BaseType()
    {
        baseValue = 3;
    }

    protected BaseType(
        SerializationInfo info, StreamingContext context)
    {
        baseValue = info.GetInt32("baseValue");
    }

    public virtual void GetObjectData(
        SerializationInfo info, StreamingContext context)
    {
        info.AddValue("baseValue", baseValue);
    }
}
```

Related rules

- [CA2229: Implement serialization constructors](#)
- [CA2235: Mark all non-serializable fields](#)

CA2241: Provide correct arguments to formatting methods

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA2241
Category	Usage
Fix is breaking or non-breaking	Non-breaking

Cause

The `format` string argument passed to a method such as `WriteLine`, `Write`, or `System.String.Format` does not contain a format item that corresponds to each object argument, or vice versa.

By default, this rule only analyzes calls to the three methods mentioned previously, but this is [configurable](#).

Rule description

The arguments to methods such as `WriteLine`, `Write`, and `Format` consist of a format string followed by several `System.Object` instances. The format string consists of text and embedded format items of the form

`{index[,alignment][:formatString]}`. 'index' is a zero-based integer that indicates which of the objects to format. If an object does not have a corresponding index in the format string, the object is ignored. If the object specified by 'index' does not exist, a `System.FormatException` is thrown at run time.

How to fix violations

To fix a violation of this rule, provide a format item for each object argument and provide an object argument for each format item.

When to suppress warnings

Do not suppress a warning from this rule.

Configure code to analyze

Use the following options to configure additional methods to run this rule on.

- [Additional string formatting methods](#)
- [Determine additional string formatting methods automatically](#)

Additional string formatting methods

You can configure names of additional string formatting methods which should be analyzed by this rule. For example, to specify all methods named `MyFormat` as string formatting methods, you can add the following key-value pair to an `.editorconfig` file in your project:

```
dotnet_code_quality.CA2241.additional_string_formatting_methods = MyFormat
```

Allowed method name formats in the option value (separated by `|`):

- Method name only (includes all methods with the name, regardless of the containing type or namespace)
- Fully qualified names in the symbol's [documentation ID format](#), with an optional `M:` prefix.

Examples:

OPTION VALUE	SUMMARY
<code>dotnet_code_quality.CA2241.additional_string_formatting_methods = MyFormat</code>	Matches all methods named <code>MyFormat</code> in the compilation
<code>dotnet_code_quality.CA2241.additional_string_formatting_methods = MyFormat1 MyFormat2</code>	Matches all methods named either <code>MyFormat1</code> or <code>MyFormat2</code> in the compilation
<code>dotnet_code_quality.CA2241.additional_string_formatting_methods = NS.MyType.MyFormat(ParamType)</code>	Matches specific method <code>MyFormat</code> with given fully qualified signature
<code>dotnet_code_quality.CA2241.additional_string_formatting_methods = NS1.MyType1.MyFormat1(ParamType) NS2.MyType2.MyFormat2(ParamType)</code>	Matches specific methods <code>MyFormat1</code> and <code>MyFormat2</code> with respective fully qualified signature

Determine additional string formatting methods automatically

Instead of specifying an explicit list of additional string formatting methods, you can configure the analyzer to automatically attempt to determine the string formatting method. By default, this option is disabled. If the option is enabled, any method that has a `string format` parameter followed by a `params object[]` parameter is considered a string formatting method:

```
dotnet_code_quality.CA2241.try_determine_additional_string_formatting_methods_automatically = true
```

Example

The following example shows two violations of the rule.

```
Imports System

Namespace ca2241

    Class CallsStringFormat

        Sub CallFormat()

            Dim file As String = "file name"
            Dim errors As Integer = 13

            ' Violates the rule.
            Console.WriteLine(String.Format("{0}", file, errors))

            Console.WriteLine(String.Format("{0}: {1}", file, errors))

            ' Violates the rule and generates a FormatException at runtime.
            Console.WriteLine(String.Format("{0}: {1}, {2}", file, errors))

        End Sub

    End Class

End Namespace
```

```
class CallsStringFormat
{
    void CallFormat()
    {
        string file = "file name";
        int errors = 13;

        // Violates the rule.
        Console.WriteLine(string.Format("{0}", file, errors));

        Console.WriteLine(string.Format("{0}: {1}", file, errors));

        // Violates the rule and generates a FormatException at runtime.
        Console.WriteLine(string.Format("{0}: {1}, {2}", file, errors));
    }
}
```

CA2242: Test for NaN correctly

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA2242
Category	Usage
Fix is breaking or non-breaking	Non-breaking

Cause

An expression tests a value against [System.Single.NaN](#) or [System.Double.NaN](#).

Rule description

[System.Double.NaN](#), which represents a value that's not a number, results when an arithmetic operation is undefined. Any expression that tests for equality between a value and [System.Double.NaN](#) always returns `false`. Any expression that tests for inequality (`!=` in C#) between a value and [System.Double.NaN](#) always returns `true`.

How to fix violations

To fix a violation of this rule and accurately determine whether a value represents [System.Double.NaN](#), use [System.Single.IsNaN](#) or [System.Double.IsNaN](#) to test the value.

When to suppress warnings

Do not suppress a warning from this rule.

Example

The following example shows two expressions that incorrectly test a value against [System.Double.NaN](#) and an expression that correctly uses [System.Double.IsNaN](#) to test the value.

```
Imports System

Namespace ca2242

    Class NaNTests

        Shared zero As Double

        Shared Sub Main2242()
            Console.WriteLine(0 / zero = Double.NaN)
            Console.WriteLine(0 / zero <> Double.NaN)
            Console.WriteLine(Double.IsNaN(0 / zero))
        End Sub

    End Class

End Namespace
```

```
class NaNTests
{
    static double zero = 0;

    static void Main()
    {
        Console.WriteLine(0 / zero == double.NaN);
        Console.WriteLine(0 / zero != double.NaN);
        Console.WriteLine(double.IsNaN(0 / zero));
    }
}
```

CA2243: Attribute string literals should parse correctly

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA2243
Category	Usage
Fix is breaking or non-breaking	Non-breaking

Cause

An attribute's string literal parameter does not parse correctly for a URL, GUID, or Version.

Rule description

Since attributes are derived from [System.Attribute](#), and attributes are used at compile time, only constant values can be passed to their constructors. Attribute parameters that must represent URLs, GUIDs, and Versions cannot be typed as [System.Uri](#), [System.Guid](#), and [System.Version](#), because these types cannot be represented as constants. Instead, they must be represented by strings.

Because the parameter is typed as a string, it is possible that an incorrectly formatted parameter could be passed at compile time.

This rule uses a naming heuristic to find parameters that represent a uniform resource identifier (URI), a Globally Unique Identifier (GUID), or a Version, and verifies that the passed value is correct.

How to fix violations

Change the parameter string to a correctly formed URL, GUID, or Version.

When to suppress warnings

It is safe to suppress a warning from this rule if the parameter does not represent a URL, GUID, or Version.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA2243
// The code that's violating the rule is on this line.
#pragma warning restore CA2243
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA2243.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Usage.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Example

The following example shows code for the `AssemblyFileVersionAttribute` that violates this rule.

```
[AttributeUsage(AttributeTargets.Assembly, Inherited = false)]
[ComVisible(true)]
public sealed class AssemblyFileVersionAttribute : Attribute
{
    public AssemblyFileVersionAttribute(string version) { }

    public string Version { get; set; }
}

// Since the parameter is typed as a string, it is possible
// to pass an invalid version number at compile time. The rule
// would be violated by the following code: [assembly : AssemblyFileVersion("xxxxx")]
```

The rule is triggered by the following parameters:

- Parameters that contain 'version' and cannot be parsed to `System.Version`.
- Parameters that contain 'guid' and cannot be parsed to `System.Guid`.
- Parameters that contain 'uri', 'urn', or 'url' and cannot be parsed to `System.Uri`.

See also

- [CA1054: URI parameters should not be strings](#)

CA2244: Do not duplicate indexed element initializations

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA2244
Category	Usage
Fix is breaking or non-breaking	Non-breaking

Cause

An object initializer has more than one indexed element initializer with the same constant index. All but the last initializer are redundant.

Rule description

[Object initializers](#) let you assign values to any accessible fields or properties of an object at creation time without having to invoke a constructor followed by lines of assignment statements.

Indexed element initializers in objects initializers must initialize unique elements. A duplicate index will overwrite a previous element initialization.

How to fix violations

To fix violations, remove all the redundant indexed element initializers that are overwritten by any of the subsequent element initializer(s). For example, the following code snippet shows a violation of the rule and couple of potential fixes:

```
using System.Collections.Generic;

class C
{
    public void M()
    {
        var dictionary = new Dictionary<int, int>
        {
            [1] = 1, // CA2244
            [2] = 2,
            [1] = 3
        };
    }
}
```

```
using System.Collections.Generic;

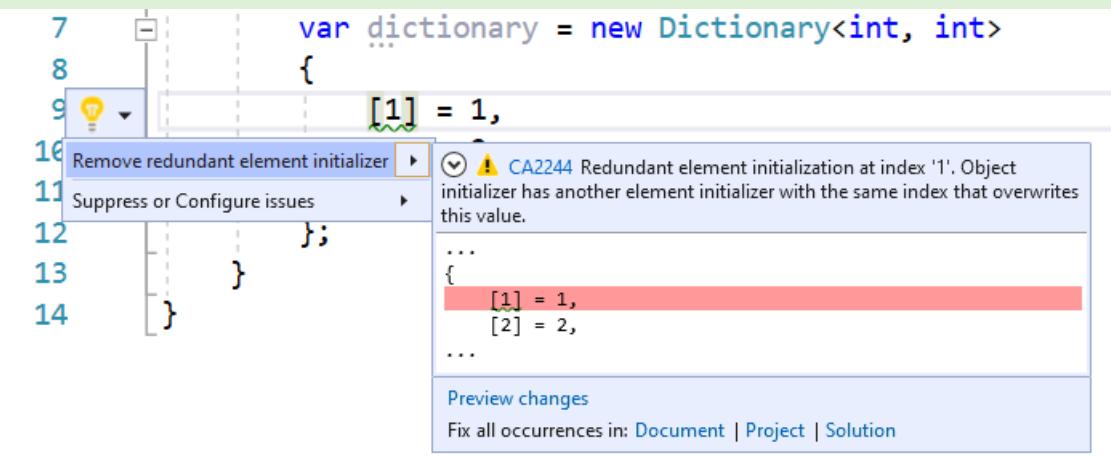
class C
{
    public void M()
    {
        var dictionary = new Dictionary<int, int>
        {
            [2] = 2,
            [1] = 3
        };
    }
}
```

```
using System.Collections.Generic;

class C
{
    public void M()
    {
        var dictionary = new Dictionary<int, int>
        {
            [1] = 1,
            [2] = 2
        };
    }
}
```

TIP

A code fix is available for this rule in Visual Studio. To use it, position the cursor on the violation and press **Ctrl+.** (period). Choose **Remove redundant element initializer** from the list of options that's presented.



When to suppress warnings

Do not suppress violations for this rule.

See also

- [Usage rules](#)

CA2245: Do not assign a property to itself

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA2245
Category	Usage
Fix is breaking or non-breaking	Non-breaking

Cause

A property was accidentally assigned to itself.

Rule description

C# compiler generates a warning [CS1717: Assignment made to same variable; did you mean to assign something else?](#) when a field, local or parameter symbol is assigned to itself. Such a mistake is common when a local, parameter, or field symbol has a name similar to another symbol in scope. Instead of using different symbols on the left-hand and right-hand side of the assignment, the same symbol was used on both sides. This leads to a redundant assignment of the value to itself and generally indicates a functional bug.

Assigning a property to itself is also a similar functional bug for almost all real world cases. However, in some extreme corner cases, fetching a property value can have side effects and the property's new value is different from the original value. If so, property self-assignment is not redundant and cannot be removed. This prevents the compiler from generating a [CS1717](#) warning for property self-assignment, without introducing a breaking change for these cases.

Rule [CA2245](#) aims at filling this gap. It reports the violation for property self-assignment to help fix these functional bugs. For the small set of corner cases where property self-assignment is desirable, [CA2245](#) violations can be suppressed in source with an appropriate justification comment.

How to fix violations

To fix violations, use different symbols on the left-hand and the right-hand side of the assignment. For example, the following code snippet shows a violation of the rule and how to fix it:

```
public class C
{
    private int p = 0;
    public int P { get; private set; }

    public void M(int p)
    {
        // CS1717: Accidentally assigned the parameter 'p' to itself.
        p = p;

        // CA2245: Accidentally assigned the property 'P' to itself.
        P = P;
    }
}
```

```
public class C
{
    private int p = 0;
    public int P { get; private set; }

    public void M(int p)
    {
        // No violation, now the parameter is assigned to the field.
        this.p = p;

        // No violation, now the parameter is assigned to the property.
        P = p;
    }
}
```

When to suppress warnings

It is safe to suppress violations from this rule if fetching a property value can have side effects and the property's new value is different from the original value. If so, property self-assignment is not redundant. A justification comment should be added to the suppression to document this as expected behavior.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA2245
// The code that's violating the rule is on this line.
#pragma warning restore CA2245
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA2245.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Usage.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

Related rules

- [CS1717: Assignment made to same variable; did you mean to assign something else?](#)
- [CA2011: Do not assign property within its setter](#)
- [CA2246: Do not assign a symbol and its member in the same statement](#)

See also

- [Usage rules](#)

CA2246: Do not assign a symbol and its member in the same statement

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA2246
Category	Usage
Fix is breaking or non-breaking	Non-breaking

Cause

A symbol and its member were assigned in the same statement. For example:

```
// 'a' and 'a.Field' are assigned in the same statement
a.Field = a = b;
```

Rule description

Assigning a symbol and its member, that is, a field or a property, in the same statement is not recommended. It is not clear if the member access was intended to use the symbol's old value prior to the assignment or the new value from the assignment in this statement. For clarity, the multi-assign statement must be split into two or more simple assignment statements.

How to fix violations

To fix violations, split the multi-assign statement into two or more simple assignment statements. For example, the following code snippet shows a violation of the rule and a couple of ways to fix it based on the user intent:

```
public class C
{
    public C Field;
}

public class Test
{
    public void M(C a, C b)
    {
        // Let us assume 'a' points to 'Instance1' and 'b' points to 'Instance2' at the start of the method.
        // It is not clear if the user intent in the below statement is to assign to 'Instance1.Field' or
        'Instance2.Field'.
        // CA2246: Symbol 'a' and its member 'Field' are both assigned in the same statement. You are at
        risk of assigning the member of an unintended object.
        a.Field = a = b;
    }
}
```

```
public class C
{
    public C Field;
}

public class Test
{
    public void M(C a, C b)
    {
        // Let us assume 'a' points to 'Instance1' and 'b' points to 'Instance2' at the start of the method.
        // 'Instance1.Field' is intended to be assigned.
        var instance1 = a;
        a = b;
        instance1.Field = a;
    }
}
```

```
public class C
{
    public C Field;
}

public class Test
{
    public void M(C a, C b)
    {
        // Let us assume 'a' points to 'Instance1' and 'b' points to 'Instance2' at the start of the method.
        // 'Instance2.Field' is intended to be assigned.
        a = b;
        b.Field = a; // or 'a.Field = a;'
    }
}
```

When to suppress warnings

Do not suppress violations from this rule.

Related rules

- [CA2245: Do not assign a property to itself](#)

See also

- [Usage rules](#)

CA2247: Argument passed to TaskCompletionSource constructor should be TaskCreationOptions enum instead of TaskContinuationOptions enum

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA2247
Category	Usage
Fix is breaking or non-breaking	Non-breaking

Cause

Constructing a `System.Threading.Tasks.TaskCompletionSource` with a `System.Threading.Tasks.TaskContinuationOptions` enum value rather than a `System.Threading.Tasks.TaskCreationOptions` enum value. Using `System.Object.ReferenceEquals` method to test one or more value types for equality.

Rule description

The `TaskCompletionSource` type has a constructor that accepts a `System.Threading.Tasks.TaskCreationOptions` enum value, and another constructor that accepts a `Object`. Accidentally passing a `System.Threading.Tasks.TaskContinuationOptions` enum value instead of a `System.Threading.Tasks.TaskCreationOptions` enum value will result in calling the `Object`-based constructor: it will compile and run, but it will not have the intended behavior.

How to fix violations

To fix the violation, replace the `System.Threading.Tasks.TaskContinuationOptions` enum value with the corresponding `System.Threading.Tasks.TaskCreationOptions` enum value.

```
// Violation
var tcs = new TaskCompletionSource<int>(TaskContinuationOptions.RunContinuationsAsynchronously);

// Fixed
var tcs = new TaskCompletionSource<int>(TaskCreationOptions.RunContinuationsAsynchronously);
```

When to suppress warnings

A violation of this rule almost always highlights a bug in the calling code, such that the code will not behave as the developer intended, with the `TaskCompletionSource` effectively ignoring the specified option. The only time it is safe to suppress the warning is if the developer actually intended to pass a boxed `System.Threading.Tasks.TaskContinuationOptions` as the object state argument to the `TaskCompletionSource`.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA2247
// The code that's violating the rule is on this line.
#pragma warning restore CA2247
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.cs,vb]
dotnet_diagnostic.CA2247.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.cs,vb]
dotnet_analyzer_diagnostic.category-Usage.severity = none
```

For more information, see [How to suppress code analysis warnings..](#)

See also

- [Usage rules](#)

CA2248: Provide correct enum argument to `Enum.HasFlag`

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA2248
Category	Usage
Fix is breaking or non-breaking	Non-breaking

Cause

The enum type passed as an argument to the `HasFlag` method call is different from the calling enum type.

Rule description

The `Enum.HasFlag` method expects the `enum` argument to be of the same `enum` type as the instance on which the method is invoked. If these are different `enum` types, an unhandled exception will be thrown at run time.

How to fix violations

To fix violations, use the same enum type on both the argument and the caller:

```
public class C
{
    [Flags]
    public enum MyEnum { A, B, }

    [Flags]
    public enum OtherEnum { A, }

    public void Method(MyEnum m)
    {
        m.HasFlag(OtherEnum.A); // Enum types are different, this call will cause an `ArgumentException` to
        // be thrown at run time

        m.HasFlag(MyEnum.A); // Valid call
    }
}
```

When to suppress warnings

Do not suppress violations from this rule.

CA2249: Consider using String.Contains instead of String.IndexOf

9/20/2022 • 3 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA2249
Category	Usage
Fix is breaking or non-breaking	Non-breaking

Cause

This rule locates calls to `IndexOf` where the result is used to check for the presence or absence of a substring, and suggests using `Contains` instead, to improve readability.

Rule description

When `IndexOf` is used to check if the result is equal to `-1` or greater or equal than `0`, the call can be safely substituted with `Contains` without an impact on performance.

Depending on the `IndexOf` overload being used, the suggested fix could get a `comparisonType` argument added:

OVERLOAD	SUGGESTED FIX
<code>String.IndexOf(char)</code>	<code>String.Contains(char)</code>
<code>String.IndexOf(string)</code>	<code>String.Contains(string, StringComparison.CurrentCulture)</code>
<code>String.IndexOf(char, StringComparison.OrdinalIgnoreCase)</code>	<code>String.Contains(char)</code>
<code>String.IndexOf(string, StringComparison.OrdinalIgnoreCase)</code>	<code>String.Contains(string)</code>
<code>String.IndexOf(char, NON StringComparison.OrdinalIgnoreCase) *</code>	<code>String.Contains(char, NON StringComparison.OrdinalIgnoreCase)</code> *
<code>String.IndexOf(string, NON StringComparison.OrdinalIgnoreCase)</code> *	<code>String.Contains(string, NON StringComparison.OrdinalIgnoreCase)</code> *

* Any `StringComparison` enum value other than `StringComparison.OrdinalIgnoreCase`:

- `CurrentCulture`
- `CurrentCultureIgnoreCase`
- `InvariantCulture`
- `InvariantCultureIgnoreCase`
- `OrdinalIgnoreCase`

How to fix violations

The violation can either be fixed manually, or, in some cases, using Quick Actions to fix code in Visual Studio.

Examples

The following two code snippets show all possible violations of the rule in C# and how to fix them:

```
using System;

class MyClass
{
    void MyMethod()
    {
        string str = "My text";
        bool found;

        // No comparisonType in char overload, so no comparisonType added in resulting fix
        found = str.IndexOf('x') == -1;
        found = str.IndexOf('x') >= 0;

        // No comparisonType in string overload, adds StringComparison.CurrentCulture to resulting fix
        found = str.IndexOf("text") == -1;
        found = str.IndexOf("text") >= 0;

        // comparisonType equal to StringComparison.Ordinal, removes the argument
        found = str.IndexOf('x', StringComparison.Ordinal) == -1;
        found = str.IndexOf('x', StringComparison.Ordinal) >= 0;

        found = str.IndexOf("text", StringComparison.Ordinal) == -1;
        found = str.IndexOf("text", StringComparison.Ordinal) >= 0;

        // comparisonType different than StringComparison.Ordinal, preserves the argument
        found = str.IndexOf('x', StringComparison.OrdinalIgnoreCase) == -1;
        found = str.IndexOf('x', StringComparison.CurrentCulture) >= 0;

        found = str.IndexOf("text", StringComparison.InvariantCultureIgnoreCase) == -1;
        found = str.IndexOf("text", StringComparison.InvariantCulture) >= 0;

        // Suggestion message provided, but no automatic fix offered, must be fixed manually
        int index = str.IndexOf("text");
        if (index == -1)
        {
            Console.WriteLine("'text' Not found.");
        }
    }
}
```

```

using System;

class MyClass
{
    void MyMethod()
    {
        string str = "My text";
        bool found;

        // No comparisonType in char overload, so no comparisonType added in resulting fix
        found = !str.Contains('x');
        found = str.Contains('x');

        // No comparisonType in string overload, adds StringComparison.CurrentCulture to resulting fix
        found = !string.Contains("text", StringComparison.CurrentCulture);
        found = string.Contains("text", StringComparison.CurrentCulture);

        // comparisonType equal to StringComparison.Ordinal, removes the argument
        found = !string.Contains('x');
        found = string.Contains('x');

        found = !string.Contains("text");
        found = string.Contains("text");

        // comparisonType different than StringComparison.Ordinal, preserves the argument
        ;found = !string.Contains('x', StringComparison.OrdinalIgnoreCase)
        found = string.Contains('x', StringComparison.CurrentCulture);

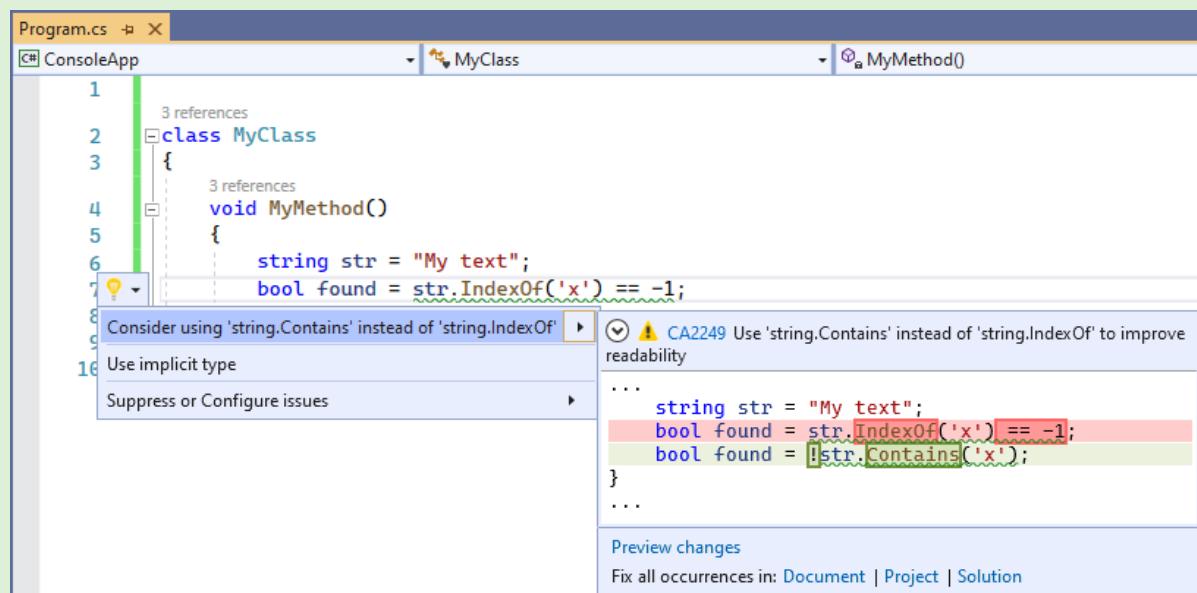
        found = !string.Contains("text", StringComparison.InvariantCultureIgnoreCase);
        found = string.Contains("text", StringComparison.InvariantCulture);

        // This case had to be manually fixed
        if (!str.Contains("text"))
        {
            Console.WriteLine("'text' Not found.");
        }
    }
}

```

TIP

A code fix is available for this rule in Visual Studio. To use it, position the cursor on the violation and press **Ctrl+.** (period). Choose **Consider using 'string.Contains' instead of 'string.IndexOf'** from the list of options that's presented.



When to suppress warnings

It's safe to suppress a violation of this rule if improving code readability is not a concern.

SUPPRESS A WARNING

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA2249
// The code that's violating the rule is on this line.
#pragma warning restore CA2249
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA2249.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Usage.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

SEE ALSO

- [Usage rules](#)

CA2250: Use `ThrowIfCancellationRequested`

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA2250
Category	Usage
Fix is breaking or non-breaking	Non-breaking

Cause

This rule flags conditional statements that check `IsCancellationRequested` before throwing `OperationCanceledException`.

Rule description

You can accomplish the same thing by calling `CancellationToken.ThrowIfCancellationRequested()`.

How to fix violations

To fix violations, replace the conditional statement with a call to `ThrowIfCancellationRequested()`.

```
using System;
using System.Threading;

public void MySlowMethod(CancellationToken token)
{
    // Violation
    if (token.IsCancellationRequested)
        throw new OperationCanceledException();

    // Fix
    token.ThrowIfCancellationRequested();

    // Violation
    if (token.IsCancellationRequested)
        throw new OperationCanceledException();
    else
        DoSomethingElse();

    // Fix
    token.ThrowIfCancellationRequested();
    DoSomethingElse();
}
```

```
Imports System
Imports System.Threading

Public Sub MySlowMethod(token As CancellationToken)

    ' Violation
    If token.IsCancellationRequested Then
        Throw New OperationCanceledException()
    End If

    ' Fix
    token.ThrowIfCancellationRequested()

    ' Violation
    If token.IsCancellationRequested Then
        Throw New OperationCanceledException()
    Else
        DoSomethingElse()
    End If

    ' Fix
    token.ThrowIfCancellationRequested()
    DoSomethingElse()
End Sub
```

When to suppress warnings

It is safe to suppress warnings from this rule.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA2250
// The code that's violating the rule is on this line.
#pragma warning restore CA2250
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA2250.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Usage.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Usage Warnings](#)

CA2251: Use `String.Equals` over `String.Compare`

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA2251
Category	Usage
Fix is breaking or non-breaking	Non-breaking

Cause

The result of a call to `String.Compare` is compared to zero.

Rule description

`String.Compare` is designed to produce a total-order comparison that can be used for sorting. If you only care whether the strings are equal, it is both clearer and likely faster to use an equivalent overload of `String.Equals`.

How to fix violations

To fix violations of this rule, replace the expression comparing the result of `String.Compare` with a call to `String.Equals`.

When to suppress warnings

It is safe to suppress warnings from this rule.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA2251
// The code that's violating the rule is on this line.
#pragma warning restore CA2251
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA2251.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Usage.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Performance warnings](#)

CA2252: Opt in to preview features before using them

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA2252
Category	Usage
Fix is breaking or non-breaking	Non-breaking

Cause

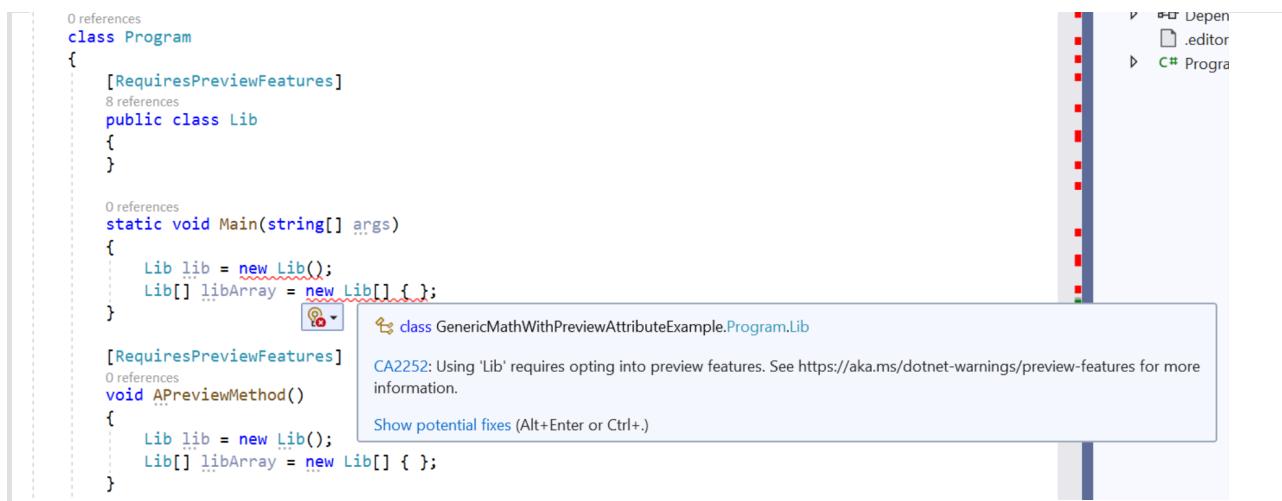
A client uses preview APIs or types in their assembly without explicitly opting in either locally or at the module or assembly level.

Rule description

When an API or assembly that's decorated with the [RequiresPreviewFeaturesAttribute](#) attribute is consumed, this rule checks if the call site has opted in to preview features. A call site has opted in to preview features if one of the following applies:

- It is within the scope of a `RequiresPreviewFeaturesAttribute` annotation.
- It is part of an assembly or module that has already opted in to preview features.

The following image shows an example of the CA2252 diagnostic.



Here, `Lib` is a preview type that's constructed in the `Main` method. `Main` itself is not annotated as a preview method, so diagnostics are produced on the two constructors calls inside `Main`.

How to fix violations

There are two ways to fix violations:

- Bring a call site within the scope of an annotation by annotating its parent with `RequiresPreviewFeaturesAttribute`. In the previous example, `APreviewMethod` is annotated with the `RequiresPreviewFeatures` attribute, so the analyzer ignores preview type usage inside `APreviewMethod`. It follows that callers of `APreviewMethod` will have to perform a similar exercise.
- You can also opt in to preview features at an assembly or module level. This indicates to the analyzer that preview type usage in the assembly is desired and, as a consequence, no errors will be produced by this rule. This is the preferred way to consume preview dependencies. To enable preview features inside the entire assembly, set the `EnablePreviewFeatures` property in a `.csproj` file:

```
<PropertyGroup>
  <EnablePreviewFeatures>true</EnablePreviewFeatures>
</PropertyGroup>
```

When to suppress warnings

Suppressing warnings from this rule is only recommended for advanced use cases where diagnostics on APIs need to explicitly disabled. In this case, you must be willing to take on the responsibility of marking preview APIs appropriately. For example, consider a case where an existing type implements a new preview interface. Since the entire type cannot be marked as preview (for backwards compatibility), the diagnostic around the type definition can be disabled locally. Further, you need to mark the preview interface implementations as preview. Now, the existing type can be used as before, but calls to the new interface methods will get diagnostics.

`System.Private.CoreLib.csproj` uses this technique to expose generic math features on numeric types such as `Int32`, `Double`, and `Decimal`.

The following images show how to disable the CA2252 analyzer locally.

```
public readonly struct Int32 : IComparable, IConvertible, ISpanFormattable, IComparable<int>, IEquatable<int>
#if FEATURE_GENERIC_MATH
#pragma warning disable SA1001, CA2252 // SA1001: Comma positioning; CA2252: Preview Features
    , IBinaryInteger<int>,
    IMin.MaxValue<int>,
    ISignedNumber<int>
#endif // FEATURE_GENERIC_MATH
```

```
//  
// IBinaryInteger  
  
[RequiresPreviewFeatures]  
static int IBinaryInteger<int>.LeadingZeroCount(int value)  
    => BitOperations.LeadingZeroCount((uint)value);  
  
[RequiresPreviewFeatures]  
static int IBinaryInteger<int>.PopCount(int value)  
    => BitOperations.PopCount((uint)value);  
  
[RequiresPreviewFeatures]  
static int IBinaryInteger<int>.RotateLeft(int value, int rotateAmount)  
    => (int)BitOperations.RotateLeft((uint)value, rotateAmount);  
  
[RequiresPreviewFeatures]  
static int IBinaryInteger<int>.RotateRight(int value, int rotateAmount)  
    => (int)BitOperations.RotateRight((uint)value, rotateAmount);  
  
[RequiresPreviewFeatures]  
static int IBinaryInteger<int>.TrailingZeroCount(int value)  
    => BitOperations.TrailingZeroCount(value);
```

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA2252  
// The code that's violating the rule is on this line.  
#pragma warning restore CA2252
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]  
dotnet_diagnostic.CA2252.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]  
dotnet_analyzer_diagnostic.category-Usage.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [EnablePreviewFeatures and GenerateRequiresPreviewFeaturesAttribute](#)
- [Preview feature design document](#)

CA2253: Named placeholders should not be numeric values

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA2253
Category	Usage
Fix is breaking or non-breaking	Non-breaking

Cause

A message placeholder consists of numeric characters only.

Rule description

Named placeholders in the logging message template should not be comprised of only numeric characters.

How to fix violations

Rename the numeric placeholder.

For usage examples, see the [LoggerExtensions.LogInformation](#) method.

When to suppress errors

Do not suppress a warning from this rule.

See also

- [Usage warnings](#)

CA2254: Template should be a static expression

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA2254
Category	Usage
Fix is breaking or non-breaking	Non-breaking

Cause

A message template passed to a logger API is not constant.

Rule description

When performing logging, it's desirable to preserve the structure of the log (including placeholder names) along with the placeholder values. Preserving this information allows for better observability and search in log aggregation and monitoring software.

Preferred:

```
var firstName = "Lorenz";
var lastName = "Otto";

// This tells the logger that there are FirstName and LastName properties
// on the log message, and correlates them with the argument values.
logger.Warning("Person {FirstName} {LastName} encountered an issue", firstName, lastName);
```

Not preferred:

```
// DO NOT DO THIS

var firstName = "Lorenz";
var lastName = "Otto";

// Here, the log template itself is changing, and the association between named placeholders and their
// values is lost.
logger.Warning("Person " + firstName + " " + lastName + " encountered an issue");

// String interpolation also loses the association between placeholder names and their values.
logger.Warning($"Person {firstName} {lastName} encountered an issue");
```

The logging message template should not vary between calls.

How to fix violations

Update the message template to be a constant expression. If you're using values directly in the template, refactor them to use named placeholders instead.

```
logger.Warning("Person {FirstName} {LastName} encountered an issue", firstName, lastName);
```

When to suppress errors

Do not suppress a warning from this rule.

See also

- [Usage warnings](#)

CA2255: The `ModuleInitializer` attribute should not be used in libraries

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA2255
Category	Usage
Fix is breaking or non-breaking	Non-breaking

Cause

Applying `ModuleInitializerAttribute` to a method within a Class Library.

Rule description

Module initializers are intended to be used by application code to ensure an application's components are initialized before the application code begins executing. If library code declares a method with the `ModuleInitializerAttribute`, it can interfere with application initialization and also lead to limitations in that application's trimming abilities. Library code should therefore not utilize the `ModuleInitializerAttribute` attribute.

How to fix violations

Instead of using methods marked with `ModuleInitializerAttribute`, the library should expose methods that can be used to initialize any components within the library and allow the application to invoke the method during application initialization.

When to suppress warnings

It is safe to suppress warnings from this rule if a solution uses a Class Library for code factoring purposes, and the `ModuleInitializerAttribute` method is not part of a shared or distributed library or package.

Suppress a warning

If you just want to suppress a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable CA2255
// The code that's violating the rule is on this line.
#pragma warning restore CA2255
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.CA2255.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Usage.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Usage warnings](#)

CA2256: All members declared in parent interfaces must have an implementation in a DynamicInterfaceCastableImplementation-attributed interface

9/20/2022 • 2 minutes to read • [Edit Online](#)

	Value
Rule ID	CA2256
Category	Usage
Fix is breaking or non-breaking	Non-breaking

Cause

An interface with [DynamicInterfaceCastableImplementationAttribute](#) has a non-implemented member.

Rule description

Types attributed with [DynamicInterfaceCastableImplementationAttribute](#) act as an interface implementation for a type that implements the `IDynamicInterfaceCastable` type. As a result, it must provide an implementation of all of the members defined in the inherited interfaces, because the type that implements `IDynamicInterfaceCastable` will not provide them otherwise.

How to fix violations

Implement the missing interface members.

When to suppress errors

Do not suppress a warning from this rule.

See also

- [Usage warnings](#)

CA2257: Members defined on an interface with the 'DynamicInterfaceCastableImplementationAttribute' should be 'static'

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA2257
Category	Usage
Fix is breaking or non-breaking	Non-breaking

Cause

An interface member isn't explicitly implemented or marked `static`.

Rule description

Since a type that implements `IDynamicInterfaceCastable` may not implement a dynamic interface in metadata, calls to an instance interface member that is not an explicit implementation defined on this type are likely to fail at run time. To avoid run-time errors, mark new interface members `static`.

How to fix violations

Mark the interface member `static`.

When to suppress errors

Do not suppress a warning from this rule.

See also

- [Usage warnings](#)

CA2258: Providing a 'DynamicInterfaceCastableImplementation' interface in Visual Basic is unsupported

9/20/2022 • 2 minutes to read • [Edit Online](#)

	VALUE
Rule ID	CA2258
Category	Usage
Fix is breaking or non-breaking	Non-breaking

Cause

Use of [DynamicInterfaceCastableImplementationAttribute](#) in Visual Basic.

Rule description

Providing a functional interface that's attributed with [DynamicInterfaceCastableImplementationAttribute](#) requires the Default Interface Members feature, which is not supported in Visual Basic.

When to suppress errors

Do not suppress a warning from this rule.

See also

- [Usage warnings](#)

Code-style rules

9/20/2022 • 4 minutes to read • [Edit Online](#)

.NET code-style analysis provides rules that aim to maintain consistent *style* in your codebase. These rules have an "IDE" prefix in the rule ID.

Most of the rules have one or more associated options to customize the preferred style. For example, [Use simple 'using' statement \(IDE0063\)](#) has the associated option `csharp_prefer_simple_using_statement` that lets you define whether you prefer a `using` declaration or a `using` statement. The rule enforces whichever options you choose at a specified level (for example, warning or error).

The code-style rules are organized into the following subcategories:

- [Language rules](#)

Rules that pertain to the C# or Visual Basic language. For example, you can specify rules that regard the use of `var` when defining variables, or whether expression-bodied members are preferred.

- [Unnecessary code rules](#)

Rules that pertain to unnecessary code that indicates a potential readability, maintainability, performance, or functional problem. For example, unreachable code within methods or unused private fields, properties, or methods is unnecessary code.

- [Formatting rules](#)

Rules that pertain to the layout and structure of your code in order to make it easier to read. For example, you can specify a formatting option that defines whether spaces in control blocks are preferred or not.

- [Naming rules](#)

Rules that pertain to the naming of code elements. For example, you can specify that `async` method names must have an "Async" suffix.

- [Miscellaneous rules](#)

Rules that do not belong in other categories.

Index

The following table lists all the code-style rules by ID and [options](#), if any.

RULE ID	TITLE	OPTION
IDE0001	Simplify name	
IDE0002	Simplify member access	
IDE0003	Remove <code>this</code> or <code>Me</code> qualification	<code>dotnet_style_qualification_for_field</code> <code>dotnet_style_qualification_for_property</code> <code>dotnet_style_qualification_for_method</code> <code>dotnet_style_qualification_for_event</code>
IDE0004	Remove unnecessary cast	

RULE ID	TITLE	OPTION
IDE0005	Remove unnecessary import	
IDE0007	Use <code>var</code> instead of explicit type	<code>csharp_style_var_for_builtin_types</code> <code>csharp_style_var_when_type_is_apparent</code> <code>csharp_style_var_elsewhere</code>
IDE0008	Use explicit type instead of <code>var</code>	<code>csharp_style_var_for_builtin_types</code> <code>csharp_style_var_when_type_is_apparent</code> <code>csharp_style_var_elsewhere</code>
IDE0009	Add <code>this</code> or <code>Me</code> qualification	<code>dotnet_style_qualification_for_field</code> <code>dotnet_style_qualification_for_property</code> <code>dotnet_style_qualification_for_method</code> <code>dotnet_style_qualification_for_event</code>
IDE0010	Add missing cases to switch statement	
IDE0011	Add braces	<code>csharp_prefer_braces</code>
IDE0016	Use throw expression	<code>csharp_style_throw_expression</code>
IDE0017	Use object initializers	<code>dotnet_style_object_initializer</code>
IDE0018	Inline variable declaration	<code>csharp_style_inlined_variable_declaration</code>
IDE0019	Use pattern matching to avoid <code>as</code> followed by a <code>null</code> check	<code>csharp_style_pattern_matching_over_as_with_null_check</code>
IDE0020	Use pattern matching to avoid <code>is</code> check followed by a cast (with variable)	<code>csharp_style_pattern_matching_over_is_with_cast_check</code>
IDE0021	Use expression body for constructors	<code>csharp_style_expression_bodied_constructors</code>
IDE0022	Use expression body for methods	<code>csharp_style_expression_bodied_methods</code>
IDE0023	Use expression body for conversion operators	<code>csharp_style_expression_bodied_operators</code>
IDE0024	Use expression body for operators	<code>csharp_style_expression_bodied_operators</code>
IDE0025	Use expression body for properties	<code>csharp_style_expression_bodied_properties</code>
IDE0026	Use expression body for indexers	<code>csharp_style_expression_bodied_indexers</code>

RULE ID	TITLE	OPTION
IDE0027	Use expression body for accessors	csharp_style_expression_bodied_accessors
IDE0028	Use collection initializers	dotnet_style_collection_initializer
IDE0029	Use coalesce expression (non-nullable types)	dotnet_style_coalesce_expression
IDE0030	Use coalesce expression (nullable types)	dotnet_style_coalesce_expression
IDE0031	Use null propagation	dotnet_style_null_propagation
IDE0032	Use auto property	dotnet_style_prefer_auto_properties
IDE0033	Use explicitly provided tuple name	dotnet_style_explicit_tuple_names
IDE0034	Simplify <code>default</code> expression	csharp_prefer_simple_default_expression
IDE0035	Remove unreachable code	
IDE0036	Order modifiers	csharp_preferred_modifier_order visual_basic_preferred_modifier_order
IDE0037	Use inferred member name	dotnet_style_prefer_inferred_tuple_names dotnet_style_prefer_inferred_anonymous_type_member_names
IDE0038	Use pattern matching to avoid <code>is</code> check followed by a cast (without variable)	csharp_style_pattern_matching_over_is_with_cast_check
IDE0039	Use local function instead of lambda	csharp_style_prefer_local_over_anonymous_function
IDE0040	Add accessibility modifiers	dotnet_style_require_accessibility_modifiers
IDE0041	Use <code>is null</code> check	dotnet_style_prefer_is_null_check_over_reference_equality_method
IDE0042	Deconstruct variable declaration	csharp_style_deconstructed_variable_declaration
IDE0044	Add <code>readonly</code> modifier	dotnet_style_READONLY_field
IDE0045	Use conditional expression for assignment	dotnet_style_prefer_conditional_expression_over_assignment
IDE0046	Use conditional expression for return	dotnet_style_prefer_conditional_expression_over_return

RULE ID	TITLE	OPTION
IDE0047	Remove unnecessary parentheses	dotnet_style_parentheses_in_arithmetic_binary_operators dotnet_style_parentheses_in_relational_binary_operators dotnet_style_parentheses_in_other_binary_operators dotnet_style_parentheses_in_other_operators
IDE0048	Add parentheses for clarity	dotnet_style_parentheses_in_arithmetic_binary_operators dotnet_style_parentheses_in_relational_binary_operators dotnet_style_parentheses_in_other_binary_operators dotnet_style_parentheses_in_other_operators
IDE0049	Use language keywords instead of framework type names for type references	dotnet_style_predefined_type_for_locals_parameters_members dotnet_style_predefined_type_for_member_access
IDE0050	Convert anonymous type to tuple	
IDE0051	Remove unused private member	
IDE0052	Remove unread private member	
IDE0053	Use expression body for lambdas	csharp_style_expression_bodied_lambdas
IDE0054	Use compound assignment	dotnet_style_prefer_compound_assignment
IDE0055	Fix formatting	(Too many to list here. See .NET formatting options and C# formatting options .)
IDE0056	Use index operator	csharp_style_prefer_index_operator
IDE0057	Use range operator	csharp_style_prefer_range_operator
IDE0058	Remove unused expression value	csharp_style_unused_value_expression_statement_preference visual_basic_style_unused_value_expression_statement_preference
IDE0059	Remove unnecessary value assignment	csharp_style_unused_value_assignment_preference visual_basic_style_unused_value_assignment_preference
IDE0060	Remove unused parameter	dotnet_code_quality_unused_parameters

RULE ID	TITLE	OPTION
IDE0061	Use expression body for local functions	csharp_style_expression_bodied_local_functions
IDE0062	Make local function static	csharp_prefer_static_local_function
IDE0063	Use simple <code>using</code> statement	csharp_prefer_simple_using_statement
IDE0064	Make struct fields writable	
IDE0065	<code>using</code> directive placement	csharp_using_directive_placement
IDE0066	Use switch expression	csharp_style_prefer_switch_expression
IDE0070	Use <code>System.HashCode.Combine</code>	
IDE0071	Simplify interpolation	dotnet_style_prefer_simplified_interpolation
IDE0072	Add missing cases to switch expression	
IDE0073	Use file header	file_header_template
IDE0074	Use coalesce compound assignment	dotnet_style_prefer_compound_assignment
IDE0075	Simplify conditional expression	dotnet_style_prefer_simplified_boolean_expressions
IDE0076	Remove invalid global <code>SuppressMessageAttribute</code>	
IDE0077	Avoid legacy format target in global <code>SuppressMessageAttribute</code>	
IDE0078	Use pattern matching	csharp_style_prefer_pattern_matching
IDE0079	Remove unnecessary suppression	dotnet_remove_unnecessary_suppression_exclusions
IDE0080	Remove unnecessary suppression operator	
IDE0081	Remove <code>ByVal</code>	
IDE0082	Convert <code>typeof</code> to <code>nameof</code>	
IDE0083	Use pattern matching (<code>not</code> operator)	csharp_style_prefer_not_pattern
IDE0084	Use pattern matching (<code> IsNot</code> operator)	visual_basic_style_prefer_isnot_expression

Rule ID	Title	Option
IDE0090	Simplify <code>new</code> expression	<code>csharp_style_implicit_object_creation_when_type_is_apparent</code>
IDE0100	Remove unnecessary equality operator	
IDE0110	Remove unnecessary discard	
IDE0140	Simplify object creation	<code>visual_basic_style_prefer_simplified_object_creation</code>
IDE0150	Prefer <code>null</code> check over type check	<code>csharp_style_prefer_null_check_over_type_check</code>
IDE0160	Use block-scoped namespace	<code>csharp_style_namespace_declarations</code>
IDE0161	Use file-scoped namespace	<code>csharp_style_namespace_declarations</code>
IDE0170	Simplify property pattern	<code>csharp_style_prefer_extended_property_pattern</code>
IDE0180	Use tuple to swap values	<code>csharp_style_prefer_tuple_swap</code>
IDE1005	Use conditional delegate call	<code>csharp_style_conditional_delegate_call</code>
IDE1006	Naming styles	

Legend

The following table shows the type of information that's provided for each rule in the reference documentation.

Item	Description
Rule ID	The unique identifier for the rule. Used for configuring rule severity and suppressing warnings in the code file.
Title	The title for the rule.
Category	The category for the rule.
Subcategory	The subcategory for the rule, such as Language rules, Formatting rules, or Naming rules.
Applicable languages	Applicable .NET languages (C# or Visual Basic), along with the minimum language version, if applicable.
Introduced version	Version of the .NET SDK or Visual Studio when the rule was first introduced.
Options	Any available options for the rule.

See also

- Enforce code style on build
- Quick Actions in Visual Studio
- Create portable custom editor options in Visual Studio

Language rules

9/20/2022 • 3 minutes to read • [Edit Online](#)

Code-style language rules affect how various constructs of .NET programming languages, for example, modifiers, and parentheses, are used. The rules fall into the following categories:

- **.NET style rules:** Rules that apply to both C# and Visual Basic. The option names for these rules start with the prefix `dotnet_style_`.
- **C# style rules:** Rules that are specific to C# code. The option names for these rules start with the prefix `csharp_style_`.
- **Visual Basic style rules:** Rules that are specific to Visual Basic code. The option names for these rules start with the prefix `visual_basic_style_`.

Option format

Options for language rules can be specified in a [configuration file](#) with the following format:

`option_name = value` (Visual Studio 2019 version 16.9 and later)

or

`option_name = value:severity`

- **Value**

For each language rule, you specify a value that defines if or when to prefer the style. Many rules accept a value of `true` (prefer this style) or `false` (do not prefer this style). Other rules accept values such as `when_on_single_line` or `never`.

- **Severity** (optional in Visual Studio 2019 version 16.9 and later versions)

The second part of the rule specifies the [severity level](#) for the rule. When specified in this way, *the severity setting is only respected inside development IDEs, such as Visual Studio*. It is *not* respected during build.

To enforce code style rules at build time, set the severity by using the rule ID-based severity configuration syntax for analyzers instead. The syntax takes the form

`dotnet_diagnostic.<rule ID>.severity = <severity>`, for example,
`dotnet_diagnostic.IDE0040.severity = none`. For more information, see [severity level](#).

TIP

Starting in Visual Studio 2019 version 16.3, you can configure code style rules from the [Quick Actions](#) light bulb menu after a style violation occurs. For more information, see [Automatically configure code styles in Visual Studio](#).

.NET style rules

The style rules in this section are applicable to both C# and Visual Basic.

- ['this.' and 'Me.' qualifiers](#)
- [Language keywords instead of framework type names for type references](#)
- [Modifier preferences](#)
 - [Order modifiers \(IDE0036\)](#)

- Add accessibility modifiers (IDE0040)
- Add readonly modifier (IDE0044)
- Parentheses preferences
- Expression-level preferences
 - Use object initializers (IDE0017)
 - Use collection initializers (IDE0028)
 - Use auto-implemented property (IDE0032)
 - Use explicitly provided tuple name (IDE0033)
 - Use inferred member names (IDE0037)
 - Use conditional expression for assignment (IDE0045)
 - Use conditional expression for return (IDE0046)
 - Use compound assignment (IDE0054 and IDE0074)
 - Simplify interpolation (IDE0071)
 - Simplify conditional expression (IDE0075)
 - Add missing cases to switch statement (IDE0010)
 - Convert anonymous type to tuple (IDE0050)
 - Use 'System.HashCode.Combine' (IDE0070)
 - Convert `typeof` to `nameof` (IDE0082)
- Null-checking preferences
 - Use coalesce expression (IDE0029 and IDE0030)
 - Use null propagation (IDE0031)
 - Use 'is null' check (IDE0041)
- File header preferences

C# style rules

The style rules in this section are applicable to C# language only.

- 'var' preferences (IDE0007 and IDE0008)
- Expression-bodied members
 - Use expression body for constructors (IDE0021)
 - Use expression body for methods (IDE0022)
 - Use expression body for operators (IDE0023 and IDE0024)
 - Use expression body for properties (IDE0025)
 - Use expression body for indexers (IDE0026)
 - Use expression body for accessors (IDE0027)
 - Use expression body for lambdas (IDE0053)
 - Use expression body for local functions (IDE0061)
- Pattern matching preferences
 - Use pattern matching to avoid 'as' followed by a 'null' check (IDE0019)
 - Use pattern matching to avoid 'is' check followed by a cast (IDE0020 and IDE0038)
 - Use switch expression (IDE0066)
 - Use pattern matching (IDE0078)
 - Use pattern matching (`not` operator) (IDE0083)
 - Simplify property pattern (IDE0170)
- Expression-level preferences
 - Inline variable declaration (IDE0018)
 - Simplify 'default' expression (IDE0034)

- Use local function instead of lambda (IDE0039)
- Deconstruct variable declaration (IDE0042)
- Use index operator (IDE0056)
- Use range operator (IDE0057)
- Simplify `new` expression (IDE0090)
- Add missing cases to switch expression (IDE0072)
- Use tuple to swap values (IDE0180)
- "Null" checking preferences
 - Use throw expression (IDE0016)
 - Use conditional delegate call (IDE1005)
 - Prefer 'null' check over type check (IDE0150)
- Code block preferences
 - Add braces (IDE0011)
 - Use simple 'using' statement (IDE0063)
- 'using' directive preferences (IDE0065)
- Modifier preferences
 - Make local function static (IDE0062)
 - Make struct fields writable (IDE0064)
- Namespace declaration preferences (IDE0160 and IDE0161)

Visual Basic style rules

The style rules in this section are applicable to Visual Basic language only.

- Pattern matching preferences
 - Use pattern matching (`IsNot` operator) (IDE0084)

See also

- Unnecessary code rules
- Formatting rules
- Naming rules
- .NET code style rules reference

this and Me preferences (IDE0003 and IDE0009)

9/20/2022 • 3 minutes to read • [Edit Online](#)

This article describes two related rules, [IDE0003](#) and [IDE0009](#).

PROPERTY	VALUE
Rule ID	IDE0003
Title	Remove <code>this</code> or <code>Me</code> qualification
Category	Style
Subcategory	Language rules
Applicable languages	C# and Visual Basic
Options	<code>dotnet_style_qualification_for_field</code> <code>dotnet_style_qualification_for_property</code> <code>dotnet_style_qualification_for_method</code> <code>dotnet_style_qualification_for_event</code>

PROPERTY	VALUE
Rule ID	IDE0009
Title	Add <code>this</code> or <code>Me</code> qualification
Category	Style
Subcategory	Language rules
Applicable languages	C# and Visual Basic
Options	<code>dotnet_style_qualification_for_field</code> <code>dotnet_style_qualification_for_property</code> <code>dotnet_style_qualification_for_method</code> <code>dotnet_style_qualification_for_event</code>

Overview

These two rules define whether or not you prefer the use of `this` (C#) and `Me.` (Visual Basic) qualifiers. To

enforce that the qualifiers *aren't* present, set the severity of `IDE0003` to warning or error. To enforce that the qualifiers *are* present, set the severity of `IDE0009` to warning or error.

For example, if you prefer qualifiers for fields and properties but not for methods or events, then you can enable `IDE0009` and set the options `dotnet_style_qualification_for_field` and `dotnet_style_qualification_for_property` to `true`. However, this configuration would not flag methods and events that *do* have `this` and `Me` qualifiers. To also enforce that methods and events *don't* have qualifiers, enable `IDE0003`.

Options

This rule's associated options define which of the following symbols this style preference should be applied to:

- Fields (`dotnet_style_qualification_for_field`)
- Properties (`dotnet_style_qualification_for_property`)
- Methods (`dotnet_style_qualification_for_method`)
- Events (`dotnet_style_qualification_for_event`)

An option value of `true` means prefer the code symbol to be prefaced with `this.` in C# and `Me.` in Visual Basic. An option value of `false` means prefer the code element *not* to be prefaced with `this.` or `Me.`.

For more information about configuring options, see [Option format](#).

`dotnet_style_qualification_for_field`

PROPERTY	VALUE	DESCRIPTION
Option name	<code>dotnet_style_qualification_for_field</code>	
Option values	<code>true</code>	Prefer fields to be prefaced with <code>this.</code> in C# or <code>Me.</code> in Visual Basic
	<code>false</code>	Prefer fields <i>not</i> to be prefaced with <code>this.</code> or <code>Me.</code>
Default option value	<code>false</code>	

```
// dotnet_style_qualification_for_field = true
this.capacity = 0;

// dotnet_style_qualification_for_field = false
capacity = 0;
```

```
' dotnet_style_qualification_for_field = true
Me.capacity = 0

' dotnet_style_qualification_for_field = false
capacity = 0
```

`dotnet_style_qualification_for_property`

PROPERTY	VALUE	DESCRIPTION
Option name	<code>dotnet_style_qualification_for_property</code>	

PROPERTY	VALUE	DESCRIPTION
Option values	true	Prefer properties to be prefaced with <code>this.</code> in C# or <code>Me.</code> in Visual Basic.
	false	Prefer properties <i>not</i> to be prefaced with <code>this.</code> or <code>Me.</code> .
Default option value	false	

```
// dotnet_style_qualification_for_property = true
this.ID = 0;

// dotnet_style_qualification_for_property = false
ID = 0;
```

```
' dotnet_style_qualification_for_property = true
Me.ID = 0

' dotnet_style_qualification_for_property = false
ID = 0
```

dotnet_style_qualification_for_method

PROPERTY	VALUE	DESCRIPTION
Option name	dotnet_style_qualification_for_method	
Option values	true	Prefer methods to be prefaced with <code>this.</code> in C# or <code>Me.</code> in Visual Basic.
	false	Prefer methods <i>not</i> to be prefaced with <code>this.</code> or <code>Me.</code> .
Default option value	false	

```
// dotnet_style_qualification_for_method = true
this.Display();

// dotnet_style_qualification_for_method = false
Display();
```

```
' dotnet_style_qualification_for_method = true
Me.Display()

' dotnet_style_qualification_for_method = false
Display()
```

dotnet_style_qualification_for_event

PROPERTY	VALUE	DESCRIPTION
Option name	dotnet_style_qualification_for_event	

PROPERTY	VALUE	DESCRIPTION
Option values	<code>true</code>	Prefer events to be prefaced with <code>this.</code> in C# or <code>Me.</code> in Visual Basic.
	<code>false</code>	Prefer events <i>not</i> to be prefaced with <code>this.</code> or <code>Me.</code> .
Default option value	<code>false</code>	

```
// dotnet_style_qualification_for_event = true
this.Elapsed += Handler;

// dotnet_style_qualification_for_event = false
Elapsed += Handler;
```

```
' dotnet_style_qualification_for_event = true
AddHandler Me.Elapsed, AddressOf Handler

' dotnet_style_qualification_for_event = false
AddHandler Elapsed, AddressOf Handler
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0003 // Or IDE0009
// The code that's violating the rule is on this line.
#pragma warning restore IDE0003 // Or IDE0009
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.IDE0003.severity = none
dotnet_diagnostic.IDE0009.severity = none
```

To disable all of the code-style rules, set the severity for the category `Style` to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [this \(C# Reference\)](#)
- [Code style language rules](#)
- [Code style rules reference](#)

Use language keywords instead of framework type names for type references (IDE0049)

9/20/2022 • 2 minutes to read • [Edit Online](#)

PROPERTY	VALUE
Rule ID	IDE0049
Title	Use language keywords instead of framework type names for type references
Category	Style
Subcategory	Language rules
Applicable languages	C# and Visual Basic
Options	<code>dotnet_style_predefined_type_for_locals_parameters_members</code> <code>dotnet_style_predefined_type_for_member_access</code>

Overview

This rule concerns the use of language keywords, [where they exist](#), instead of framework type names.

Options

Use the associated options for this rule to apply it to:

- Local variables, method parameters, and class members - [`dotnet_style_predefined_type_for_locals_parameters_members`](#)
- Type-member access expressions - [`dotnet_style_predefined_type_for_member_access`](#)

An option value of `true` means prefer the language keyword (for example, `int` or `Integer`) instead of the type name (for example, `Int32`) for types that have a keyword to represent them. A value of `false` means prefer the type name instead of the language keyword.

For information about configuring options, see [Option format](#).

`dotnet_style_predefined_type_for_locals_parameters_members`

PROPERTY	VALUE	DESCRIPTION
Option name	<code>dotnet_style_predefined_type_for_locals_parameters_members</code>	
Option values	<code>true</code>	Prefer the language keyword for local variables, method parameters, and class members

PROPERTY	VALUE	DESCRIPTION
	false	Prefer the type name for local variables, method parameters, and class members
Default option value	true	

```
// dotnet_style_predefined_type_for_locals_parameters_members = true
private int _member;

// dotnet_style_predefined_type_for_locals_parameters_members = false
private Int32 _member;
```

```
' dotnet_style_predefined_type_for_locals_parameters_members = true
Private _member As Integer

' dotnet_style_predefined_type_for_locals_parameters_members = false
Private _member As Int32
```

dotnet_style_predefined_type_for_member_access

PROPERTY	VALUE	DESCRIPTION
Option name	dotnet_style_predefined_type_for_member_access	
Option values	true	Prefer the language keyword for member access expressions
	false	Prefer the type name for member access expressions
Default option value	true	

```
// dotnet_style_predefined_type_for_member_access = true
var local = int.MaxValue;

// dotnet_style_predefined_type_for_member_access = false
var local = Int32.MaxValue;
```

```
' dotnet_style_predefined_type_for_member_access = true
Dim local = Integer.MaxValue

' dotnet_style_predefined_type_for_member_access = false
Dim local = Int32.MaxValue
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0049
// The code that's violating the rule is on this line.
#pragma warning restore IDE0049
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.IDE0049.severity = none
```

To disable all of the code-style rules, set the severity for the category `Style` to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Code style language rules](#)
- [Code style rules reference](#)

Modifier preferences

9/20/2022 • 2 minutes to read • [Edit Online](#)

.NET modifier preferences

The style rules in this section concern the following modifier preferences that are common to C# and Visual Basic:

- [Order modifiers \(IDE0036\)](#)
- [Add accessibility modifiers \(IDE0040\)](#)
- [Add readonly modifier \(IDE0044\)](#)

C# modifier preferences

The style rules in this section concern the following modifier preferences that are specific to C#:

- [Make local function static \(IDE0062\)](#)
- [Make struct fields writable \(IDE0064\)](#)

See also

- [Code style rules reference](#)
- [Code style language rules](#)

Order modifiers (IDE0036)

9/20/2022 • 2 minutes to read • [Edit Online](#)

PROPERTY	VALUE
Rule ID	IDE0036
Title	Order modifiers
Category	Style
Subcategory	Language rules (modifier preferences)
Applicable languages	C# and Visual Basic
Introduced version	Visual Studio 2017 version 15.5
Options	<code>csharp_preferred_modifier_order</code>
	<code>visual_basic_preferred_modifier_order</code>

Overview

This rule lets you enforce a desired [modifier](#) sort order.

- When this rule is enabled and the associated options are set to a list of modifiers, prefer the specified ordering.
- When this rule is not enabled, no specific modifier order is preferred.

Options

The associated options for this rule let you specify the desired modifier order for C# and Visual Basic, respectively.

- [`csharp_preferred_modifier_order`](#)
- [`visual_basic_preferred_modifier_order`](#)

For information about configuring options, see [Option format](#).

`csharp_preferred_modifier_order`

PROPERTY	VALUE	DESCRIPTION
Option name	<code>csharp_preferred_modifier_order</code>	
Applicable languages	C#	
Option values	One or more C# modifiers, such as <code>public</code> , <code>private</code> , and <code>protected</code>	

PROPERTY	VALUE	DESCRIPTION
Default option value	public, private, protected, internal, file, static, extern, new, virtual, abstract, sealed, override, readonly, unsafe, required, volatile, async	

```
// csharp_preferred_modifier_order =
public,private,protected,internal,static,extern,new,virtual,abstract,sealed,override,readonly,unsafe,volatile,async
class MyClass
{
    private static readonly int _daysInYear = 365;
}
```

visual_basic_preferred_modifier_order

PROPERTY	VALUE	DESCRIPTION
Option name	visual_basic_preferred_modifier_order	
Applicable languages	Visual Basic	
Option values	One or more Visual Basic modifiers, such as <code>Partial</code> , <code>Private</code> , and <code>Public</code>	
Default option value	<code>Partial, Default, Private, Protected, Public, Friend, NotOverridable, Overridable, MustOverride, Overloads, Overrides, MustInherit, NotInheritable, Static, Shared, Shadows, ReadOnly, WriteOnly, Dim, Const, WithEvents, Widening, Narrowing, Custom, Async</code>	

```
' visual_basic_preferred_modifier_order =
Partial,Default,Private,Protected,Public,Friend,NotOverridable,Overridable,MustOverride,Overloads,Overrides,
MustInherit,NotInheritable,Static,Shared,Shadows,ReadOnly,WriteOnly,Dim,Const,WithEvents,Widening,Narrowing,
Custom,Async
Public Class MyClass
    Private Shared ReadOnly daysInYear As Int = 365
End Class
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0036
// The code that's violating the rule is on this line.
#pragma warning restore IDE0036
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.IDE0036.severity = none
```

To disable all of the code-style rules, set the severity for the category `Style` to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Modifier preferences](#)
- [Code style language rules](#)
- [Code style rules reference](#)

Add accessibility modifiers (IDE0040)

9/20/2022 • 2 minutes to read • [Edit Online](#)

PROPERTY	VALUE
Rule ID	IDE0040
Title	Add accessibility modifiers
Category	Style
Subcategory	Language rules (modifier preferences)
Applicable languages	C# and Visual Basic
Introduced version	Visual Studio 2017 version 15.5
Options	<code>dotnet_style_require_accessibility_modifiers</code>

Overview

This style rule concerns requiring [accessibility modifiers](#) in declarations.

Options

The option value specifies the preferences for required accessibility modifiers.

For information about configuring options, see [Option format](#).

`dotnet_style_require_accessibility_modifiers`

PROPERTY	VALUE	DESCRIPTION
Option name	<code>dotnet_style_require_accessibility_modifiers</code>	
Option values	<code>always</code>	Prefer accessibility modifiers to be specified.
	<code>for_non_interface_members</code>	Prefer accessibility modifiers except for public interface members.
	<code>never</code>	Do not prefer accessibility modifiers to be specified.
	<code>omit_if_default</code>	Prefer accessibility modifiers except if they are the default modifier.
Default option value	<code>for_non_interface_members</code>	

```
// dotnet_style_require_accessibility_modifiers = always
// dotnet_style_require_accessibility_modifiers = for_non_interface_members
class MyClass
{
    private const string thisFieldIsConst = "constant";
}

// dotnet_style_require_accessibility_modifiers = never
class MyClass
{
    const string thisFieldIsConst = "constant";
}
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0040
// The code that's violating the rule is on this line.
#pragma warning restore IDE0040
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.IDE0040.severity = none
```

To disable all of the code-style rules, set the severity for the category `Style` to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Modifier preferences](#)
- [Code style language rules](#)
- [Code style rules reference](#)

Add readonly modifier (IDE0044)

9/20/2022 • 2 minutes to read • [Edit Online](#)

PROPERTY	VALUE
Rule ID	IDE0044
Title	Add readonly modifier
Category	Style
Subcategory	Language rules (modifier preferences)
Applicable languages	C# and Visual Basic
Introduced version	Visual Studio 2017 version 15.7
Options	<code>dotnet_style_READONLY_FIELD</code>

Overview

This style rule concerns specifying the `readonly` (C#) or `ReadOnly` (Visual Basic) modifier for private fields that are initialized (either inline or inside of a constructor) but never reassigned.

Options

Options specify the behavior that you want the rule to enforce. For information about configuring options, see [Option format](#).

`dotnet_style_READONLY_FIELD`

PROPERTY	VALUE	DESCRIPTION
Option name	<code>dotnet_style_READONLY_FIELD</code>	
Option values	<code>true</code>	Prefer that private fields be marked <code>readonly</code> if they're only ever assigned inline or in a constructor
	<code>false</code>	Specify no preference over whether private fields are marked <code>readonly</code>
Default option value	<code>true</code>	

```
// dotnet_style_READONLY_FIELD = true
class MyClass
{
    private readonly int _daysInYear = 365;
}
```

```
' dotnet_style_READONLY_field = true
Public Class MyClass
    Private ReadOnly daysInYear As Int = 365
End Class
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0044
// The code that's violating the rule is on this line.
#pragma warning restore IDE0044
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.IDE0044.severity = none
```

To disable all of the code-style rules, set the severity for the category `Style` to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Modifier preferences](#)
- [Code style language rules](#)
- [Code style rules reference](#)

Make local function static (IDE0062)

9/20/2022 • 2 minutes to read • [Edit Online](#)

PROPERTY	VALUE
Rule ID	IDE0062
Title	Make local function static
Category	Style
Subcategory	Language rules (modifier preferences)
Applicable languages	C# 8.0+
Options	<code>csharp_prefer_static_local_function</code>

Overview

This style rule concerns the preference of marking [local functions](#) as `static` or not.

Options

Options specify the behavior that you want the rule to enforce. For information about configuring options, see [Option format](#).

`csharp_prefer_static_local_function`

PROPERTY	VALUE	DESCRIPTION
Option name	<code>csharp_prefer_static_local_function</code>	
Option values	<code>true</code>	Prefer local functions to be marked <code>static</code>
	<code>false</code>	Prefer local functions <i>not</i> to be marked <code>static</code>
Default option value	<code>true:suggestion</code>	

```
// csharp_prefer_static_local_function = true
void M()
{
    Hello();
    static void Hello()
    {
        Console.WriteLine("Hello");
    }
}

// csharp_prefer_static_local_function = false
void M()
{
    Hello();
    void Hello()
    {
        Console.WriteLine("Hello");
    }
}
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0062
// The code that's violating the rule is on this line.
#pragma warning restore IDE0062
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.IDE0062.severity = none
```

To disable all of the code-style rules, set the severity for the category `Style` to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Modifier preferences](#)
- [Code style language rules](#)
- [Code style rules reference](#)

Make struct fields writable (IDE0064)

9/20/2022 • 2 minutes to read • [Edit Online](#)

PROPERTY	VALUE
Rule ID	IDE0064
Title	Make struct fields writable
Category	CodeQuality
Subcategory	Language rules (modifier preferences)
Applicable languages	C#

Overview

This rule detects structs that contain one or more `readonly` fields and also contain an assignment to `this` outside of the constructor. The rule recommends converting `readonly` fields to non-read only, that is, writable. Marking such struct fields as `readonly` can lead to unexpected behavior, because the value assigned to the field can change when `this` is assigned outside the constructor.

Options

This rule has no associated code-style options.

Example

```
// Code with violations
struct MyStruct
{
    public readonly int Value;

    public MyStruct(int value)
    {
        Value = value;
    }

    public void Test()
    {
        this = new MyStruct(5);
    }
}

// Fixed code
struct MyStruct
{
    public int Value;

    public MyStruct(int value)
    {
        Value = value;
    }

    public void Test()
    {
        this = new MyStruct(5);
    }
}
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0064
// The code that's violating the rule is on this line.
#pragma warning restore IDE0064
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.IDE0064.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-CodeQuality.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Modifier preferences](#)
- [Code style language rules](#)
- [Code style rules reference](#)

Parentheses preferences (IDE0047 and IDE0048)

9/20/2022 • 3 minutes to read • [Edit Online](#)

This article describes two related rules, [IDE0047](#) and [IDE0048](#).

PROPERTY	VALUE
Rule ID	IDE0047
Title	Remove unnecessary parentheses
Category	Style
Subcategory	Language rules
Applicable languages	C# and Visual Basic
Introduced version	Visual Studio 2017 version 15.8
Options	<code>dotnet_style_parentheses_in_arithmetic_binary_operators</code> <code>dotnet_style_parentheses_in_relational_binary_operators</code> <code>dotnet_style_parentheses_in_other_binary_operators</code> <code>dotnet_style_parentheses_in_other_operators</code>

PROPERTY	VALUE
Rule ID	IDE0048
Title	Add parentheses for clarity
Category	Style
Subcategory	Language rules
Applicable languages	C# and Visual Basic
Introduced version	Visual Studio 2017 version 15.8
Options	<code>dotnet_style_parentheses_in_arithmetic_binary_operators</code> <code>dotnet_style_parentheses_in_relational_binary_operators</code> <code>dotnet_style_parentheses_in_other_binary_operators</code> <code>dotnet_style_parentheses_in_other_operators</code>

Overview

The style rules in this section concern parentheses preferences, including the use of parentheses to clarify precedence for arithmetic, relational, and other binary operators.

Options

This rule has associated options to specify preferences based on the type of operator:

- Arithmetic binary operators - [dotnet_style_parentheses_in_arithmetic_binary_operators](#)
- Relational binary operators - [dotnet_style_parentheses_in_relational_binary_operators](#)
- Other binary operators - [dotnet_style_parentheses_in_other_binary_operators](#)
- Other operators - [dotnet_style_parentheses_in_other_operators](#)

For information about configuring options, see [Option format](#).

dotnet_style_parentheses_in_arithmetic_binary_operators

PROPERTY	VALUE	DESCRIPTION
Option name	dotnet_style_parentheses_in_arithmetic_binary_operators	
Option values	<code>always_for_clarity</code>	Prefer parentheses to clarify arithmetic operator precedence
	<code>never_if_unnecessary</code>	Prefer no parentheses when arithmetic operator precedence is obvious
Default option value	<code>always_for_clarity</code>	

The arithmetic binary operators are: `*`, `/`, `%`, `+`, `-`, `<<`, `>>`, `&`, `^`, and `|`.

```
// dotnet_style_parentheses_in_arithmetic_binary_operators = always_for_clarity
var v = a + (b * c);

// dotnet_style_parentheses_in_arithmetic_binary_operators = never_if_unnecessary
var v = a + b * c;
```

```
' dotnet_style_parentheses_in_arithmetic_binary_operators = always_for_clarity
Dim v = a + (b * c)

' dotnet_style_parentheses_in_arithmetic_binary_operators = never_if_unnecessary
Dim v = a + b * c
```

dotnet_style_parentheses_in_relational_binary_operators

PROPERTY	VALUE	DESCRIPTION
Option name	dotnet_style_parentheses_in_relational_binary_operators	
Option values	<code>always_for_clarity</code>	Prefer parentheses to clarify relational operator precedence

PROPERTY	VALUE	DESCRIPTION
	never_if_unnecessary	Prefer to not have parentheses when relational operator precedence is obvious
Default option value	always_for_clarity	

The relational binary operators are: >, <, <=, >=, is, as, ==, and !=.

```
// dotnet_style_parentheses_in_relational_binary_operators = always_for_clarity
var v = (a < b) == (c > d);

// dotnet_style_parentheses_in_relational_binary_operators = never_if_unnecessary
var v = a < b == c > d;
```

```
' dotnet_style_parentheses_in_relational_binary_operators = always_for_clarity
Dim v = (a < b) = (c > d)

' dotnet_style_parentheses_in_relational_binary_operators = never_if_unnecessary
Dim v = a < b = c > d
```

dotnet_style_parentheses_in_other_binary_operators

PROPERTY	VALUE	DESCRIPTION
Option name	dotnet_style_parentheses_in_other_binary_operators	
Option values	always_for_clarity	Prefer parentheses to clarify other binary operator precedence
	never_if_unnecessary	Prefer to not have parentheses when other binary operator precedence is obvious
Default option value	always_for_clarity	

The other binary operators are: &&, ||, and ??.

```
// dotnet_style_parentheses_in_other_binary_operators = always_for_clarity
var v = a || (b && c);

// dotnet_style_parentheses_in_other_binary_operators = never_if_unnecessary
var v = a || b && c;
```

```
' dotnet_style_parentheses_in_other_binary_operators = always_for_clarity
Dim v = a OrElse (b AndAlso c)

' dotnet_style_parentheses_in_other_binary_operators = never_if_unnecessary
Dim v = a OrElse b AndAlso c
```

dotnet_style_parentheses_in_other_operators

PROPERTY	VALUE	DESCRIPTION
Option name	dotnet_style_parentheses_in_other_operators	
Option values	always_for_clarity never_if_unnecessary	Prefer parentheses to clarify other operator precedence
	never_if_unnecessary	Prefer to not have parentheses when other operator precedence is obvious
Default option value	never_if_unnecessary	

This option applies to operators *other than* the following:

```
* , / , % , + , - , << , >> , & , ^ , | , > , < , <= , >= , is , as , == , != , && , || , ??
```

```
// dotnet_style_parentheses_in_other_operators = always_for_clarity
var v = (a.b).Length;

// dotnet_style_parentheses_in_other_operators = never_if_unnecessary
var v = a.b.Length;
```

```
' dotnet_style_parentheses_in_other_operators = always_for_clarity
Dim v = (a.b).Length

' dotnet_style_parentheses_in_other_operators = never_if_unnecessary
Dim v = a.b.Length
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0047 // Or IDE0048
// The code that's violating the rule is on this line.
#pragma warning restore IDE0047 // Or IDE0048
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.IDE0047.severity = none
dotnet_diagnostic.IDE0048.severity = none
```

To disable all of the code-style rules, set the severity for the category `Style` to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Code style language rules](#)
- [Code style rules reference](#)

Expression-level preferences

9/20/2022 • 2 minutes to read • [Edit Online](#)

.NET expression-level preferences

The style rules in this section concern the following expression-level preferences that are common to C# and Visual Basic:

- [Add missing cases to switch statement \(IDE0010\)](#)
- [Use object initializers \(IDE0017\)](#)
- [Use collection initializers \(IDE0028\)](#)
- [Use auto property \(IDE0032\)](#)
- [Use explicitly provided tuple name \(IDE0033\)](#)
- [Use inferred member name \(IDE0037\)](#)
- [Use conditional expression for assignment \(IDE0045\)](#)
- [Use conditional expression for return \(IDE0046\)](#)
- [Convert anonymous type to tuple \(IDE0050\)](#)
- [Use compound assignment \(IDE0054 and IDE0074\)](#)
- [Use 'System.HashCode.Combine' \(IDE0070\)](#)
- [Simplify interpolation \(IDE0071\)](#)
- [Simplify conditional expression \(IDE0075\)](#)
- [Convert 'typeof' to 'nameof' \(IDE0082\)](#)

C# expression-level preferences

The style rules in this section concern the following expression-level preferences that are specific to C#:

- [Inline variable declaration \(IDE0018\)](#)
- [Simplify 'default' expression \(IDE0034\)](#)
- [Use local function instead of lambda \(IDE0039\)](#)
- [Deconstruct variable declaration \(IDE0042\)](#)
- [Use index operator \(IDE0056\)](#)
- [Use range operator \(IDE0057\)](#)
- [Add missing cases to switch expression \(IDE0072\)](#)
- [Simplify 'new' expression \(IDE0090\)](#)
- [Use tuple to swap values \(IDE0180\)](#)

See also

- [Code style rules reference](#)
- [Code style language rules](#)

Add missing cases to switch statement (IDE0010)

9/20/2022 • 2 minutes to read • [Edit Online](#)

PROPERTY	VALUE
Rule ID	IDE0010
Title	Add missing cases to switch statement
Category	Style
Subcategory	Language rules (expression-level preferences)
Applicable languages	C# and Visual Basic

Overview

This rule concerns specifying all the missing switch cases for a `switch` statement. A `switch` statement is considered incomplete in the following scenarios:

- An `enum` `switch` statement that's missing cases for one or more enum members.
- A `switch` statement with a missing `default` case.

Options

This rule has no associated code-style options.

Example

```

enum E
{
    A,
    B
}

class C
{
    // Code with violations
    int M(E e)
    {
        // IDE0010: Add missing cases
        switch (e)
        {
            case E.A:
                return 0;
        }

        return -1;
    }

    // Fixed code
    int M(E e)
    {
        switch (e)
        {
            case E.A:
                return 0;
            case E.B:
                return 1;
            default:
                return -1;
        }
    }
}

```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```

#pragma warning disable IDE0010
// The code that's violating the rule is on this line.
#pragma warning restore IDE0010

```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```

[*.{cs,vb}]
dotnet_diagnostic.IDE0010.severity = none

```

To disable all of the code-style rules, set the severity for the category `Style` to `none` in the [configuration file](#).

```

[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Style.severity = none

```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Switch statement](#)
- [Add missing cases to switch expression \(IDE0072\)](#)
- [Expression-level preferences](#)
- [Code style language rules](#)
- [Code style rules reference](#)

Use object initializers (IDE0017)

9/20/2022 • 2 minutes to read • [Edit Online](#)

PROPERTY	VALUE
Rule ID	IDE0017
Title	Use object initializers
Category	Style
Subcategory	Language rules (expression-level preferences)
Applicable languages	C# and Visual Basic
Options	<code>dotnet_style_object_initializer</code>

Overview

This style rule concerns the use of [object initializers](#) for object initialization.

Options

The option value for this rule specifies whether or not initializers are desired.

For more information about configuring options, see [Option format](#).

`dotnet_style_object_initializer`

PROPERTY	VALUE	DESCRIPTION
Option name	<code>dotnet_style_object_initializer</code>	
Option values	<code>true</code>	Prefer objects to be initialized using object initializers when possible
	<code>false</code>	Prefer objects to <i>not</i> be initialized using object initializers
Default option value	<code>true</code>	

```
// dotnet_style_object_initializer = true
var c = new Customer() { Age = 21 };

// dotnet_style_object_initializer = false
var c = new Customer();
c.Age = 21;
```

```
' dotnet_style_object_initializer = true
Dim c = New Customer() With {.Age = 21}

' dotnet_style_object_initializer = false
Dim c = New Customer()
c.Age = 21
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0017
// The code that's violating the rule is on this line.
#pragma warning restore IDE0017
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.IDE0017.severity = none
```

To disable all of the code-style rules, set the severity for the category `Style` to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Object and Collection Initializers \(C# Programming Guide\)](#)
- [IDE0028 - Use collection initializers](#)
- [Expression-level preferences](#)
- [Code style language rules](#)
- [Code style rules reference](#)

Inline variable declaration (IDE0018)

9/20/2022 • 2 minutes to read • [Edit Online](#)

PROPERTY	VALUE
Rule ID	IDE0018
Title	Inline variable declaration
Category	Style
Subcategory	Language rules (expression-level preferences)
Applicable languages	C# 7.0+
Options	<code>csharp_style_inlined_variable_declaration</code>

Overview

This style rule concerns whether `out` variables are declared inline or not. Starting in C# 7, you can [declare an out variable in the argument list of a method call](#), rather than in a separate variable declaration.

Options

The associated option for this rule specifies whether you prefer `out` variables to be declared inline or separately.

For more information about configuring options, see [Option format](#).

`csharp_style_inlined_variable_declaration`

PROPERTY	VALUE	DESCRIPTION
Option name	<code>csharp_style_inlined_variable_declaratio</code> n	
Option values	<code>true</code>	Prefer <code>out</code> variables to be declared inline in the argument list of a method call when possible
	<code>false</code>	Prefer <code>out</code> variables to be declared before the method call
Default option value	<code>true</code>	

```
// csharp_style_inlined_variable_declaration = true
if (int.TryParse(value, out int i) {...}

// csharp_style_inlined_variable_declaration = false
int i;
if (int.TryParse(value, out i) {...}
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0018
// The code that's violating the rule is on this line.
#pragma warning restore IDE0018
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.IDE0018.severity = none
```

To disable all of the code-style rules, set the severity for the category `Style` to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Expression-level preferences](#)
- [Code style language rules](#)
- [Code style rules reference](#)

Use collection initializers (IDE0028)

9/20/2022 • 2 minutes to read • [Edit Online](#)

PROPERTY	VALUE
Rule ID	IDE0028
Title	Use collection initializers
Category	Style
Subcategory	Language rules (expression-level preferences)
Applicable languages	C# and Visual Basic
Options	<code>dotnet_style_collection_initializer</code>

Overview

This style rule concerns the use of [collection initializers](#) for collection initialization.

Options

Set the value of the associated option for this rule to specify whether or not collection initializers are preferred when initializing collections.

For more information about configuring options, see [Option format](#).

`dotnet_style_collection_initializer`

PROPERTY	VALUE	DESCRIPTION
Option name	<code>dotnet_style_collection_initializer</code>	
Option values	<code>true</code>	Prefer collections to be initialized using collection initializers when possible
	<code>false</code>	Prefer collections to <i>not</i> be initialized using collection initializers
Default option value	<code>true</code>	

```
// dotnet_style_collection_initializer = true
var list = new List<int> { 1, 2, 3 };

// dotnet_style_collection_initializer = false
var list = new List<int>();
list.Add(1);
list.Add(2);
list.Add(3);
```

```
' dotnet_style_collection_initializer = true
Dim list = New List(Of Integer) From {1, 2, 3}

' dotnet_style_collection_initializer = false
Dim list = New List(Of Integer)
list.Add(1)
list.Add(2)
list.Add(3)
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0028
// The code that's violating the rule is on this line.
#pragma warning restore IDE0028
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.IDE0028.severity = none
```

To disable all of the code-style rules, set the severity for the category `Style` to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Use object initializers](#)
- [Expression-level preferences](#)
- [Code style language rules](#)
- [Code style rules reference](#)

Use auto-implemented property (IDE0032)

9/20/2022 • 2 minutes to read • [Edit Online](#)

PROPERTY	VALUE
Rule ID	IDE0032
Title	Use auto-implemented property
Category	Style
Subcategory	Language rules (expression-level preferences)
Applicable languages	C# and Visual Basic
Introduced version	Visual Studio 2017 version 15.7
Options	<code>dotnet_style_prefer_auto_properties</code>

Overview

This style rule concerns the use of [auto-implemented properties](#) versus properties with private backing fields.

Options

Options specify the behavior that you want the rule to enforce. For information about configuring options, see [Option format](#).

`dotnet_style_prefer_auto_properties`

PROPERTY	VALUE	DESCRIPTION
Option name	<code>dotnet_style_prefer_auto_properties</code>	
Option values	<code>true</code>	Prefer auto-implemented properties
	<code>false</code>	Prefer properties with private backing fields
Default option value	<code>true</code>	

```
// dotnet_style_prefer_auto_properties = true
private int Age { get; }

// dotnet_style_prefer_auto_properties = false
private int age;

public int Age
{
    get
    {
        return age;
    }
}
```

```
' dotnet_style_prefer_auto_properties = true
Public ReadOnly Property Age As Integer

' dotnet_style_prefer_auto_properties = false
Private _age As Integer

Public ReadOnly Property Age As Integer
    Get
        Return _age
    End Get
End Property
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0032
// The code that's violating the rule is on this line.
#pragma warning restore IDE0032
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.IDE0032.severity = none
```

To disable all of the code-style rules, set the severity for the category `Style` to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Expression-level preferences](#)
- [Code style language rules](#)
- [Code style rules reference](#)

Use explicitly provided tuple name (IDE0033)

9/20/2022 • 2 minutes to read • [Edit Online](#)

PROPERTY	VALUE
Rule ID	IDE0033
Title	Use explicitly provided tuple name
Category	Style
Subcategory	Language rules (expression-level preferences)
Applicable languages	C# 7.0+ and Visual Basic 15+
Options	<code>dotnet_style_explicit_tuple_names</code>

Overview

This style rule concerns the use of explicit [tuple](#) names versus implicit 'ItemX' properties when accessing tuple fields.

Options

Options specify the behavior that you want the rule to enforce. For information about configuring options, see [Option format](#).

`dotnet_style_explicit_tuple_names`

PROPERTY	VALUE	DESCRIPTION
Option name	<code>dotnet_style_explicit_tuple_names</code>	
Option values	<code>true</code>	Prefer tuple names to ItemX properties
	<code>false</code>	Prefer ItemX properties to tuple names
Default option value	<code>true</code>	

```
// dotnet_style_explicit_tuple_names = true
(string name, int age) customer = GetCustomer();
var name = customer.name;

// dotnet_style_explicit_tuple_names = false
(string name, int age) customer = GetCustomer();
var name = customer.Item1;
```

```
' dotnet_style_explicit_tuple_names = true
Dim customer As (name As String, age As Integer) = GetCustomer()
Dim name = customer.name

' dotnet_style_explicit_tuple_names = false
Dim customer As (name As String, age As Integer) = GetCustomer()
Dim name = customer.Item1
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0033
// The code that's violating the rule is on this line.
#pragma warning restore IDE0033
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.IDE0033.severity = none
```

To disable all of the code-style rules, set the severity for the category `Style` to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Use inferred member names \(IDE0037\)](#)
- [Use object initializers](#)
- [Expression-level preferences](#)
- [Code style language rules](#)
- [Code style rules reference](#)

Simplify 'default' expression (IDE0034)

9/20/2022 • 2 minutes to read • [Edit Online](#)

PROPERTY	VALUE
Rule ID	IDE0034
Title	Simplify <code>default</code> expression
Category	Style
Subcategory	Language rules (expression-level preferences)
Applicable languages	C# 7.1+
Options	<code>csharp_prefer_simple_default_expression</code>

Overview

This style rule concerns using the [default literal for default value expressions](#) when the compiler can infer the type of the expression.

Options

Options specify the behavior that you want the rule to enforce. For information about configuring options, see [Option format](#).

`csharp_prefer_simple_default_expression`

PROPERTY	VALUE	DESCRIPTION
Option name	<code>csharp_prefer_simple_default_expression</code>	
Option values	<code>true</code>	Prefer <code>default</code> over <code>default(T)</code>
	<code>false</code>	Prefer <code>default(T)</code> over <code>default</code>

```
// csharp_prefer_simple_default_expression = true
void DoWork(CancellationToken cancellationToken = default) { ... }

// csharp_prefer_simple_default_expression = false
void DoWork(CancellationToken cancellationToken = default(CancellationToken)) { ... }
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and

then re-enable the rule.

```
#pragma warning disable IDE0034
// The code that's violating the rule is on this line.
#pragma warning restore IDE0034
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.IDE0034.severity = none
```

To disable all of the code-style rules, set the severity for the category `Style` to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [default literal](#)
- [Expression-level preferences](#)
- [Code style language rules](#)
- [Code style rules reference](#)

Use inferred member names (IDE0037)

9/20/2022 • 2 minutes to read • [Edit Online](#)

PROPERTY	VALUE
Rule ID	IDE0037
Title	Use inferred member name
Category	Style
Subcategory	Language rules (expression-level preferences)
Applicable languages	C# 7.1+ and Visual Basic 15+
Introduced version	Visual Studio 2017 version 15.6
Options	<code>dotnet_style_prefer_inferred_tuple_names</code>
	<code>dotnet_style_prefer_inferred_anonymous_type_member_names</code>

Overview

This rule enforces whether inferred [tuple](#) element names and inferred [anonymous type](#) member names are preferred when the tuple or anonymous type is declared.

Options

Set the values of the rule's associated options to specify whether inferred or explicit names are preferred for tuple elements and anonymous type members.

- [dotnet_style_prefer_inferred_tuple_names](#)
- [dotnet_style_prefer_inferred_anonymous_type_member_names](#)

For information about configuring options, see [Option format](#).

dotnet_style_prefer_inferred_tuple_names

PROPERTY	VALUE	DESCRIPTION
Option name	<code>dotnet_style_prefer_inferred_tuple_names</code>	
Option values	<code>true</code>	Prefer inferred tuple element names
	<code>false</code>	Prefer explicit tuple element names
Default option value	<code>true</code>	

```
// dotnet_style_prefer_inferred_tuple_names = true
var tuple = (age, name);

// dotnet_style_prefer_inferred_tuple_names = false
var tuple = (age: age, name: name);
```

```
' dotnet_style_prefer_inferred_tuple_names = true
Dim tuple = (name, age)

' dotnet_style_prefer_inferred_tuple_names = false
Dim tuple = (name:=name, age:=age)
```

dotnet_style_prefer_inferred_anonymous_type_member_names

PROPERTY	VALUE	DESCRIPTION
Option name	dotnet_style_prefer_inferred_anonymous_type_member_names	
Option values	<code>true</code>	Prefer inferred anonymous type member names
	<code>false</code>	Prefer explicit anonymous type member names
Default option value	<code>true</code>	

```
// dotnet_style_prefer_inferred_anonymous_type_member_names = true
var anon = new { age, name };

// dotnet_style_prefer_inferred_anonymous_type_member_names = false
var anon = new { age = age, name = name };
```

```
' dotnet_style_prefer_inferred_anonymous_type_member_names = true
Dim anon = New With {name, age}

' dotnet_style_prefer_inferred_anonymous_type_member_names = false
Dim anon = New With {.name = name, .age = age}
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0037
// The code that's violating the rule is on this line.
#pragma warning restore IDE0037
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.IDE0037.severity = none
```

To disable all of the code-style rules, set the severity for the category `Style` to `none` in the [configuration file](#).

```
[*.{cs,vb}]\ndotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Use explicitly provided tuple name \(IDE0033\)](#)
- [Expression-level preferences](#)
- [Code style language rules](#)
- [Code style rules reference](#)

Use local function instead of lambda (IDE0039)

9/20/2022 • 2 minutes to read • [Edit Online](#)

PROPERTY	VALUE
Rule ID	IDE0039
Title	Use local function instead of lambda
Category	Style
Subcategory	Language rules (expression-level preferences)
Applicable languages	C# 7.0+
Options	<code>csharp_style_prefer_local_over_anonymous_function</code>

Overview

This style rule concerns the use of [local functions](#) versus [lambda expressions](#) (anonymous functions).

Options

Options specify the behavior that you want the rule to enforce. For information about configuring options, see [Option format](#).

`csharp_style_prefer_local_over_anonymous_function`

PROPERTY	VALUE	DESCRIPTION
Option name	<code>csharp_style_prefer_local_over_anonymous_function</code>	
Option values	<code>true</code>	Prefer local functions over anonymous functions
	<code>false</code>	Prefer anonymous functions over local functions
Default option value	<code>true</code>	

```
// csharp_style_prefer_local_over_anonymous_function = true
int fibonacci(int n)
{
    return n <= 1 ? 1 : fibonacci(n-1) + fibonacci(n-2);
}

// csharp_style_prefer_local_over_anonymous_function = false
Func<int, int> fibonacci = null;
fibonacci = (int n) =>
{
    return n <= 1 ? 1 : fibonacci(n - 1) + fibonacci(n - 2);
};
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0039
// The code that's violating the rule is on this line.
#pragma warning restore IDE0039
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.IDE0039.severity = none
```

To disable all of the code-style rules, set the severity for the category `Style` to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Expression-level preferences](#)
- [Code style language rules](#)
- [Code style rules reference](#)

Deconstruct variable declaration (IDE0042)

9/20/2022 • 2 minutes to read • [Edit Online](#)

PROPERTY	VALUE
Rule ID	IDE0042
Title	Deconstruct variable declaration
Category	Style
Subcategory	Language rules (expression-level preferences)
Applicable languages	C# 7.0+
Options	<code>csharp_style_deconstructed_variable_declaration</code>

Overview

This style rule concerns the use of [deconstruction](#) in variable declarations, when possible.

Options

Options specify the behavior that you want the rule to enforce. For information about configuring options, see [Option format](#).

`csharp_style_deconstructed_variable_declaration`

PROPERTY	VALUE	DESCRIPTION
Option name	<code>csharp_style_deconstructed_variable_d eclaration</code>	
Option values	<code>true</code>	Prefer deconstructed variable declaration
	<code>false</code>	Do not prefer deconstruction in variable declarations
Default option value	<code>true</code>	

```
// csharp_style_deconstructed_variable_declaration = true
var (name, age) = GetPersonTuple();
Console.WriteLine($"{name} {age}");

(int x, int y) = GetPointTuple();
Console.WriteLine($"{x} {y}");

// csharp_style_deconstructed_variable_declaration = false
var person = GetPersonTuple();
Console.WriteLine($"{person.name} {person.age}");

(int x, int y) point = GetPointTuple();
Console.WriteLine($"{point.x} {point.y}");
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0042
// The code that's violating the rule is on this line.
#pragma warning restore IDE0042
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.IDE0042.severity = none
```

To disable all of the code-style rules, set the severity for the category `Style` to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Expression-level preferences](#)
- [Code style language rules](#)
- [Code style rules reference](#)

Use conditional expression for assignment (IDE0045)

9/20/2022 • 2 minutes to read • [Edit Online](#)

PROPERTY	VALUE
Rule ID	IDE0045
Title	Use conditional expression for assignment
Category	Style
Subcategory	Language rules (expression-level preferences)
Applicable languages	C# and Visual Basic
Introduced version	Visual Studio 2017 version 15.8
Options	<code>dotnet_style_prefer_conditional_expression_over_assignment</code>

Overview

This style rule concerns the use of a [ternary conditional expression](#) versus an [if-else statement](#) for assignments that require conditional logic.

Options

Options specify the behavior that you want the rule to enforce. For information about configuring options, see [Option format](#).

`dotnet_style_prefer_conditional_expression_over_assignment`

PROPERTY	VALUE	DESCRIPTION
Option name	<code>dotnet_style_prefer_conditional_expression_over_assignment</code>	
Option values	<code>true</code>	Prefer assignments with a ternary conditional
	<code>false</code>	Prefer assignments with an if-else statement
Default option value	<code>true</code>	

```
// dotnet_style_prefer_conditional_expression_over_assignment = true
string s = expr ? "hello" : "world";

// dotnet_style_prefer_conditional_expression_over_assignment = false
string s;
if (expr)
{
    s = "hello";
}
else
{
    s = "world";
}
```

```
' dotnet_style_prefer_conditional_expression_over_assignment = true
Dim s As String = If(expr, "hello", "world")

' dotnet_style_prefer_conditional_expression_over_assignment = false
Dim s As String
If expr Then
    s = "hello"
Else
    s = "world"
End If
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0045
// The code that's violating the rule is on this line.
#pragma warning restore IDE0045
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.IDE0045.severity = none
```

To disable all of the code-style rules, set the severity for the category `Style` to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Use conditional expression for return \(IDE0046\)](#)
- [Expression-level preferences](#)
- [Code style language rules](#)
- [Code style rules reference](#)

Use conditional expression for return (IDE0046)

9/20/2022 • 2 minutes to read • [Edit Online](#)

PROPERTY	VALUE
Rule ID	IDE0046
Title	Use conditional expression for return
Category	Style
Subcategory	Language rules (expression-level preferences)
Applicable languages	C# and Visual Basic
Introduced version	Visual Studio 2017 version 15.8
Options	<code>dotnet_style_prefer_conditional_expression_over_return</code>

Overview

This style rule concerns the use of a [ternary conditional expression](#) versus an [if-else statement](#) for return statements that require conditional logic.

Options

Options specify the behavior that you want the rule to enforce. For information about configuring options, see [Option format](#).

`dotnet_style_prefer_conditional_expression_over_return`

PROPERTY	VALUE	DESCRIPTION
Option name	<code>dotnet_style_prefer_conditional_expression_over_return</code>	
Option values	<code>true</code>	Prefer return statements to use a ternary conditional
	<code>false</code>	Prefer return statements to use an if-else statement
Default option value	<code>true</code>	

```
// dotnet_style_prefer_conditional_expression_over_return = true
return expr ? "hello" : "world"

// dotnet_style_prefer_conditional_expression_over_return = false
if (expr)
{
    return "hello";
}
else
{
    return "world";
}
```

```
' dotnet_style_prefer_conditional_expression_over_return = true
Return If(expr, "hello", "world")

' dotnet_style_prefer_conditional_expression_over_return = false
If expr Then
    Return "hello"
Else
    Return "world"
End If
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0046
// The code that's violating the rule is on this line.
#pragma warning restore IDE0046
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.IDE0046.severity = none
```

To disable all of the code-style rules, set the severity for the category `Style` to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Use conditional expression for assignment](#)
- [Expression-level preferences](#)
- [Code style language rules](#)
- [Code style rules reference](#)

Convert anonymous type to tuple (IDE0050)

9/20/2022 • 2 minutes to read • [Edit Online](#)

IMPORTANT

This style rule was removed and converted to a Visual Studio refactoring in Visual Studio 2022. For information about the refactoring, see [Convert anonymous type to tuple](#).

PROPERTY	VALUE
Rule ID	IDE0050
Title	Convert anonymous type to tuple
Category	Style
Subcategory	Language rules (expression-level preferences)
Applicable languages	C# and Visual Basic

Overview

This rule recommends use of [tuples](#) over [anonymous types](#), when the anonymous type has two or more fields.

Options

This rule has no associated code-style options.

Example

```
// Code with violations
var t1 = new { a = 1, b = 2 };

// Fixed code
var t1 = (a: 1, b: 2);
```

```
' Code with violations
Dim t1 = New With { .a = 1, .b = 2 }

' Fixed code
Dim t1 = (a:=1, b:=2)
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0050
// The code that's violating the rule is on this line.
#pragma warning restore IDE0050
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.IDE0050.severity = none
```

To disable all of the code-style rules, set the severity for the category `Style` to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Tuples](#)
- [Anonymous types](#)
- [Expression-level preferences](#)
- [Code style language rules](#)
- [Code style rules reference](#)

Use compound assignment (IDE0054 and IDE0074)

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article describes two related rules, [IDE0054](#) and [IDE0074](#).

PROPERTY	VALUE
Rule ID	IDE0054
Title	Use compound assignment
Category	Style
Subcategory	Language rules (expression-level preferences)
Applicable languages	C# and Visual Basic
Options	dotnet_style_prefer_compound_assignment

PROPERTY	VALUE
Rule ID	IDE0074
Title	Use coalesce compound assignment
Category	Style
Subcategory	Language rules (expression-level preferences)
Applicable languages	C# and Visual Basic
Options	dotnet_style_prefer_compound_assignment

Overview

These rules concern the use of [compound assignment](#). [IDE0074](#) is reported for coalesce compound assignments and [IDE0054](#) is reported for other compound assignments.

Options

The option value specifies whether or not compound assignments are desired.

For information about configuring options, see [Option format](#).

dotnet_style_prefer_compound_assignment

PROPERTY	VALUE	DESCRIPTION

PROPERTY	VALUE	DESCRIPTION
Option name	dotnet_style_prefer_compound_assignment	
Option values	<code>true</code>	Prefer compound assignment expressions
	<code>false</code>	Don't prefer compound assignment expressions
Default option value	<code>true</code>	

```
// dotnet_style_prefer_compound_assignment = true
x += 5;

// dotnet_style_prefer_compound_assignment = false
x = x + 5;
```

```
' dotnet_style_prefer_compound_assignment = true
x += 5

' dotnet_style_prefer_compound_assignment = false
x = x + 5
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0054 // Or IDE0074
// The code that's violating the rule is on this line.
#pragma warning restore IDE0054 // Or IDE0074
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.IDE0054.severity = none
dotnet_diagnostic.IDE0074.severity = none
```

To disable all of the code-style rules, set the severity for the category `Style` to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Expression-level preferences](#)
- [Code style language rules](#)
- [Code style rules reference](#)

Use index operator (IDE0056)

9/20/2022 • 2 minutes to read • [Edit Online](#)

PROPERTY	VALUE
Rule ID	IDE0056
Title	Use index operator
Category	Style
Subcategory	Language rules (expression-level preferences)
Applicable languages	C# 8.0+
Options	<code>csharp_style_prefer_index_operator</code>

Overview

This style rule concerns the use of the [index-from-end operator \(`^`\)](#), which is available in C# 8.0 and later.

Options

Options specify the behavior that you want the rule to enforce. For information about configuring options, see [Option format](#).

`csharp_style_prefer_index_operator`

PROPERTY	VALUE	DESCRIPTION
Option name	<code>csharp_style_prefer_index_operator</code>	
Option values	<code>true</code>	Prefer to use the <code>^</code> operator when calculating an index from the end of a collection
	<code>false</code>	Prefer not to use the <code>^</code> operator when calculating an index from the end of a collection
Default option value	<code>true</code>	

```
// csharp_style_prefer_index_operator = true
string[] names = { "Archimedes", "Pythagoras", "Euclid" };
var index = names[^1];

// csharp_style_prefer_index_operator = false
string[] names = { "Archimedes", "Pythagoras", "Euclid" };
var index = names[names.Length - 1];
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0056
// The code that's violating the rule is on this line.
#pragma warning restore IDE0056
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.cs,vb]
dotnet_diagnostic.IDE0056.severity = none
```

To disable all of the code-style rules, set the severity for the category `Style` to `none` in the [configuration file](#).

```
[*.cs,vb]
dotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [index-from-end operator](#)
- [Expression-level preferences](#)
- [Code style language rules](#)
- [Code style rules reference](#)

Use range operator (IDE0057)

9/20/2022 • 2 minutes to read • [Edit Online](#)

PROPERTY	VALUE
Rule ID	IDE0057
Title	Use range operator
Category	Style
Subcategory	Language rules (expression-level preferences)
Applicable languages	C# 8.0+
Options	<code>csharp_style_prefer_range_operator</code>

Overview

This style rule concerns the use of the [range operator](#) (`..`), which is available in C# 8.0 and later.

Options

Options specify the behavior that you want the rule to enforce. For information about configuring options, see [Option format](#).

`csharp_style_prefer_range_operator`

PROPERTY	VALUE	DESCRIPTION
Option name	<code>csharp_style_prefer_range_operator</code>	
Option values	<code>true</code>	Prefer to use the range operator <code>..</code> when extracting a "slice" of a collection
	<code>false</code>	Prefer <i>not</i> to use the range operator <code>..</code> when extracting a "slice" of a collection
Default option value	<code>true</code>	

```
// csharp_style_prefer_range_operator = true
string sentence = "the quick brown fox";
var sub = sentence[0..^4];

// csharp_style_prefer_range_operator = false
string sentence = "the quick brown fox";
var sub = sentence.Substring(0, sentence.Length - 4);
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0057
// The code that's violating the rule is on this line.
#pragma warning restore IDE0057
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.cs,vb]
dotnet_diagnostic.IDE0057.severity = none
```

To disable all of the code-style rules, set the severity for the category `Style` to `none` in the [configuration file](#).

```
[*.cs,vb]
dotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Expression-level preferences](#)
- [Code style language rules](#)
- [Code style rules reference](#)

Use 'System.HashCode.Combine' (IDE0070)

9/20/2022 • 2 minutes to read • [Edit Online](#)

PROPERTY	VALUE
Rule ID	IDE0070
Title	Use <code>System.HashCode.Combine</code>
Category	Style
Subcategory	Language rules (expression-level preferences)
Applicable languages	C# and Visual Basic

Overview

This rule recommends the use of the [System.HashCode.Combine](#) method to compute a hash code instead of using custom hash code computation logic.

Options

This rule has no associated code-style options.

Example

```
class B
{
    public override int GetHashCode() => 0;
}

class C : B
{
    int j;

    // Code with violations
    public override int GetHashCode()
    {
        // IDE0070: GetHashCode can be simplified.
        var hashCode = 339610899;
        hashCode = hashCode * -1521134295 + base.GetHashCode();
        hashCode = hashCode * -1521134295 + j.GetHashCode();
        return hashCode;
    }

    // Fixed code
    public override int GetHashCode()
    {
        return System.HashCode.Combine(base.GetHashCode(), j);
    }
}
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0070
// The code that's violating the rule is on this line.
#pragma warning restore IDE0070
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.cs,vb]
dotnet_diagnostic.IDE0070.severity = none
```

To disable all of the code-style rules, set the severity for the category `Style` to `none` in the [configuration file](#).

```
[*.cs,vb]
dotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Expression-level preferences](#)
- [Code style language rules](#)
- [Code style rules reference](#)

Simplify interpolation (IDE0071)

9/20/2022 • 2 minutes to read • [Edit Online](#)

PROPERTY	VALUE
Rule ID	IDE0071
Title	Simplify interpolation
Category	Style
Subcategory	Language rules (expression-level preferences)
Applicable languages	C# 6.0+ and Visual Basic 14+
Options	<code>dotnet_style_prefer_simplified_interpolation</code>

Overview

This style rule concerns with simplification of [interpolated strings](#) to improve code readability. It recommends removal of certain explicit method calls, such as `ToString()`, when the same method would be implicitly invoked by the compiler if the explicit method call is removed.

Options

Options specify the behavior that you want the rule to enforce. For information about configuring options, see [Option format](#).

`dotnet_style_prefer_simplified_interpolation`

PROPERTY	VALUE	DESCRIPTION
Option name	<code>dotnet_style_prefer_simplified_interpolation</code>	
Option values	<code>true</code>	Prefer simplified interpolated strings
	<code>false</code>	Do not prefer simplified interpolated strings
Default option value	<code>true</code>	

```
// dotnet_style_prefer_simplified_interpolation = true
var str = $"prefix {someValue} suffix";

// dotnet_style_prefer_simplified_interpolation = false
var str = $"prefix {someValue.ToString()} suffix";
```

```
' dotnet_style_prefer_simplified_interpolation = true
Dim str = $"prefix {someValue} suffix"

' dotnet_style_prefer_simplified_interpolation = false
Dim str = $"prefix {someValue.ToString()} suffix"
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0071
// The code that's violating the rule is on this line.
#pragma warning restore IDE0071
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.IDE0071.severity = none
```

To disable all of the code-style rules, set the severity for the category `Style` to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [interpolated strings](#)
- [Expression-level preferences](#)
- [Code style language rules](#)
- [Code style rules reference](#)

Add missing cases to switch expression (IDE0072)

9/20/2022 • 2 minutes to read • [Edit Online](#)

PROPERTY	VALUE
Rule ID	IDE0072
Title	Add missing cases to switch expression
Category	Style
Subcategory	Language rules (expression-level preferences)
Applicable languages	C# 8.0+

Overview

This rule concerns specifying all the missing cases for a [switch expression](#). A switch expression is considered incomplete with missing cases in following scenarios:

- When an [enum](#) switch expression is missing cases for one or more enum members.
- When the fall-through case `_` is missing.

Options

This rule has no associated code-style options.

Example

```
enum E
{
    A,
    B
}

class C
{
    // Code with violations
    int M(E e)
    {
        // IDE0072: Add missing cases
        return e switch
        {
            E.A => 0,
            _ => -1,
        };
    }

    // Fixed code
    int M(E e)
    {
        return e switch
        {
            E.A => 0,
            E.B => 1,
            _ => -1,
        };
    }
}
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0072
// The code that's violating the rule is on this line.
#pragma warning restore IDE0072
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.IDE0072.severity = none
```

To disable all of the code-style rules, set the severity for the category `Style` to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Switch expression](#)
- [Add missing cases to switch statement \(IDE0010\)](#)
- [Expression-level preferences](#)

- [Code style language rules](#)
- [Code style rules reference](#)

Use compound assignment (IDE0054 and IDE0074)

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article describes two related rules, [IDE0054](#) and [IDE0074](#).

PROPERTY	VALUE
Rule ID	IDE0054
Title	Use compound assignment
Category	Style
Subcategory	Language rules (expression-level preferences)
Applicable languages	C# and Visual Basic
Options	dotnet_style_prefer_compound_assignment

PROPERTY	VALUE
Rule ID	IDE0074
Title	Use coalesce compound assignment
Category	Style
Subcategory	Language rules (expression-level preferences)
Applicable languages	C# and Visual Basic
Options	dotnet_style_prefer_compound_assignment

Overview

These rules concern the use of [compound assignment](#). [IDE0074](#) is reported for coalesce compound assignments and [IDE0054](#) is reported for other compound assignments.

Options

The option value specifies whether or not compound assignments are desired.

For information about configuring options, see [Option format](#).

dotnet_style_prefer_compound_assignment

PROPERTY	VALUE	DESCRIPTION

PROPERTY	VALUE	DESCRIPTION
Option name	dotnet_style_prefer_compound_assignment	
Option values	<code>true</code>	Prefer compound assignment expressions
	<code>false</code>	Don't prefer compound assignment expressions
Default option value	<code>true</code>	

```
// dotnet_style_prefer_compound_assignment = true
x += 5;

// dotnet_style_prefer_compound_assignment = false
x = x + 5;
```

```
' dotnet_style_prefer_compound_assignment = true
x += 5

' dotnet_style_prefer_compound_assignment = false
x = x + 5
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0054 // Or IDE0074
// The code that's violating the rule is on this line.
#pragma warning restore IDE0054 // Or IDE0074
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.IDE0054.severity = none
dotnet_diagnostic.IDE0074.severity = none
```

To disable all of the code-style rules, set the severity for the category `Style` to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Expression-level preferences](#)
- [Code style language rules](#)
- [Code style rules reference](#)

Simplify conditional expression (IDE0075)

9/20/2022 • 2 minutes to read • [Edit Online](#)

PROPERTY	VALUE
Rule ID	IDE0075
Title	Simplify conditional expression
Category	Style
Subcategory	Language rules (expression-level preferences)
Applicable languages	C# and Visual Basic
Options	<code>dotnet_style_prefer_simplified_boolean_expressions</code>

Overview

This style rule concerns simplifying [conditional expressions](#) that return a constant value of `true` or `false` versus retaining conditional expressions with explicit `true` or `false` return values.

Options

Options specify the behavior that you want the rule to enforce. For information about configuring options, see [Option format](#).

`dotnet_style_prefer_simplified_boolean_expressions`

PROPERTY	VALUE	DESCRIPTION
Option name	<code>dotnet_style_prefer_simplified_boolean_expressions</code>	
Option values	<code>true</code>	Prefer simplified conditional expressions
	<code>false</code>	Do not prefer simplified conditional expressions
Default option value	<code>true</code>	

```
// dotnet_style_prefer_simplified_boolean_expressions = true
var result1 = M1() && M2();
var result2 = M1() || M2();

// dotnet_style_prefer_simplified_boolean_expressions = false
var result1 = M1() && M2() ? true : false;
var result2 = M1() ? true : M2();
```

```
' dotnet_style_prefer_simplified_boolean_expressions = true
Dim result1 = M1() AndAlso M2()
Dim result2 = M1() OrElse M2()

' dotnet_style_prefer_simplified_boolean_expressions = false
Dim result1 = If (M1() AndAlso M2(), True, False)
Dim result2 = If (M1(), True, M2())
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0075
// The code that's violating the rule is on this line.
#pragma warning restore IDE0075
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.IDE0075.severity = none
```

To disable all of the code-style rules, set the severity for the category `Style` to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Conditional operator](#)
- [Expression-level preferences](#)
- [Code style language rules](#)
- [Code style rules reference](#)

Convert `typeof` to `nameof` (IDE0082)

9/20/2022 • 2 minutes to read • [Edit Online](#)

PROPERTY	VALUE
Rule ID	IDE0082
Title	Convert <code>typeof</code> to <code>nameof</code>
Category	Style
Subcategory	Language rules (expression-level preferences)
Applicable languages	C# and Visual Basic

Overview

This style rule recommends use of the [nameof operator](#) over the [typeof operator](#) followed by [Name](#) member access. It only fires when the name will be identical in both cases.

Options

This rule has no associated code-style options.

Example

```
// Code with violations
var n1 = typeof(T).Name;
var n2 = typeof(int).Name;

// Fixed code
var n1 = nameof(T);
var n2 = nameof(Int32);
```

```
' Code with violations
Dim n1 = GetType(T).Name
Dim n2 = GetType(Integer).Name

' Fixed code
Dim n1 = NameOf(T)
Dim n2 = NameOf(Int32)
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0082
// The code that's violating the rule is on this line.
#pragma warning restore IDE0082
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.IDE0082.severity = none
```

To disable all of the code-style rules, set the severity for the category `Style` to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [nameof operator](#)
- [typeof operator](#)
- [Expression-level preferences](#)
- [Code style language rules](#)
- [Code style rules reference](#)

Simplify `new` expression (IDE0090)

9/20/2022 • 2 minutes to read • [Edit Online](#)

PROPERTY	VALUE
Rule ID	IDE0090
Title	Simplify <code>new</code> expression
Category	Style
Subcategory	Language rules (expression-level preferences)
Applicable languages	C# 9.0+
Options	<code>csharp_style_implicit_object_creation_when_type_is_apparent</code>

Overview

This style rule concerns the use of C# 9.0 [target-typed new](#) expressions when the created type is apparent.

Options

Options specify the behavior that you want the rule to enforce. For information about configuring options, see [Option format](#).

`csharp_style_implicit_object_creation_when_type_is_apparent`

PROPERTY	VALUE	DESCRIPTION
Option name	<code>csharp_style_implicit_object_creation_when_type_is_apparent</code>	
Option values	<code>true</code>	Prefer target-typed <code>new</code> expressions when created type is apparent
	<code>false</code>	Do not prefer target-typed <code>new</code> expressions
Default option value	<code>true</code>	

```
// csharp_style_implicit_object_creation_when_type_is_apparent = true
C c = new();
C c2 = new() { Field = 0 };

// csharp_style_implicit_object_creation_when_type_is_apparent = false
C c = new C();
C c2 = new C() { Field = 0 };
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0090
// The code that's violating the rule is on this line.
#pragma warning restore IDE0090
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.cs,vb]
dotnet_diagnostic.IDE0090.severity = none
```

To disable all of the code-style rules, set the severity for the category `Style` to `none` in the [configuration file](#).

```
[*.cs,vb]
dotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Target-typed new expressions](#)
- [Expression-level preferences](#)
- [Code style language rules](#)
- [Code style rules reference](#)

Use tuple to swap values (IDE0180)

9/20/2022 • 2 minutes to read • [Edit Online](#)

PROPERTY	VALUE
Rule ID	IDE0180
Title	Use tuple to swap values
Category	Style
Subcategory	Language rules (expression-level preferences)
Applicable languages	C#
Options	<code>csharp_style_prefer_tuple_swap</code>

Overview

This style rule flags code that swaps two values using multiple lines of code instead of using a [tuple](#).

Options

Options specify the behavior that you want the rule to enforce. For information about configuring options, see [Option format](#).

`csharp_style_prefer_tuple_swap`

PROPERTY	VALUE	DESCRIPTION
Option name	<code>csharp_style_prefer_tuple_swap</code>	
Option values	<code>true</code>	Prefer using a tuple to swap two values.
	<code>false</code>	Disables the rule.
Default option value	<code>true</code>	

Example

```
List<int> numbers = new List<int>() { 5, 6, 4 };

// Violates IDE0180.
int temp = numbers[0];
numbers[0] = numbers[1];
numbers[1] = temp;

// Fixed code.
(numbers[1], numbers[0]) = (numbers[0], numbers[1]);
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0180
// The code that's violating the rule is on this line.
#pragma warning restore IDE0180
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.IDE0180.severity = none
```

To disable all of the code-style rules, set the severity for the category `Style` to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Expression-level preferences](#)
- [Code style language rules](#)
- [Code style rules reference](#)

Namespace declaration preferences (IDE0160 and IDE0161)

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article describes two related rules, [IDE0160](#) and [IDE0161](#).

PROPERTY	VALUE
Rule ID	IDE0160
Title	Use block-scoped namespace
Category	Style
Subcategory	Language rules (namespace preferences)
Applicable languages	C#
Options	<code>csharp_style_namespace_declarations</code>

PROPERTY	VALUE
Rule ID	IDE0161
Title	Use file-scoped namespace
Category	Style
Subcategory	Language rules (namespace preferences)
Applicable languages	C#
Options	<code>csharp_style_namespace_declarations</code>

Overview

These rules apply to [namespace declarations](#). For [IDE0161](#) to report violations when block-scoped namespaces are used, you must set the associated option to [file_scoped](#).

Options

The option value specifies whether namespace declarations should be block scoped or file scoped. By default, namespace declarations are block-scoped. This option is used by Visual Studio to determine how namespaces are declared when new code files are added to projects. Visual Studio honors the option value even if both [IDE0160](#) and [IDE0161](#) are disabled.

For information about configuring options, see [Option format](#).

csharp_style_namespace_declarations

PROPERTY	VALUE	DESCRIPTION
Option name	csharp_style_namespace_declarations	
Applicable languages	C#	
Introduced version	Visual Studio 2019 version 16.10	
Option values	<code>block_scoped</code>	Namespace declarations should be block scoped.
	<code>file_scoped</code>	Namespace declarations should be file scoped.
Default option value	<code>block_scoped</code>	

```
// csharp_style_namespace_declarations = block_scoped
using System;

namespace Convention
{
    class C
    {
    }
}

// csharp_style_namespace_declarations = file_scoped
using System;

namespace Convention;
class C
{
}
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0160 // Or IDE0161
// The code that's violating the rule is on this line.
#pragma warning restore IDE0160 // Or IDE0161
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.IDE0160.severity = none
dotnet_diagnostic.IDE0161.severity = none
```

To disable all of the code-style rules, set the severity for the category `Style` to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Namespace declarations \(C#\)](#)
- [Code style language rules](#)
- [Code style rules reference](#)

Null-checking preferences

9/20/2022 • 2 minutes to read • [Edit Online](#)

.NET null-checking preferences

The style rules in this section concern the following null-checking preferences that are common to C# and Visual Basic:

- [Use coalesce expression \(IDE0029 and IDE0030\)](#)
- [Use null propagation \(IDE0031\)](#)
- [Use is null check \(IDE0041\)](#)
- [Prefer `null` check over type check \(IDE0150\)](#)

C# null-checking preferences

The style rules in this section concern the following null-checking preferences that are specific to C#:

- [Use throw expression \(IDE0016\)](#)
- [Use conditional delegate call \(IDE1005\)](#)

See also

- [Code style rules reference](#)
- [Code style language rules](#)

Use throw expression (IDE0016)

9/20/2022 • 2 minutes to read • [Edit Online](#)

PROPERTY	VALUE
Rule ID	IDE0016
Title	Use throw expression
Category	Style
Subcategory	Language rules (null-checking preferences)
Applicable languages	C# 7.0+
Options	<code>csharp_style_throw_expression</code>

Overview

This style rule concerns the use of [throw expressions](#) instead of `throw` statements. Set the severity of rule `IDE0016` to define how the rule should be enforced, for example, as a warning or an error.

Options

The associated option for this rule specifies whether you prefer `throw` expressions or `throw` statements.

For more information about configuring options, see [Option format](#).

`csharp_style_throw_expression`

PROPERTY	VALUE	DESCRIPTION
Option name	<code>csharp_style_throw_expression</code>	
Option values	<code>true</code>	Prefer to use <code>throw</code> expressions instead of <code>throw</code> statements
	<code>false</code>	Prefer to use <code>throw</code> statements instead of <code>throw</code> expressions
Default option value	<code>true</code>	

```
// csharp_style_throw_expression = true
this.s = s ?? throw new ArgumentNullException(nameof(s));

// csharp_style_throw_expression = false
if (s == null) { throw new ArgumentNullException(nameof(s)); }
this.s = s;
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0016
// The code that's violating the rule is on this line.
#pragma warning restore IDE0016
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.cs,vb]
dotnet_diagnostic.IDE0016.severity = none
```

To disable all of the code-style rules, set the severity for the category `Style` to `none` in the [configuration file](#).

```
[*.cs,vb]
dotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [throw expressions](#)
- [Null-checking preferences](#)
- [Code style language rules](#)
- [Code style rules reference](#)

Use coalesce expression (IDE0029 and IDE0030)

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article describes two related rules, [IDE0029](#) and [IDE0030](#).

PROPERTY	VALUE
Rule ID	IDE0029
Title	Use coalesce expression (non-nullable types)
Category	Style
Subcategory	Language rules (null-checking preferences)
Applicable languages	C# and Visual Basic
Options	dotnet_style_coalesce_expression

PROPERTY	VALUE
Rule ID	IDE0030
Title	Use coalesce expression (nullable types)
Category	Style
Subcategory	Language rules (null-checking preferences)
Applicable languages	C# and Visual Basic
Options	dotnet_style_coalesce_expression

Overview

These style rules concern the use of [null-coalescing expressions](#), for example, `x ?? y`, versus [ternary conditional expressions](#) with `null` checks, for example, `x != null ? x : y`. The two related rule IDs differ with respect to the nullability of the expressions:

- [IDE0029](#): Used when non-nullable expressions are involved. For example, this rule could recommend `x ?? y` instead of `x != null ? x : y` when `x` and `y` are non-nullable reference types.
- [IDE0030](#): Used when nullable expressions are involved. For example, this rule could recommend `x ?? y` instead of `x != null ? x : y` when `x` and `y` are [nullable value types](#) or [nullable reference types](#).

Options

Set the value of the associated option to specify whether null-coalescing expressions or ternary operator checking is preferred. The rule enforces whichever option you choose.

For more information about configuring options, see [Option format](#).

dotnet_style_coalesce_expression

PROPERTY	VALUE	DESCRIPTION
Option name	dotnet_style_coalesce_expression	
Option values	true	Prefer null coalescing expressions to ternary operator checking
	false	Prefer ternary operator checking to null coalescing expressions
Default option value	true	

```
// dotnet_style_coalesce_expression = true
var v = x ?? y;

// dotnet_style_coalesce_expression = false
var v = x != null ? x : y; // or
var v = x == null ? y : x;
```

```
' dotnet_style_coalesce_expression = true
Dim v = If(x, y)

' dotnet_style_coalesce_expression = false
Dim v = If(x Is Nothing, y, x) ' or
Dim v = If(x IsNot Nothing, x, y)
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0029 // Or IDE0030
// The code that's violating the rule is on this line.
#pragma warning restore IDE0029 // Or IDE0030
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.IDE0029.severity = none
dotnet_diagnostic.IDE0030.severity = none
```

To disable all of the code-style rules, set the severity for the category `Style` to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Null-checking preferences](#)
- [Code style language rules](#)
- [Code style rules reference](#)

Use null propagation (IDE0031)

9/20/2022 • 2 minutes to read • [Edit Online](#)

PROPERTY	VALUE
Rule ID	IDE0031
Title	Use null propagation
Category	Style
Subcategory	Language rules (null-checking preferences)
Applicable languages	C# 6.0+ and Visual Basic 14+
Options	<code>dotnet_style_null_propagation</code>

Overview

This style rule concerns the use of the [null-conditional operator](#) versus a [ternary conditional expression](#) with null check.

Options

Set the value of the associated option to specify whether null-conditional operators or ternary conditional expressions with null checks.

For more information about configuring options, see [Option format](#).

`dotnet_style_null_propagation`

PROPERTY	VALUE	DESCRIPTION
Option name	<code>dotnet_style_null_propagation</code>	
Option values	<code>true</code>	Prefer to use null-conditional operator when possible
	<code>false</code>	Prefer to use ternary null checking where possible
Default option value	<code>true</code>	

```
// dotnet_style_null_propagation = true
var v = o?.ToString();

// dotnet_style_null_propagation = false
var v = o == null ? null : o.ToString(); // or
var v = o != null ? o.ToString() : null;
```

```
' dotnet_style_null_propagation = true
Dim v = o?.ToString()

' dotnet_style_null_propagation = false
Dim v = If(o Is Nothing, Nothing, o.ToString()) ' or
Dim v = If(o IsNot Nothing, o.ToString(), Nothing)
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0031
// The code that's violating the rule is on this line.
#pragma warning restore IDE0031
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.IDE0031.severity = none
```

To disable all of the code-style rules, set the severity for the category `Style` to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Null-checking preferences](#)
- [Code style language rules](#)
- [Code style rules reference](#)

Use 'is null' check (IDE0041)

9/20/2022 • 2 minutes to read • [Edit Online](#)

PROPERTY	VALUE
Rule ID	IDE0041
Title	Use 'is null' check
Category	Style
Subcategory	Language rules (null-checking preferences)
Applicable languages	C# 6.0+ and Visual Basic 14+
Introduced version	Visual Studio 2017 version 15.7
Options	<code>dotnet_style_prefer_is_null_check_over_reference_equality_method</code>

Overview

This style rule concerns the use of a null check with pattern-matching versus calling the reference-equality method [Object.ReferenceEquals\(Object, Object\)](#).

Options

Options specify the behavior that you want the rule to enforce. For information about configuring options, see [Option format](#).

`dotnet_style_prefer_is_null_check_over_reference_equality_method`

PROPERTY	VALUE	DESCRIPTION
Option name	<code>dotnet_style_prefer_is_null_check_over_reference_equality_method</code>	
Option values	<code>true</code>	Prefer <code>is null</code> check
	<code>false</code>	Prefer reference equality method
Default option value	<code>true</code>	

```
// dotnet_style_prefer_is_null_check_over_reference_equality_method = true
if (value is null)
    return;

// dotnet_style_prefer_is_null_check_over_reference_equality_method = false
if (object.ReferenceEquals(value, null))
    return;
```

```
' dotnet_style_prefer_is_null_check_over_reference_equality_method = true
If value Is Nothing
    Return
End If

' dotnet_style_prefer_is_null_check_over_reference_equality_method = false
If Object.ReferenceEquals(value, Nothing)
    Return
End If
```

SUPPRESS A WARNING

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0041
// The code that's violating the rule is on this line.
#pragma warning restore IDE0041
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.cs,vb]
dotnet_diagnostic.IDE0041.severity = none
```

To disable all of the code-style rules, set the severity for the category `Style` to `none` in the [configuration file](#).

```
[*.cs,vb]
dotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

SEE ALSO

- [Null-checking preferences](#)
- [Code style language rules](#)
- [Code style rules reference](#)

Prefer 'null' check over type check (IDE0150)

9/20/2022 • 2 minutes to read • [Edit Online](#)

PROPERTY	VALUE
Rule ID	IDE0150
Title	Prefer <code>null</code> check over type check
Category	Style
Subcategory	Language rules (null-checking preferences)
Applicable languages	C#
Options	<code>csharp_style_prefer_null_check_over_type_check</code>

Overview

This style rule flags use of the `is {type}` statement when `is not null` can be used instead. Similarly, it flags use of the `is not {type}` statement in favor of `is null`. Using `is null` or `is not null` improves code readability.

Options

Options specify the behavior that you want the rule to enforce. For information about configuring options, see [Option format](#).

`csharp_style_prefer_null_check_over_type_check`

PROPERTY	VALUE	DESCRIPTION
Option name	<code>csharp_style_prefer_null_check_over_type_check</code>	
Option values	<code>true</code>	Prefer null check over type check.
	<code>false</code>	Disables the rule.
Default option value	<code>true</code>	

Example

```
// Violates IDE0150.  
if (numbers is not IEnumerable<int>) ...  
  
// Fixed code.  
if (numbers is null) ...
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0150
// The code that's violating the rule is on this line.
#pragma warning restore IDE0150
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.cs,vb]
dotnet_diagnostic.IDE0150.severity = none
```

To disable all of the code-style rules, set the severity for the category `Style` to `none` in the [configuration file](#).

```
[*.cs,vb]
dotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Null-checking preferences](#)
- [Code style language rules](#)
- [Code style rules reference](#)

Use conditional delegate call (IDE1005)

9/20/2022 • 2 minutes to read • [Edit Online](#)

PROPERTY	VALUE
Rule ID	IDE1005
Title	Use conditional delegate call
Category	Style
Subcategory	Language rules (null-checking preferences)
Applicable languages	C# 6.0+
Options	<code>csharp_style_conditional_delegate_call</code>

Overview

This style rule concerns the use of the [null-conditional operator](#) (`?.`) when invoking a lambda expression, as opposed to performing a null check.

Options

Options specify the behavior that you want the rule to enforce. For information about configuring options, see [Option format](#).

`csharp_style_conditional_delegate_call`

PROPERTY	VALUE	DESCRIPTION
Option name	<code>csharp_style_conditional_delegate_call</code>	
Option values	<code>true</code>	Prefer to use the conditional coalescing operator (<code>?.</code>) when invoking a lambda expression
	<code>false</code>	Prefer to perform a null check before invoking a lambda expression
Default option value	<code>true</code>	

```
// csharp_style_conditional_delegate_call = true
func?.Invoke(args);

// csharp_style_conditional_delegate_call = false
if (func != null) { func(args); }
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE1005
// The code that's violating the rule is on this line.
#pragma warning restore IDE1005
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.IDE1005.severity = none
```

To disable all of the code-style rules, set the severity for the category `Style` to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [null-conditional operator](#)
- [Null-checking preferences](#)
- [Code style language rules](#)
- [Code style rules reference](#)

'var' preferences (IDE0007 and IDE0008)

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article describes two related rules, [IDE0007](#) and [IDE0008](#).

PROPERTY	VALUE
Rule ID	IDE0007
Title	Use <code>var</code> instead of explicit type
Category	Style
Subcategory	Language rules
Applicable languages	C#
Options	<code>csharp_style_var_for_builtin_types</code> <code>csharp_style_var_when_type_is_apparent</code> <code>csharp_style_var_elsewhere</code>
PROPERTY	VALUE
Rule ID	IDE0008
Title	Use explicit type instead of <code>var</code>
Category	Style
Subcategory	Language rules
Applicable languages	C#
Options	<code>csharp_style_var_for_builtin_types</code> <code>csharp_style_var_when_type_is_apparent</code> <code>csharp_style_var_elsewhere</code>

Overview

These two style rules define whether the `var` keyword or an explicit type should be used in a variable declaration. To enforce that `var` is used, set the severity of [IDE0007](#) to warning or error. To enforce that the explicit type is used, set the severity of [IDE0008](#) to warning or error.

Options

This rule's associated options define where this style preference should be applied:

- Built-in types ([csharp_style_var_for_builtin_types](#))
- Places where the type is apparent ([csharp_style_var_when_type_is_apparent](#))
- Elsewhere ([csharp_style_var_elsewhere](#))

For more information about configuring options, see [Option format](#).

csharp_style_var_for_builtin_types

PROPERTY	VALUE	DESCRIPTION
Option name	csharp_style_var_for_builtin_types	
Option values	<code>true</code>	Prefer <code>var</code> is used to declare variables with built-in system types such as <code>int</code>
	<code>false</code>	Prefer explicit type over <code>var</code> to declare variables with built-in system types such as <code>int</code>
Default option value	<code>false</code>	

```
// csharp_style_var_for_builtin_types = true
var x = 5;

// csharp_style_var_for_builtin_types = false
int x = 5;
```

csharp_style_var_when_type_is_apparent

PROPERTY	VALUE	DESCRIPTION
Option name	csharp_style_var_when_type_is_apparent	
Option values	<code>true</code>	Prefer <code>var</code> when the type is already mentioned on the right-hand side of a declaration expression
	<code>false</code>	Prefer explicit type when the type is already mentioned on the right-hand side of a declaration expression
Default option value	<code>false</code>	

```
// csharp_style_var_when_type_is_apparent = true
var obj = new Customer();

// csharp_style_var_when_type_is_apparent = false
Customer obj = new Customer();
```

csharp_style_var_elsewhere

PROPERTY	VALUE	DESCRIPTION
Option name	csharp_style_var_elsewhere	
Option values	<code>true</code>	Prefer <code>var</code> over explicit type in all cases, unless overridden by another code style rule
	<code>false</code>	Prefer explicit type over <code>var</code> in all cases, unless overridden by another code style rule
Default option value	<code>false</code>	

```
// csharp_style_var_elsewhere = true
var f = this.Init();

// csharp_style_var_elsewhere = false
bool f = this.Init();
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0007 // Or IDE0008
// The code that's violating the rule is on this line.
#pragma warning restore IDE0007 // Or IDE0008
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.IDE0007.severity = none
dotnet_diagnostic.IDE0008.severity = none
```

To disable all of the code-style rules, set the severity for the category `style` to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [var keyword](#)
- [Code style language rules](#)
- [Code style rules reference](#)

Expression-bodied-members

9/20/2022 • 2 minutes to read • [Edit Online](#)

Overview

The style rules in this section concern the use of [expression-bodied members](#) when the logic consists of a single expression.

- [Use expression body for constructors \(IDE0021\)](#)
- [Use expression body for methods \(IDE0022\)](#)
- [Use expression body for operators \(IDE0023 and IDE0024\)](#)
- [Use expression body for properties \(IDE0025\)](#)
- [Use expression body for indexers \(IDE0026\)](#)
- [Use expression body for accessors \(IDE0027\)](#)
- [Use expression body for lambda expressions \(IDE0053\)](#)
- [Use expression body for local functions \(IDE0061\)](#)

See also

- [Code style rules reference](#)
- [Code style language rules](#)

Use expression body for constructors (IDE0021)

9/20/2022 • 2 minutes to read • [Edit Online](#)

PROPERTY	VALUE
Rule ID	IDE0021
Title	Use expression body for constructors
Category	Style
Subcategory	Language rules (expression-bodied members)
Applicable languages	C# 7.0+
Options	<code>csharp_style_expression_bodied_constructors</code>

Overview

This style rule concerns the use of [expression bodies](#) versus block bodies for constructors.

Options

Set the value of the associated option for this rule to specify whether expression bodies or block bodies for constructors are preferred, and if expression bodies are preferred, whether they're preferred only for single-line expressions.

For more information about configuring options, see [Option format](#).

`csharp_style_expression_bodied_constructors`

PROPERTY	VALUE	DESCRIPTION
Option name	<code>csharp_style_expression_bodied_constructors</code>	
Option values	<code>true</code>	Prefer expression bodies for constructors
	<code>when_on_single_line</code>	Prefer expression bodies for constructors when they will be a single line
	<code>false</code>	Prefer block bodies for constructors
Default option value	<code>false</code>	

```
// csharp_style_expression_bodied_constructors = true
public Customer(int age) => Age = age;

// csharp_style_expression_bodied_constructors = false
public Customer(int age) { Age = age; }
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0021
// The code that's violating the rule is on this line.
#pragma warning restore IDE0021
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.IDE0021.severity = none
```

To disable all of the code-style rules, set the severity for the category `Style` to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Expression-bodied members](#)
- [Code style rules for expression-bodied members](#)
- [Code style language rules](#)
- [Code style rules reference](#)

Use expression body for methods (IDE0022)

9/20/2022 • 2 minutes to read • [Edit Online](#)

PROPERTY	VALUE
Rule ID	IDE0022
Title	Use expression body for methods
Category	Style
Subcategory	Language rules (expression-bodied members)
Applicable languages	C# 6.0+
Options	<code>csharp_style_expression_bodied_methods</code>

Overview

This style rule concerns the use of [expression bodies](#) versus block bodies for methods.

Options

Set the value of the associated option for this rule to specify whether expression bodies or block bodies for methods are preferred, and if expression bodies are preferred, whether they're preferred only for single-line expressions.

For more information about configuring options, see [Option format](#).

`csharp_style_expression_bodied_methods`

PROPERTY	VALUE	DESCRIPTION
Option name	<code>csharp_style_expression_bodied_methods</code>	
Option values	<code>true</code>	Prefer expression bodies for methods
	<code>when_on_single_line</code>	Prefer expression bodies for methods when they will be a single line
	<code>false</code>	Prefer block bodies for methods
Default option value	<code>false</code>	

```
// csharp_style_expression_bodied_methods = true
public int GetAge() => this.Age;

// csharp_style_expression_bodied_methods = false
public int GetAge() { return this.Age; }
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0022
// The code that's violating the rule is on this line.
#pragma warning restore IDE0022
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.IDE0022.severity = none
```

To disable all of the code-style rules, set the severity for the category `Style` to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Expression-bodied members](#)
- [Code style rules for expression-bodied members](#)
- [Code style language rules](#)
- [Code style rules reference](#)

Use expression body for operators (IDE0023 and IDE0024)

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article describes two related rules, [IDE0023](#) and [IDE0024](#), which apply to *conversion operators* and *operators*, respectively.

PROPERTY	VALUE
Rule ID	IDE0023
Title	Use expression body for conversion operators
Category	Style
Subcategory	Language rules (expression-bodied members)
Applicable languages	C# 7.0+
Options	<code>csharp_style_expression_bodied_operators</code>

PROPERTY	VALUE
Rule ID	IDE0024
Title	Use expression body for operators
Category	Style
Subcategory	Language rules (expression-bodied members)
Applicable languages	C# 7.0+
Options	<code>csharp_style_expression_bodied_operators</code>

Overview

This style rule concerns the use of [expression bodies](#) versus block bodies for operators.

Options

Set the value of the associated option for these rules to specify whether expression bodies or block bodies for operators are preferred, and if expression bodies are preferred, whether they're preferred only for single-line expressions.

For more information about configuring options, see [Option format](#).

`csharp_style_expression_bodied_operators`

PROPERTY	VALUE	DESCRIPTION
Option name	csharp_style_expression_bodied_operators	
Option values	<code>true</code>	Prefer expression bodies for operators
	<code>when_on_single_line</code>	Prefer expression bodies for operators when they will be a single line
	<code>false</code>	Prefer block bodies for operators
Default option value	<code>false</code>	

```
// csharp_style_expression_bodied_operators = true
public static ComplexNumber operator + (ComplexNumber c1, ComplexNumber c2)
    => new ComplexNumber(c1.Real + c2.Real, c1.Imaginary + c2.Imaginary);

// csharp_style_expression_bodied_operators = false
public static ComplexNumber operator + (ComplexNumber c1, ComplexNumber c2)
{ return new ComplexNumber(c1.Real + c2.Real, c1.Imaginary + c2.Imaginary); }
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0023 // Or IDE0024
// The code that's violating the rule is on this line.
#pragma warning restore IDE0023 // Or IDE0024
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.IDE0023.severity = none
dotnet_diagnostic.IDE0024.severity = none
```

To disable all of the code-style rules, set the severity for the category `Style` to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Expression-bodied members](#)
- [Code style rules for expression-bodied members](#)
- [Code style language rules](#)
- [Code style rules reference](#)

Use expression body for properties (IDE0025)

9/20/2022 • 2 minutes to read • [Edit Online](#)

PROPERTY	VALUE
Rule ID	IDE0025
Title	Use expression body for properties
Category	Style
Subcategory	Language rules (expression-bodied members)
Applicable languages	C# 7.0+
Options	<code>csharp_style_expression_bodied_properties</code>

Overview

This style rule concerns the use of [expression bodies](#) versus block bodies for properties.

Options

Set the value of the associated option for this rule to specify whether expression bodies or block bodies for properties are preferred, and if expression bodies are preferred, whether they're preferred only for single-line expressions.

For more information about configuring options, see [Option format](#).

`csharp_style_expression_bodied_properties`

PROPERTY	VALUE	DESCRIPTION
Option name	<code>csharp_style_expression_bodied_properties</code>	
Option values	<code>true</code>	Prefer expression bodies for properties
	<code>when_on_single_line</code>	Prefer expression bodies for properties when they will be a single line
	<code>false</code>	Prefer block bodies for properties
Default option value	<code>true</code>	

```
// csharp_style_expression_bodied_properties = true
public int Age => _age;

// csharp_style_expression_bodied_properties = false
public int Age { get { return _age; } }
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0025
// The code that's violating the rule is on this line.
#pragma warning restore IDE0025
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.IDE0025.severity = none
```

To disable all of the code-style rules, set the severity for the category `Style` to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Expression-bodied members](#)
- [Code style rules for expression-bodied members](#)
- [Code style language rules](#)
- [Code style rules reference](#)

Use expression body for indexers (IDE0026)

9/20/2022 • 2 minutes to read • [Edit Online](#)

PROPERTY	VALUE
Rule ID	IDE0026
Title	Use expression body for indexers
Category	Style
Subcategory	Language rules (expression-bodied members)
Applicable languages	C# 7.0+
Options	<code>csharp_style_expression_bodied_indexers</code>

Overview

This style rule concerns the use of [expression bodies](#) versus block bodies for indexers.

Options

Set the value of the associated option for this rule to specify whether expression bodies or block bodies for indexers are preferred, and if expression bodies are preferred, whether they're preferred only for single-line expressions.

For more information about configuring options, see [Option format](#).

`csharp_style_expression_bodied_indexers`

PROPERTY	VALUE	DESCRIPTION
Option name	<code>csharp_style_expression_bodied_indexers</code>	
Option values	<code>true</code>	Prefer expression bodies for indexers
	<code>when_on_single_line</code>	Prefer expression bodies for indexers when they will be a single line
	<code>false</code>	Prefer block bodies for indexers
Default option value	<code>true</code>	

```
// csharp_style_expression_bodied_indexers = true
public T this[int i] => _values[i];

// csharp_style_expression_bodied_indexers = false
public T this[int i] { get { return _values[i]; } }
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0026
// The code that's violating the rule is on this line.
#pragma warning restore IDE0026
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.IDE0026.severity = none
```

To disable all of the code-style rules, set the severity for the category `Style` to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Expression-bodied members](#)
- [Code style rules for expression-bodied members](#)
- [Code style language rules](#)
- [Code style rules reference](#)

Use expression body for accessors (IDE0027)

9/20/2022 • 2 minutes to read • [Edit Online](#)

PROPERTY	VALUE
Rule ID	IDE0027
Title	Use expression body for accessors
Category	Style
Subcategory	Language rules (expression-bodied members)
Applicable languages	C# 7.0+
Options	<code>csharp_style_expression_bodied_accessors</code>

Overview

This style rule concerns the use of [expression bodies](#) versus block bodies for accessors.

Options

Set the value of the associated option for this rule to specify whether expression bodies or block bodies for accessors are preferred, and if expression bodies are preferred, whether they're preferred only for single-line expressions.

For more information about configuring options, see [Option format](#).

`csharp_style_expression_bodied_accessors`

PROPERTY	VALUE	DESCRIPTION
Option name	<code>csharp_style_expression_bodied_accessors</code>	
Option values	<code>true</code>	Prefer expression bodies for accessors
	<code>when_on_single_line</code>	Prefer expression bodies for accessors when they will be a single line
	<code>false</code>	Prefer block bodies for accessors
Default option value	<code>true</code>	

```
// csharp_style_expression_bodied_accessors = true
public int Age { get => _age; set => _age = value; }

// csharp_style_expression_bodied_accessors = false
public int Age { get { return _age; } set { _age = value; } }
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0027
// The code that's violating the rule is on this line.
#pragma warning restore IDE0027
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.IDE0027.severity = none
```

To disable all of the code-style rules, set the severity for the category `Style` to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Expression-bodied members](#)
- [Code style rules for expression-bodied members](#)
- [Code style language rules](#)
- [Code style rules reference](#)

Use expression body for lambdas (IDE0053)

9/20/2022 • 2 minutes to read • [Edit Online](#)

PROPERTY	VALUE
Rule ID	IDE0053
Title	Use expression body for lambdas
Category	Style
Subcategory	Language rules (expression-bodied members)
Applicable languages	C# 7.0+
Options	<code>csharp_style_expression_bodied_lambdas</code>

Overview

This style rule concerns the use of [expression bodies](#) versus block bodies for [lambda expressions](#).

Options

Options specify the behavior that you want the rule to enforce. For information about configuring options, see [Option format](#).

`csharp_style_expression_bodied_lambdas`

PROPERTY	VALUE	DESCRIPTION
Option name	<code>csharp_style_expression_bodied_lambdas</code>	
Option values	<code>true</code>	Prefer expression bodies for lambdas
	<code>when_on_single_line</code>	Prefer expression bodies for lambdas when they'll be a single line
	<code>false</code>	Prefer block bodies for lambdas
Default option value	<code>true</code>	

```
// csharp_style_expression_bodied_lambdas = true
Func<int, int> square = x => x * x;

// csharp_style_expression_bodied_lambdas = false
Func<int, int> square = x => { return x * x; };
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0053
// The code that's violating the rule is on this line.
#pragma warning restore IDE0053
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.IDE0053.severity = none
```

To disable all of the code-style rules, set the severity for the category `Style` to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Expression-bodied members](#)
- [Code style rules for expression-bodied members](#)
- [Code style language rules](#)
- [Code style rules reference](#)

Use expression body for local functions (IDE0061)

9/20/2022 • 2 minutes to read • [Edit Online](#)

PROPERTY	VALUE
Rule ID	IDE0061
Title	Use expression body for local functions
Category	Style
Subcategory	Language rules (expression-bodied members)
Applicable languages	C# 7.0+
Options	<code>csharp_style_expression_bodied_local_functions</code>

Overview

This style rule concerns the use of [expression bodies](#) versus block bodies for [local functions](#). Local functions are private methods of a type that are nested in another member.

Options

Options specify the behavior that you want the rule to enforce. For information about configuring options, see [Option format](#).

`csharp_style_expression_bodied_local_functions`

PROPERTY	VALUE	DESCRIPTION
Option name	<code>csharp_style_expression_bodied_local_functions</code>	
Option values	<code>true</code>	Prefer expression bodies for local functions
	<code>when_on_single_line</code>	Prefer expression bodies for local functions when they'll be a single line
	<code>false</code>	Prefer block bodies for local functions
Default option value	<code>false</code>	

```
// csharp_style_expression_bodied_local_functions = true
void M()
{
    Hello();
    void Hello() => Console.WriteLine("Hello");
}

// csharp_style_expression_bodied_local_functions = false
void M()
{
    Hello();
    void Hello()
    {
        Console.WriteLine("Hello");
    }
}
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0061
// The code that's violating the rule is on this line.
#pragma warning restore IDE0061
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.IDE0061.severity = none
```

To disable all of the code-style rules, set the severity for the category `Style` to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Expression-bodied members](#)
- [Code style rules for expression-bodied members](#)
- [Code style language rules](#)
- [Code style rules reference](#)

Pattern matching preferences

9/20/2022 • 2 minutes to read • [Edit Online](#)

C# preferences

The style rules in this section concern the use of [pattern matching](#) in C#.

- Use pattern matching to avoid `as` followed by a `null` check (IDE0019)
- Use pattern matching to avoid `is` check followed by a cast (IDE0020 and IDE0038)
- Use switch expression (IDE0066)
- Use pattern matching (IDE0078)
- Use pattern matching (`not` operator) (IDE0083)
- Simplify property pattern (IDE0170)

Visual Basic preferences

The style rules in this section concern the use of pattern matching in Visual Basic.

- Use pattern matching (`IsNot` operator) (IDE0084)

See also

- [Code style rules reference](#)
- [Code style language rules](#)

Use pattern matching to avoid 'as' followed by a 'null' check (IDE0019)

9/20/2022 • 2 minutes to read • [Edit Online](#)

PROPERTY	VALUE
Rule ID	IDE0019
Title	Use pattern matching to avoid <code>as</code> followed by a <code>null</code> check
Category	Style
Subcategory	Language rules (pattern matching preferences)
Applicable languages	C# 7.0+
Options	<code>csharp_style_pattern_matching_over_as_with_null_check</code>

Overview

This style rule concerns the use of C# [pattern matching](#) over an `as` expression followed by a `null` check.

Options

The associated option for this rule specifies whether to prefer pattern match or an `as` expression with null checks to determine if something is of a particular type.

For more information about configuring options, see [Option format](#).

`csharp_style_pattern_matching_over_as_with_null_check`

PROPERTY	VALUE	DESCRIPTION
Option name	<code>csharp_style_pattern_matching_over_a_s_with_null_check</code>	
Option values	<code>true</code>	Prefer pattern matching to determine if something is of a particular type
	<code>false</code>	Prefer <code>as</code> expressions with null checks to determine if something is of a particular type
Default option value	<code>true</code>	

```
// csharp_style_pattern_matching_over_as_with_null_check = true
if (o is string s) {...}

// csharp_style_pattern_matching_over_as_with_null_check = false
var s = o as string;
if (s != null) {...}
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0019
// The code that's violating the rule is on this line.
#pragma warning restore IDE0019
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.cs,vb]
dotnet_diagnostic.IDE0019.severity = none
```

To disable all of the code-style rules, set the severity for the category `Style` to `none` in the [configuration file](#).

```
[*.cs,vb]
dotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Pattern matching in C#](#)
- [Pattern matching preferences](#)
- [Code style language rules](#)
- [Code style rules reference](#)

Use pattern matching to avoid 'is' check followed by a cast (IDE0020 and IDE0038)

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article describes two related rules, [IDE0020](#) and [IDE0038](#).

PROPERTY	VALUE
Rule ID	IDE0020
Title	Use pattern matching to avoid <code>is</code> check followed by a cast (with variable)
Category	Style
Subcategory	Language rules (pattern matching preferences)
Applicable languages	C# 7.0+
Options	<code>csharp_style_pattern_matching_over_is_with_cast_check</code>

PROPERTY	VALUE
Rule ID	IDE0038
Title	Use pattern matching to avoid <code>is</code> check followed by a cast (without variable)
Category	Style
Subcategory	Language rules (pattern matching preferences)
Applicable languages	C# 7.0+
Options	<code>csharp_style_pattern_matching_over_is_with_cast_check</code>

Overview

This style rule concerns the use of C# [pattern matching](#), for example, `o is int i`, over an `is` check followed by a cast, for example, `if (o is int) { ... (int)o ... }`. Enable either [IDE0020](#) or [IDE0038](#) based on whether or not the cast expression should be saved into a separate local variable:

- [IDE0020](#) : Cast expression *is* saved into a local variable. For example, `if (o is int) { var i = (int)o; }` saves the result of `(int)o` in a local variable.
- [IDE0038](#) : Cast expression *is not* saved into a local variable. For example, `if (o is int) { if ((int)o == 1) { ... } }` does not save the result of `(int)o` into a local variable.

Options

Set the value of the associated option for this rule to specify whether pattern matching or `is` check followed by a type cast is preferred.

For more information about configuring options, see [Option format](#).

csharp_style_pattern_matching_over_is_with_cast_check

PROPERTY	VALUE	DESCRIPTION
Option name	<code>csharp_style_pattern_matching_over_is_with_cast_check</code>	
Option values	<code>true</code>	Prefer pattern matching instead of <code>is</code> expressions with type casts
	<code>false</code>	Prefer <code>is</code> expressions with type casts instead of pattern matching
Default option value	<code>true</code>	

```
// csharp_style_pattern_matching_over_is_with_cast_check = true
if (o is int i) {...}

// csharp_style_pattern_matching_over_is_with_cast_check = false
if (o is int) {var i = (int)o; ... }
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0020 // Or IDE0038
// The code that's violating the rule is on this line.
#pragma warning restore IDE0020 // Or IDE0038
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.IDE0020.severity = none
dotnet_diagnostic.IDE0038.severity = none
```

To disable all of the code-style rules, set the severity for the category `style` to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Pattern matching in C#](#)
- [Pattern matching preferences](#)

- [Code style language rules](#)
- [Code style rules reference](#)

Use switch expression (IDE0066)

9/20/2022 • 2 minutes to read • [Edit Online](#)

PROPERTY	VALUE
Rule ID	IDE0066
Title	Use switch expression
Category	Style
Subcategory	Language rules (pattern matching preferences)
Applicable languages	C# 8.0+
Introduced version	Visual Studio 2019 version 16.2
Options	<code>csharp_style_prefer_switch_expression</code>

Overview

This style rule concerns the use of [switch expressions](#), which were introduced in C# 8.0, versus [switch statements](#).

Options

Options specify the behavior that you want the rule to enforce. For information about configuring options, see [Option format](#).

`csharp_style_prefer_switch_expression`

PROPERTY	VALUE	DESCRIPTION
Option name	<code>csharp_style_prefer_switch_expression</code>	
Option values	<code>true</code>	Prefer to use a <code>switch</code> expression
	<code>false</code>	Prefer to use a <code>switch</code> statement
Default option value	<code>true</code>	

```
// csharp_style_prefer_switch_expression = true
return x switch
{
    1 => 1 * 1,
    2 => 2 * 2,
    _ => 0,
};

// csharp_style_prefer_switch_expression = false
switch (x)
{
    case 1:
        return 1 * 1;
    case 2:
        return 2 * 2;
    default:
        return 0;
}
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0066
// The code that's violating the rule is on this line.
#pragma warning restore IDE0066
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.IDE0066.severity = none
```

To disable all of the code-style rules, set the severity for the category `Style` to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [switch expression](#)
- [Pattern matching preferences](#)
- [Code style language rules](#)
- [Code style rules reference](#)

Use pattern matching (IDE0078)

9/20/2022 • 2 minutes to read • [Edit Online](#)

PROPERTY	VALUE
Rule ID	IDE0078
Title	Use pattern matching
Category	Style
Subcategory	Language rules (pattern matching preferences)
Applicable languages	C# 9.0+
Options	<code>csharp_style_prefer_pattern_matching</code>

Overview

This style rule concerns the use of C# 9.0 [pattern matching](#) constructs.

Options

Options specify the behavior that you want the rule to enforce. For information about configuring options, see [Option format](#).

`csharp_style_prefer_pattern_matching`

PROPERTY	VALUE	DESCRIPTION
Option name	<code>csharp_style_prefer_pattern_matching</code>	
Option values	<code>true</code>	Prefer to use pattern matching constructs, when possible
	<code>false</code>	Prefer not to use pattern matching constructs.
Default option value	<code>true</code>	

```
// csharp_style_prefer_pattern_matching = true
var x = i is default(int) or > (default(int));
var y = o is not C c;

// csharp_style_prefer_pattern_matching = false
var x = i == default || i > default(int);
var y = !(o is C c);
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0078
// The code that's violating the rule is on this line.
#pragma warning restore IDE0078
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.IDE0078.severity = none
```

To disable all of the code-style rules, set the severity for the category `Style` to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [C# 9.0 pattern matching](#)
- [Pattern matching preferences](#)
- [Code style language rules](#)
- [Code style rules reference](#)

Use pattern matching (`not` operator) (IDE0083)

9/20/2022 • 2 minutes to read • [Edit Online](#)

PROPERTY	VALUE
Rule ID	IDE0083
Title	Use pattern matching (<code>not</code> operator)
Category	Style
Subcategory	Language rules (pattern matching preferences)
Applicable languages	C# 9.0+
Options	<code>csharp_style_prefer_not_pattern</code>

Overview

This style rule concerns the use of C# 9.0 `not` pattern, when possible.

Options

Options specify the behavior that you want the rule to enforce. For information about configuring options, see [Option format](#).

`csharp_style_prefer_not_pattern`

PROPERTY	VALUE	DESCRIPTION
Option name	<code>csharp_style_prefer_not_pattern</code>	
Option values	<code>true</code>	Prefer to use the <code>not</code> pattern, when possible
	<code>false</code>	Prefer <i>not</i> to use the <code>not</code> pattern.
Default option value	<code>true</code>	

NOTE

When the option is set to `false`, the analyzer *does not* flag uses of the `not` pattern. However, any code that's generated won't use the `not` pattern. When the option is set to `true`, code that doesn't use the `not` pattern is flagged, and any code that's generated uses the `not` pattern where applicable.

The following examples show how code would be generated by code-generating features when the option is set to either `true` or `false`.

```
// csharp_style_prefer_not_pattern = true  
var y = o is not C c;  
  
// csharp_style_prefer_not_pattern = false  
var y = !(o is C c);
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0083  
// The code that's violating the rule is on this line.  
#pragma warning restore IDE0083
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]  
dotnet_diagnostic.IDE0083.severity = none
```

To disable all of the code-style rules, set the severity for the category `Style` to `none` in the [configuration file](#).

```
[*.{cs,vb}]  
dotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [IDE0078: Use pattern matching](#)
- [IDE0084: Use pattern matching \(Visual Basic IsNot operator\)](#)
- [C# 9.0 pattern matching](#)
- [Pattern matching preferences](#)
- [Code style language rules](#)
- [Code style rules reference](#)

Use pattern matching (`IsNot` operator) (IDE0084)

9/20/2022 • 2 minutes to read • [Edit Online](#)

PROPERTY	VALUE
Rule ID	IDE0084
Title	Use pattern matching (<code> IsNot</code> operator)
Category	Style
Subcategory	Language rules (pattern matching preferences)
Applicable languages	Visual Basic 14.0+
Options	<code>visual_basic_style_prefer_isnot_expression</code>

Overview

This style rule concerns the use of the Visual Basic 14.0 `IsNot` pattern, when possible.

Options

Options specify the behavior that you want the rule to enforce. For information about configuring options, see [Option format](#).

`visual_basic_style_prefer_isnot_expression`

PROPERTY	VALUE	DESCRIPTION
Option name	<code>visual_basic_style_prefer_isnot_expressi on</code>	
Option values	<code>true</code>	Prefer to use the <code> IsNot</code> pattern, when possible
	<code>false</code>	Prefer <i>not</i> to use the <code> IsNot</code> pattern.
Default option value	<code>true</code>	

```
' visual_basic_style_prefer_isnot_expression = true
Dim y = o IsNot C

' visual_basic_style_prefer_isnot_expression = false
Dim y = Not o Is C
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and

then re-enable the rule.

```
#pragma warning disable IDE0084
// The code that's violating the rule is on this line.
#pragma warning restore IDE0084
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.IDE0084.severity = none
```

To disable all of the code-style rules, set the severity for the category `Style` to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [IDE0078: Use pattern matching](#)
- [IDE0083: Use pattern matching \(C# not operator\)](#)
- [C# 9.0 pattern matching](#)
- [Pattern matching preferences](#)
- [Code style language rules](#)
- [Code style rules reference](#)

Simplify property pattern (IDE0170)

9/20/2022 • 2 minutes to read • [Edit Online](#)

PROPERTY	VALUE
Rule ID	IDE0170
Title	Simplify property pattern
Category	Style
Subcategory	Language rules (pattern matching preferences)
Applicable languages	C#
Options	<code>csharp_style_prefer_extended_property_pattern</code>

Overview

This style rule flags the use of a nested pattern in a [property pattern](#). A nested pattern can be simplified to use an extended property pattern in which property subpatterns are used to reference nested members. Extended property patterns improve code readability.

Options

Options specify the behavior that you want the rule to enforce. For information about configuring options, see [Option format](#).

`csharp_style_prefer_extended_property_pattern`

PROPERTY	VALUE	DESCRIPTION
Option name	<code>csharp_style_prefer_extended_property_pattern</code>	
Option values	<code>true</code>	Prefer the extended property pattern.
	<code>false</code>	Disables the rule.
Default option value	<code>true</code>	

Example

```
public record Point(int X, int Y);
public record Segment(Point Start, Point End);

// Violates IDE0170.
static bool IsEndOnXAxis(Segment segment) =>
    segment is { Start: { Y: 0 } } or { End: { Y: 0 } };

// Fixed code.
static bool IsEndOnXAxis(Segment segment) =>
    segment is { Start.Y: 0 } or { End.Y: 0 };
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0170
// The code that's violating the rule is on this line.
#pragma warning restore IDE0170
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.IDE0170.severity = none
```

To disable all of the code-style rules, set the severity for the category `Style` to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Property pattern](#)
- [Pattern matching preferences](#)
- [Code style language rules](#)
- [Code style rules reference](#)

Code block preferences

9/20/2022 • 2 minutes to read • [Edit Online](#)

Overview

The style rules in this section concern the following code block preferences:

- [Add braces \(IDE0011\)](#)
- [Use simple 'using' statement \(IDE0063\)](#)

See also

- [Code style rules reference](#)
- [Code style language rules](#)

Add braces (IDE0011)

9/20/2022 • 2 minutes to read • [Edit Online](#)

PROPERTY	VALUE
Rule ID	IDE0011
Title	Add braces
Category	Style
Subcategory	Language rules (code block preferences)
Options	<code>csharp_prefer_braces</code>

Overview

This style rule concerns the use of curly braces `{ }` to surround code blocks.

Options

Use the following option to specify whether curly braces are preferred or not, and if preferred, whether only for multi-line code blocks.

For more information about configuring options, see [Option format](#).

`csharp_prefer_braces`

PROPERTY	VALUE	DESCRIPTION
Option name	<code>csharp_prefer_braces</code>	
Option values	<code>true</code>	Prefer curly braces even for one line of code
	<code>false</code>	Prefer no curly braces if allowed
	<code>when_multiline</code>	Prefer curly braces on multiple lines
Default option value	<code>true</code>	

```
// csharp_prefer_braces = true
if (test) { this.Display(); }

// csharp_prefer_braces = false
if (test) this.Display();
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0011
// The code that's violating the rule is on this line.
#pragma warning restore IDE0011
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.IDE0011.severity = none
```

To disable all of the code-style rules, set the severity for the category `Style` to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Code block preferences](#)
- [Code style language rules](#)
- [Code style rules reference](#)

Use simple 'using' statement (IDE0063)

9/20/2022 • 2 minutes to read • [Edit Online](#)

PROPERTY	VALUE
Rule ID	IDE0063
Title	Use simple <code>using</code> statement
Category	Style
Subcategory	Language rules (code block preferences)
Applicable languages	C# 8.0+
Options	<code>csharp_prefer_simple_using_statement</code>

Overview

This style rule concerns the use of `using` statements without curly braces, also known as `using` declarations. This [alternative syntax](#) was introduced in C# 8.0.

Options

Options specify the behavior that you want the rule to enforce. For information about configuring options, see [Option format](#).

`csharp_prefer_simple_using_statement`

PROPERTY	VALUE	DESCRIPTION
Option name	<code>csharp_prefer_simple_using_statement</code>	
Option values	<code>true</code>	Prefer to use a <code>using</code> declaration
	<code>false</code>	Prefer to use a <code>using</code> statement with curly braces
Default option value	<code>true</code>	

```
// csharp_prefer_simple_using_statement = true
using var a = b;

// csharp_prefer_simple_using_statement = false
using (var a = b) { }
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and

then re-enable the rule.

```
#pragma warning disable IDE0063
// The code that's violating the rule is on this line.
#pragma warning restore IDE0063
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.IDE0063.severity = none
```

To disable all of the code-style rules, set the severity for the category `Style` to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [using statement](#)
- [Code block preferences](#)
- [Code style language rules](#)
- [Code style rules reference](#)

'using' directive placement (IDE0065)

9/20/2022 • 2 minutes to read • [Edit Online](#)

PROPERTY	VALUE
Rule ID	IDE0065
Title	'using' directive placement
Category	Style
Subcategory	Language rules ('using' directive preferences)
Applicable languages	C#
Options	csharp_using_directive_placement

Overview

This style rule concerns the preference of placing `using` directives outside or inside the namespace.

Options

Options specify the behavior that you want the rule to enforce. For information about configuring options, see [Option format](#).

csharp_using_directive_placement

PROPERTY	VALUE	DESCRIPTION
Option name	csharp_using_directive_placement	
Option values	outside_namespace	Prefer <code>using</code> directives to be placed outside the namespace
	inside_namespace	Prefer <code>using</code> directives to be placed inside the namespace
Default option value	outside_namespace	

```
// csharp_using_directive_placement = outside_namespace
using System;

namespace Conventions
{
    ...
}

// csharp_using_directive_placement = inside_namespace
namespace Conventions
{
    using System;
    ...
}
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0065
// The code that's violating the rule is on this line.
#pragma warning restore IDE0065
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.IDE0065.severity = none
```

To disable all of the code-style rules, set the severity for the category `Style` to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Remove unnecessary using directives \(IDE0005\)](#)
- [Code style language rules](#)
- [Code style rules reference](#)

Require file header (IDE0073)

9/20/2022 • 2 minutes to read • [Edit Online](#)

PROPERTY	VALUE
Rule ID	IDE0073
Title	Require file header
Category	Style
Subcategory	Language rules (file header preferences)
Applicable languages	C# and Visual Basic
Options	<code>file_header_template</code>

Overview

This style rule concerns providing a file header at top of source code files.

Options

Specify the required header text by setting the `file_header_template` option.

- When the option value is a non-empty string, require the specified file header.
- When the option value is `unset` or an empty string, do not require a file header.

For information about configuring options, see [Option format](#).

`file_header_template`

PROPERTY	VALUE	DESCRIPTION
Option name	<code>file_header_template</code>	
Option values	non-empty string, optionally containing a <code>{fileName}</code> placeholder	Prefer the string as the required file header.
	<code>unset</code> or empty string	Do not require a file header.
Default option value	<code>unset</code>	

```
// file_header_template = Copyright (c) SomeCorp. All rights reserved.\nLicensed under the xyz license.  
  
// Copyright (c) SomeCorp. All rights reserved.  
// Licensed under the xyz license.  
namespace N1  
{  
    class C1 { }  
}  
  
// file_header_template = unset  
// OR  
// file_header_template =  
namespace N2  
{  
    class C2 { }  
}
```

```
' file_header_template = Copyright (c) SomeCorp. All rights reserved.\nLicensed under the xyz license.  
  
' Copyright (c) SomeCorp. All rights reserved.  
' Licensed under the xyz license.  
Namespace N1  
    Class C1  
    End Class  
End Namespace  
  
' file_header_template = unset  
' OR  
' file_header_template =  
Namespace N2  
    Class C2  
    End Class  
End Namespace
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0073  
// The code that's violating the rule is on this line.  
#pragma warning restore IDE0073
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]  
dotnet_diagnostic.IDE0073.severity = none
```

To disable all of the code-style rules, set the severity for the category `Style` to `none` in the [configuration file](#).

```
[*.{cs,vb}]  
dotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Code style language rules](#)
- [Code style rules reference](#)

Unnecessary code rules

9/20/2022 • 2 minutes to read • [Edit Online](#)

Unnecessary code rules identify different parts of the code base that are unnecessary and can be refactored or removed. The presence of unnecessary code indicates one of more of the following problems:

- Readability: Code that is unnecessarily degrading readability. For example, [IDE0001](#) flags unnecessary type-name qualifications.
- Maintainability: Code that is no longer used after refactoring and is unnecessarily being maintained. For example, [IDE0051](#) flags unused private fields, properties, events, and methods.
- Performance: Unnecessary computation that has no side effects and leads to unnecessary performance overhead. For example, [IDE0059](#) flags unused value assignments where the expression to compute a value has no side effects.
- Functionality: Functional issue in code leading to required code being rendered redundant. For example, [IDE0060](#) flags unused parameters where the method accidentally ignores an input parameter.

The rules in this section concern the following unnecessary code rules:

- [Simplify name \(IDE0001\)](#)
- [Simplify member access \(IDE0002\)](#)
- [Remove unnecessary cast \(IDE0004\)](#)
- [Remove unnecessary import \(IDE0005\)](#)
- [Remove unreachable code \(IDE0035\)](#)
- [Remove unused private member \(IDE0051\)](#)
- [Remove unread private member \(IDE0052\)](#)
- [Remove unused expression value \(IDE0058\)](#)
- [Remove unnecessary value assignment \(IDE0059\)](#)
- [Remove unused parameter \(IDE0060\)](#)
- [Remove unnecessary suppression \(IDE0079\)](#)
- [Remove unnecessary suppression operator \(IDE0080\) - C# only.](#)
- [Remove 'ByVal' \(IDE0081\) - Visual Basic only.](#)
- [Remove unnecessary equality operator \(IDE0100\)](#)
- [Remove unnecessary discard \(IDE0110\) - C# only.](#)
- [Simplify object creation \(IDE0140\) - Visual Basic only.](#)

Some of these rules have options to configure the rule behavior.

See also

- [Code style language rules](#)
- [Code style rules reference](#)

Simplify name (IDE0001)

9/20/2022 • 2 minutes to read • [Edit Online](#)

PROPERTY	VALUE
Rule ID	IDE0001
Title	Simplify name
Category	Style
Subcategory	Unnecessary code rules
Applicable languages	C# and Visual Basic

Overview

This rule concerns the use of simplified type names in declarations and executable code, when possible. You can remove unnecessary name qualification to simplify code and improve readability.

Options

This rule has no associated code-style options.

Example

```
using System.IO;
class C
{
    // IDE0001: 'System.IO.FileInfo' can be simplified to 'FileInfo'
    System.IO.FileInfo file;

    // Fixed code
    FileInfo file;
}
```

```
Imports System.IO
Class C
    ' IDE0001: 'System.IO.FileInfo' can be simplified to 'FileInfo'
    Private file As System.IO.FileInfo

    ' Fixed code
    Private file As FileInfo
End Class
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0001
// The code that's violating the rule is on this line.
#pragma warning restore IDE0001
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.IDE0001.severity = none
```

To disable all of the code-style rules, set the severity for the category `Style` to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Simplify member access \(IDE0002\)](#)
- [Unnecessary code rules](#)
- [Code style rules reference](#)

Simplify member access (IDE0002)

9/20/2022 • 2 minutes to read • [Edit Online](#)

PROPERTY	VALUE
Rule ID	IDE0002
Title	Simplify member access
Category	Style
Subcategory	Unnecessary code rules
Applicable languages	C# and Visual Basic

Overview

This rule concerns use of simplified type member access in declarations and executable code, when possible. Unnecessary qualification can be removed to simplify code and improve readability.

Options

This rule has no associated code-style options.

Example

```
static void M1() { }
static void M2()
{
    // IDE0002: 'C.M1' can be simplified to 'M1'
    C.M1();

    // Fixed code
    M1();
}
```

```
Shared Sub M1()
End Sub

Shared Sub M2()
    ' IDE0002: 'C.M1' can be simplified to 'M1'
    C.M1()

    ' Fixed code
    M1()
End Sub
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0002
// The code that's violating the rule is on this line.
#pragma warning restore IDE0002
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.IDE0002.severity = none
```

To disable all of the code-style rules, set the severity for the category `Style` to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Simplify name \(IDE0001\)](#)
- [Unnecessary code rules](#)
- [Code style rules reference](#)

Remove unnecessary cast (IDE0004)

9/20/2022 • 2 minutes to read • [Edit Online](#)

PROPERTY	VALUE
Rule ID	IDE0004
Title	Remove unnecessary cast
Category	Style
Subcategory	Unnecessary code rules
Applicable languages	C# and Visual Basic

Overview

This rule flags unnecessary [type casts](#). A cast expression is unnecessary if the code semantics would be identical with or without it.

Options

This rule has no associated code-style options.

Example

```
// Code with violations
int v = (int)0;

// Fixed code
int v = 0;
```

```
' Code with violations
Dim v As Integer = CType(0, Integer)

' Fixed code
Dim v As Integer = 0
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0004
// The code that's violating the rule is on this line.
#pragma warning restore IDE0004
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.IDE0004.severity = none
```

To disable all of the code-style rules, set the severity for the category `Style` to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Type cast and conversions](#)
- [Unnecessary code rules](#)
- [Code style rules reference](#)

Remove unnecessary using directives (IDE0005)

9/20/2022 • 2 minutes to read • [Edit Online](#)

PROPERTY	VALUE
Rule ID	IDE0005
Title	Remove unnecessary import
Category	Style
Subcategory	Unnecessary code rules
Applicable languages	C# and Visual Basic

Overview

This rule flags the following unnecessary constructs. If unnecessary, these constructs can be removed without changing the semantics of the code:

- [using directives](#) (C#).
- [Import statements](#) (Visual Basic).

NOTE

To enable this [rule on build](#), you need to enable [XML documentation comments](#) for the project. For more information, see [dotnet/roslyn issue 41640](#).

Options

This rule has no associated code-style options.

Example

```
// Code with violations
using System;
using System.IO;    // IDE0005: Using directive is unnecessary
class C
{
    public static void M()
    {
        Console.WriteLine("Hello");
    }
}

// Fixed code
using System;
class C
{
    public static void M()
    {
        Console.WriteLine("Hello");
    }
}
```

```
' Code with violations
Imports System.IO    ' IDE0005: Imports statement is unnecessary
Class C
    Public Shared Sub M()
        Console.WriteLine("Hello")
    End Sub
End Class

' Fixed code
Class C
    Public Shared Sub M()
        Console.WriteLine("Hello")
    End Sub
End Class
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0005
// The code that's violating the rule is on this line.
#pragma warning restore IDE0005
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.IDE0005.severity = none
```

To disable all of the code-style rules, set the severity for the category `Style` to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- ['using' directive placement \(IDE0065\)](#)
- [C# using directive](#)
- [C# using static directive](#)
- [Visual Basic Import statement](#)
- [Unnecessary code rules](#)
- [Code style rules reference](#)

Remove unreachable code (IDE0035)

9/20/2022 • 2 minutes to read • [Edit Online](#)

PROPERTY	VALUE
Rule ID	IDE0035
Title	Remove unreachable code
Category	Style
Subcategory	Unnecessary code rules
Applicable languages	C# and Visual Basic

Overview

This rule flags executable code within methods and properties that can never be reached, and hence can be removed.

Options

This rule has no associated code-style options.

Example

```
// Code with violations
void M()
{
    throw new System.Exception();

    // IDE0035: Remove unreachable code
    int v = 0;
}

// Fixed code
void M()
{
    throw new System.Exception();
}
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0035
// The code that's violating the rule is on this line.
#pragma warning restore IDE0035
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.IDE0035.severity = none
```

To disable all of the code-style rules, set the severity for the category `Style` to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Remove unused private member \(IDE0051\)](#)
- [Remove unread private member \(IDE0052\)](#)
- [Unnecessary code rules](#)
- [Code style rules reference](#)

Remove unused private member (IDE0051)

9/20/2022 • 2 minutes to read • [Edit Online](#)

PROPERTY	VALUE
Rule ID	IDE0051
Title	Remove unused private member
Category	CodeQuality
Subcategory	Unnecessary code rules
Applicable languages	C# and Visual Basic

Overview

This rule flags unused private methods, fields, properties, and events that have no read or write references.

Options

This rule has no associated code-style options.

Example

```
// Code with violations
class C
{
    // IDE0051: Remove unused private members
    private readonly int _fieldPrivate;
    private int PropertyPrivate => 1;
    private int GetNumPrivate() => 1;

    // No IDE0051
    internal readonly int FieldInternal;
    private readonly int _fieldPrivateUsed;
    public int PropertyPublic => _fieldPrivateUsed;
    private int GetNumPrivateUsed() => 1;
    internal int GetNumInternal() => GetNumPrivateUsed();
    public int GetNumPublic() => GetNumPrivateUsed();
}

// Fixed code
class C
{
    // No IDE0051
    internal readonly int FieldInternal;
    private readonly int _fieldPrivateUsed;
    public int PropertyPublic => _fieldPrivateUsed;
    private int GetNumPrivateUsed() => 1;
    internal int GetNumInternal() => GetNumPrivateUsed();
    public int GetNumPublic() => GetNumPrivateUsed();
}
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0051
// The code that's violating the rule is on this line.
#pragma warning restore IDE0051
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.cs,vb]
dotnet_diagnostic.IDE0051.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.cs,vb]
dotnet_analyzer_diagnostic.category-CodeQuality.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Remove unread private member \(IDE0052\)](#)
- [Remove unreachable code \(IDE0035\)](#)
- [Unnecessary code rules](#)
- [Code style rules reference](#)

Remove unread private member (IDE0052)

9/20/2022 • 2 minutes to read • [Edit Online](#)

PROPERTY	VALUE
Rule ID	IDE0052
Title	Remove unread private member
Category	CodeQuality
Subcategory	Unnecessary code rules
Applicable languages	C# and Visual Basic

Overview

This rule flags private fields and properties that have one or more write references but no read references. In this scenario, some parts of the code can be refactored or removed to fix maintainability, performance, or functional issues.

Options

This rule has no associated code-style options.

Example

```
// Code with violations
class C
{
    // IDE0052: Remove unread private members
    private readonly int _field1;
    private int _field2;
    private int Property { get; set; }

    public C()
    {
        _field1 = 0;
    }

    public void SetMethod()
    {
        _field2 = 0;
        Property = 0;
    }
}

// Fixed code
class C
{
    public C()
    {
    }

    public void SetMethod()
    {
    }
}
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0052
// The code that's violating the rule is on this line.
#pragma warning restore IDE0052
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.IDE0052.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-CodeQuality.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Remove unused private member \(IDE0051\)](#)
- [Remove unreachable code \(IDE0035\)](#)
- [Unnecessary code rules](#)

- [Code style rules reference](#)

Remove unnecessary expression value (IDE0058)

9/20/2022 • 2 minutes to read • [Edit Online](#)

PROPERTY	VALUE
Rule ID	IDE0058
Title	Remove unnecessary expression value
Category	Style
Subcategory	Unnecessary code rules
Applicable languages	C# and Visual Basic
Options	<code>csharp_style_unused_value_expression_statement_preference</code> <code>visual_basic_style_unused_value_expression_statement_preference</code>

Overview

This rule flags unused expression values. For example:

```
void M()
{
    Compute(); // IDE0058: computed value is never used.
}

int Compute();
```

You can take one of the following actions to fix this violation:

- If the expression has no side effects, remove the entire statement. This improves performance by avoiding unnecessary computation.
- If the expression has side effects, replace the left side of the assignment with a **discard** (C# only) or a local variable that's never used. This improves code clarity by explicitly showing the intent to discard an unused value.

```
_ = Compute();
```

Options

The options for this specify whether to prefer the use of a discard or an unused local variable:

- C# - `csharp_style_unused_value_expression_statement_preference`
- Visual Basic - `visual_basic_style_unused_value_expression_statement_preference`

For information about configuring options, see [Option format](#).

`csharp_style_unused_value_expression_statement_preference`

PROPERTY	VALUE	DESCRIPTION
Option name	csharp_style_unused_value_expression_statement_preference	
Applicable languages	C#	
Option values	<input type="checkbox"/> discard_variable <input type="checkbox"/> unused_local_variable	Prefer to assign an unused expression to a discard
	<input type="checkbox"/> unused_local_variable	Prefer to assign an unused expression to a local variable that's never used
Default option value	<input type="checkbox"/> discard_variable	

```
// Original code:  
System.Convert.ToInt32("35");  
  
// After code fix for IDE0058:  
  
// csharp_style_unused_value_expression_statement_preference = discard_variable  
_ = System.Convert.ToInt32("35");  
  
// csharp_style_unused_value_expression_statement_preference = unused_local_variable  
var unused = Convert.ToInt32("35");
```

visual_basic_style_unused_value_expression_statement_preference

PROPERTY	VALUE	DESCRIPTION
Option name	visual_basic_style_unused_value_expression_statement_preference	
Applicable languages	Visual Basic	
Option values	<input type="checkbox"/> unused_local_variable	Prefer to assign an unused expression to a local variable that's never used
Default option value	<input type="checkbox"/> unused_local_variable	

```
' visual_basic_style_unused_value_expression_statement_preference = unused_local_variable  
Dim unused = Computation()
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0058  
// The code that's violating the rule is on this line.  
#pragma warning restore IDE0058
```

To disable the rule for a file, folder, or project, set its severity to none in the [configuration file](#).

```
[*.{cs,vb}]\ndotnet_diagnostic.IDE0058.severity = none
```

To disable all of the code-style rules, set the severity for the category `Style` to `none` in the [configuration file](#).

```
[*.{cs,vb}]\ndotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Remove unnecessary value assignment \(IDE0059\)](#)
- [Unnecessary code rules](#)
- [Code style rules reference](#)

Remove unnecessary value assignment (IDE0059)

9/20/2022 • 2 minutes to read • [Edit Online](#)

PROPERTY	VALUE
Rule ID	IDE0059
Title	Remove unnecessary value assignment
Category	Style
Subcategory	Unnecessary code rules
Applicable languages	C# and Visual Basic
Options	<code>csharp_style_unused_value_assignment_preference</code>
	<code>visual_basic_style_unused_value_assignment_preference</code>

Overview

This rule flags unnecessary value assignments. For example:

```
int v = Compute(); // IDE0059: value written to 'v' is never read, so assignment to 'v' is unnecessary.  
v = Compute2();
```

Users can take one of the following actions to fix this violation:

- If the expression on the right side of the assignment has no side effects, remove the expression or the entire assignment statement. This improves performance by avoiding unnecessary computation.

```
int v = Compute2();
```

- If the expression on the right side of the assignment has side effects, replace the left side of the assignment with a [discard](#) (C# only) or a local variable that's never used. This improves code clarity by explicitly showing the intent to discard an unused value.

```
_ = Compute();  
int v = Compute2();
```

Options

The options for this specify whether to prefer the use of a discard or an unused local variable:

- C# - `csharp_style_unused_value_assignment_preference`
- Visual Basic - `visual_basic_style_unused_value_assignment_preference`

For information about configuring options, see [Option format](#).

csharp_style_unused_value_assignment_preference

PROPERTY	VALUE	DESCRIPTION
Option name	csharp_style_unused_value_assignment_preference	
Applicable languages	C#	
Option values	<code>discard_variable</code>	Prefer to use a discard when assigning a value that's not used
	<code>unused_local_variable</code>	Prefer to use a local variable when assigning a value that's not used
Default option value	<code>discard_variable</code>	

```
// csharp_style_unused_value_assignment_preference = discard_variable
int GetCount(Dictionary<string, int> wordCount, string searchWord)
{
    _ = wordCount.TryGetValue(searchWord, out var count);
    return count;
}

// csharp_style_unused_value_assignment_preference = unused_local_variable
int GetCount(Dictionary<string, int> wordCount, string searchWord)
{
    var unused = wordCount.TryGetValue(searchWord, out var count);
    return count;
}
```

visual_basic_style_unused_value_assignment_preference

PROPERTY	VALUE	DESCRIPTION
Option name	visual_basic_style_unused_value_assignment_preference	
Applicable languages	Visual Basic	
Option values	<code>unused_local_variable</code>	Prefer to use a local variable when assigning a value that's not used
Default option value	<code>unused_local_variable</code>	

```
' visual_basic_style_unused_value_assignment_preference = unused_local_variable
Dim unused = Computation()
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0059
// The code that's violating the rule is on this line.
#pragma warning restore IDE0059
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.IDE0059.severity = none
```

To disable all of the code-style rules, set the severity for the category `Style` to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Remove unused expression value \(IDE0058\)](#)
- [Unnecessary code rules](#)
- [Code style rules reference](#)

Remove unused parameter (IDE0060)

9/20/2022 • 2 minutes to read • [Edit Online](#)

PROPERTY	VALUE
Rule ID	IDE0060
Title	Remove unused parameter
Category	Style
Subcategory	Unnecessary code rules
Applicable languages	C# and Visual Basic
Options	<code>dotnet_code_quality_unused_parameters</code>

Overview

This rule flags unused parameters.

This rule does not flag parameters that are named with the `discard` symbol `_`. In addition, the rule ignores parameters that are named with the discard symbol followed by an integer, for example, `_1`. This behavior reduces warning noise on parameters that are needed for signature requirements, for example, a method used as a delegate, a parameter with special attributes, or a parameter whose value is implicitly accessed at run time by a framework but is not referenced in code.

Options

The option value specifies if unused parameters should be flagged only for non-public methods or for both public and non-public methods.

For information about configuring options, see [Option format](#).

`dotnet_code_quality_unused_parameters`

PROPERTY	VALUE	DESCRIPTION
Option name	<code>dotnet_code_quality_unused_parameters</code>	
Option values	<code>all</code>	Flag methods with any accessibility that contain unused parameters
	<code>non_public</code>	Flag only non-public methods that contain unused parameters
Default option value	<code>all</code>	

```
// dotnet_code_quality_unused_parameters = all
public int GetNum1(int unusedParam) { return 1; }
internal int GetNum2(int unusedParam) { return 1; }
private int GetNum3(int unusedParam) { return 1; }

// dotnet_code_quality_unused_parameters = non_public
internal int GetNum4(int unusedParam) { return 1; }
private int GetNum5(int unusedParam) { return 1; }
```

```
' dotnet_code_quality_unused_parameters = all
Public Function GetNum1(unused As Integer)
    Return 1
End Function

Friend Function GetNum2(unused As Integer)
    Return 1
End Function

Private Function GetNum3(unused As Integer)
    Return 1
End Function

' dotnet_code_quality_unused_parameters = non_public
Friend Function GetNum4(arg1 As Integer)
    Return 1
End Function

Private Function GetNum5(arg1 As Integer)
    Return 1
End Function
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0060
// The code that's violating the rule is on this line.
#pragma warning restore IDE0060
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.IDE0060.severity = none
```

To disable all of the code-style rules, set the severity for the category `Style` to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Unnecessary code rules](#)
- [Code style rules reference](#)

Remove unnecessary suppression (IDE0079)

9/20/2022 • 2 minutes to read • [Edit Online](#)

PROPERTY	VALUE
Rule ID	IDE0079
Title	Remove unnecessary suppression
Category	CodeQuality
Subcategory	Unnecessary code rules
Applicable languages	C# and Visual Basic
Options	<code>dotnet_remove_unnecessary_suppression_exclusions</code>

Overview

This rule flags unnecessary [pragma](#) and [SuppressMessageAttribute](#) attribute suppressions in source. Source suppressions are meant to suppress violations of compiler and analyzer rules for specific parts of source code, without disabling the rules in the other parts of the code. They are generally added to suppress false positives or less important violations that user does not intend to fix. Suppressions can frequently become stale, either due to the rules getting fixed to prevent false positives or user code being refactored to render the suppressions redundant. This rule helps identify such redundant suppressions, which can be removed.

Example

```

using System.Diagnostics.CodeAnalysis;

class C1
{
    // Necessary pragma suppression
#pragma warning disable IDE0051 // IDE0051: Remove unused member
    private int UnusedMethod() => 0;
#pragma warning restore IDE0051

    // IDE0079: Unnecessary pragma suppression
#pragma warning disable IDE0051 // IDE0051: Remove unused member
    private int UsedMethod() => 0;
#pragma warning restore IDE0051

    public int PublicMethod() => UsedMethod();
}

class C2
{
    // Necessary SuppressMessage attribute suppression
    [SuppressMessage("CodeQuality", "IDE0051:Remove unused private members", Justification = "<Pending>")]
    private int _unusedField;

    // IDE0079: Unnecessary SuppressMessage attribute suppression
    [SuppressMessage("CodeQuality", "IDE0051:Remove unused private members", Justification = "<Pending>")]
    private int _usedField;

    public int PublicMethod2() => _usedField;
}

```

Options

Options specify the behavior that you want the rule to enforce. For information about configuring options, see [Option format](#).

`dotnet_remove_unnecessary_suppression_exclusions`

PROPERTY	VALUE	DESCRIPTION
Option name	dotnet_remove_unnecessary_suppression_exclusions	
Option values	<code>,</code> separated list of rule IDs or categories (prefixed with <code>category:</code>) <code>all</code>	Excludes suppressions for the listed rules Disables the rule (all rule IDs excluded)
	<code>none</code>	Enables the rule for all rules (no exclusions)
Default option value	<code>none</code>	

```
using System.Diagnostics.CodeAnalysis;

class C1
{
    // 'dotnet_remove_unnecessary_suppression_exclusions = IDE0051'

    // Unnecessary pragma suppression, but not flagged by IDE0079
#pragma warning disable IDE0051 // IDE0051: Remove unused member
    private int UsedMethod() => 0;
#pragma warning restore IDE0051

    public int PublicMethod() => UsedMethod();
}
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0079
// The code that's violating the rule is on this line.
#pragma warning restore IDE0079
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.IDE0079.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-CodeQuality.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [pragma](#)
- [SuppressMessageAttribute](#)
- [Unnecessary code rules](#)
- [Code style rules reference](#)

Remove unnecessary suppression operator (IDE0080)

9/20/2022 • 2 minutes to read • [Edit Online](#)

PROPERTY	VALUE
Rule ID	IDE0080
Title	Remove unnecessary suppression operator
Category	Style
Subcategory	Unnecessary code rules
Applicable languages	C#

Overview

This rule flags an unnecessary [suppression or null-forgiving operator](#). The operator is unnecessary when it's used in a context where it has no effect. Use of the suppression operator, for example, `x!`, declares that the expression `x` of a reference type isn't null. However, when used in a context of another operator, for example, the [is operator](#) in `o !is string`, it has no effect and can be removed.

Options

This rule has no associated code-style options.

Example

```
// Code with violations
if (o !is string) { }

// Potential fixes:
// 1.
if (o is not string) { }

// 2.
if (!(o is string)) { }

// 3.
if (o is string) { }
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0080
// The code that's violating the rule is on this line.
#pragma warning restore IDE0080
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.IDE0080.severity = none
```

To disable all of the code-style rules, set the severity for the category `Style` to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Suppression or null-forgiving operator](#)
- [Unnecessary code rules](#)
- [Code style rules reference](#)

Remove `ByVal` (IDE0081)

9/20/2022 • 2 minutes to read • [Edit Online](#)

PROPERTY	VALUE
Rule ID	IDE0081
Title	Remove <code>ByVal</code>
Category	Style
Subcategory	Unnecessary code rules
Applicable languages	Visual Basic

Overview

This rule flags an unnecessary `ByVal` keyword in a parameter declaration in Visual Basic. Parameters in Visual Basic are `ByVal` by default, hence you do not need to explicitly specify it in method signatures. It tends to produce noisy code and often leads to the non-default `ByRef` keyword being overlooked.

Options

This rule has no associated code-style options.

Example

```
' Code with violations
Sub M(ByVal p1 As Integer, ByRef p2 As Integer)
End Sub

' Fixed code
Sub M(p1 As Integer, ByRef p2 As Integer)
End Sub
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0081
// The code that's violating the rule is on this line.
#pragma warning restore IDE0081
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.IDE0081.severity = none
```

To disable all of the code-style rules, set the severity for the category `Style` to `none` in the [configuration file](#).

```
[*.{cs,vb}]\ndotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [ByVal](#)
- [ByRef](#)
- [Unnecessary code rules](#)
- [Code style rules reference](#)

Remove unnecessary equality operator (IDE0100)

9/20/2022 • 2 minutes to read • [Edit Online](#)

PROPERTY	VALUE
Rule ID	IDE0100
Title	Remove unnecessary equality operator
Category	Style
Subcategory	Unnecessary code rules
Applicable languages	C# and Visual Basic

Overview

This style rule flags an unnecessary equality operator when comparing a non-constant Boolean expression with a constant `true` or `false`.

Options

This rule has no associated code-style options.

Example

```
// Code with violations
if (x == true) { }
if (M() != false) { }

// Fixed code
if (x) { }
if (M()) { }
```

```
' Code with violations
If x = True Then
End If

If M() <> False Then
End If

' Fixed code
If x Then
End If

If M() Then
End If
```

SUPPRESS A WARNING

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and

then re-enable the rule.

```
#pragma warning disable IDE0100
// The code that's violating the rule is on this line.
#pragma warning restore IDE0100
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.IDE0100.severity = none
```

To disable all of the code-style rules, set the severity for the category `Style` to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Unnecessary code rules](#)
- [Code style rules reference](#)

Remove unnecessary discard (IDE0110)

9/20/2022 • 2 minutes to read • [Edit Online](#)

PROPERTY	VALUE
Rule ID	IDE0110
Title	Remove unnecessary discard
Category	Style
Subcategory	Unnecessary code rules
Applicable languages	C#

Overview

This rule flags unnecessary [discard patterns](#). A discard pattern is unnecessary when used in a context where it has no effect.

Options

This rule has no associated code-style options.

Example

```
// Code with violations
switch (o)
{
    case int _:
        Console.WriteLine("Value was an int");
        break;
    case string _:
        Console.WriteLine("Value was a string");
        break;
}

// Fixed code
switch (o)
{
    case int:
        Console.WriteLine("Value was an int");
        break;
    case string:
        Console.WriteLine("Value was a string");
        break;
}
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0110
// The code that's violating the rule is on this line.
#pragma warning restore IDE0110
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.IDE0110.severity = none
```

To disable all of the code-style rules, set the severity for the category `Style` to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Unnecessary code rules](#)
- [Code style rules reference](#)

Simplify object creation (IDE0140)

9/20/2022 • 2 minutes to read • [Edit Online](#)

PROPERTY	VALUE
Rule ID	IDE0140
Title	Simplify object creation
Category	Style
Subcategory	Unnecessary code rules
Applicable languages	Visual Basic
Options	<code>visual_basic_style_prefer_simplified_object_creation</code>

Overview

This style rule flags unnecessary type repetition in Visual Basic code.

Options

Options specify the behavior that you want the rule to enforce. For information about configuring options, see [Option format](#).

`visual_basic_style_prefer_simplified_object_creation`

PROPERTY	VALUE	DESCRIPTION
Option name	<code>visual_basic_style_prefer_simplified_object_creation</code>	
Option values	<code>true</code>	Prefer simplified object creation form.
	<code>false</code>	Disables the rule.
Default option value	<code>true</code>	

Example

```
' Code with violations
Dim x As Student = New Student()

' Fixed code
Dim x As New Student()
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0140
// The code that's violating the rule is on this line.
#pragma warning restore IDE0140
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.IDE0140.severity = none
```

To disable all of the code-style rules, set the severity for the category `style` to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Unnecessary code rules](#)
- [Code style rules reference](#)

Miscellaneous rules

9/20/2022 • 2 minutes to read • [Edit Online](#)

This section contains code-style rules that don't fit in any other category. The miscellaneous rules are:

- [Remove invalid global 'SuppressMessageAttribute' \(IDE0076\)](#)
- [Avoid legacy format target in global 'SuppressMessageAttribute' \(IDE0077\)](#)

See also

- [Code style rules reference](#)

Remove invalid global 'SuppressMessageAttribute' (IDE0076)

9/20/2022 • 2 minutes to read • [Edit Online](#)

PROPERTY	VALUE
Rule ID	IDE0076
Title	Remove invalid global <code>SuppressMessageAttribute</code>
Category	CodeQuality
Subcategory	Miscellaneous rules
Applicable languages	C# and Visual Basic

Overview

This rule flags [global SuppressMessageAttributes](#) that have an invalid `Scope` or `Target`. The attribute should either be removed or fixed to refer to a valid scope and target symbol.

Options

This rule has no associated code-style options.

Example

```
// IDE0076: Invalid target '~F:N.C.F2' - no matching field named 'F2'
[assembly: System.Diagnostics.CodeAnalysis.SuppressMessage("Category", "Id: Title", Scope = "member", Target = "~F:N.C.F2")]
// IDE0076: Invalid scope 'property'
[assembly: System.Diagnostics.CodeAnalysis.SuppressMessage("Category", "Id: Title", Scope = "property", Target = "~P:N.C.P")]

// Fixed code
[assembly: System.Diagnostics.CodeAnalysis.SuppressMessage("Category", "Id: Title", Scope = "member", Target = "~F:N.C.F")]
[assembly: System.Diagnostics.CodeAnalysis.SuppressMessage("Category", "Id: Title", Scope = "member", Target = "~P:N.C.P")]

namespace N
{
    class C
    {
        public int F;
        public int P { get; }
    }
}
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0076
// The code that's violating the rule is on this line.
#pragma warning restore IDE0076
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.IDE0076.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-CodeQuality.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Global SuppressMessageAttribute](#)
- [Avoid legacy format target in global 'SuppressMessageAttribute' \(IDE0077\)](#)
- [Code style rules reference](#)

Avoid legacy format target in global 'SuppressMessageAttribute' (IDE0077)

9/20/2022 • 2 minutes to read • [Edit Online](#)

PROPERTY	VALUE
Rule ID	IDE0077
Title	Avoid legacy format target in global <code>SuppressMessageAttribute</code>
Category	CodeQuality
Subcategory	Miscellaneous rules
Applicable languages	C# and Visual Basic

Overview

This rule flags [global SuppressMessageAttributes](#) that specify `Target` using the [legacy FxCop](#) target string format. Using the legacy format `Target` is known to have performance problems and should be avoided. For more information, see [dotnet/roslyn issue 44362](#).

The recommended format for `Target` is the *documentation ID* format. For information about documentation IDs, see [Documentation ID format](#).

TIP

Visual Studio 2019 provides a code fix to automatically change the `Target` of the attribute to the recommended format.

Options

This rule has no associated code-style options.

Example

```
// IDE0077: Legacy format target 'N.C.#F'  
[assembly: System.Diagnostics.CodeAnalysis.SuppressMessage("Category", "Id: Title", Scope = "member", Target = "N.C.#F")]  
  
// Fixed code  
[assembly: System.Diagnostics.CodeAnalysis.SuppressMessage("Category", "Id: Title", Scope = "member", Target = "~F:N.C.F")]  
  
namespace N  
{  
    class C  
    {  
        public int F;  
    }  
}
```

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0077  
// The code that's violating the rule is on this line.  
#pragma warning restore IDE0077
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]  
dotnet_diagnostic.IDE0077.severity = none
```

To disable this entire category of rules, set the severity for the category to `none` in the [configuration file](#).

```
[*.{cs,vb}]  
dotnet_analyzer_diagnostic.category-CodeQuality.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [Global SuppressMessageAttribute](#)
- [Performance issues with legacy format attribute 'Target'](#)
- [Remove invalid global 'SuppressMessageAttribute' \(IDE0076\)](#)
- [Documentation ID format](#)

Formatting rule (IDE0055)

9/20/2022 • 2 minutes to read • [Edit Online](#)

All formatting options have rule ID IDE0055 and title [Fix formatting](#). These formatting options affect how indentation, spaces, and new lines are aligned around .NET programming language constructs. The options fall into the following categories and are documented on separate pages:

- [.NET formatting options](#)

Options that apply to both C# and Visual Basic. The EditorConfig names for these options start with the `dotnet_` prefix.

- [C# formatting options](#)

Options that are specific to the C# language. The EditorConfig names for these options start with the `csharp_` prefix.

When you set the severity of code-style rule IDE0055, it applies to all the formatting options. To set the severity of a formatting rule violation, add the following setting to a [configuration file](#).

```
dotnet_diagnostic.IDE0055.severity = <severity value>
```

The severity value must be `warning` or `error` to be [enforced on build](#). For all possible severity values, see [severity level](#).

For more information about configuring options, see [Option format](#).

Suppress a warning

If you want to suppress only a single violation, add preprocessor directives to your source file to disable and then re-enable the rule.

```
#pragma warning disable IDE0055
// The code that's violating the rule is on this line.
#pragma warning restore IDE0055
```

To disable the rule for a file, folder, or project, set its severity to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_diagnostic.IDE0055.severity = none
```

To disable all of the code-style rules, set the severity for the category `Style` to `none` in the [configuration file](#).

```
[*.{cs,vb}]
dotnet_analyzer_diagnostic.category-Style.severity = none
```

For more information, see [How to suppress code analysis warnings](#).

See also

- [.NET formatting options](#)
- [C# formatting options](#)
- [Language rules](#)
- [Naming rules](#)
- [.NET code style rules reference](#)

.NET formatting options

9/20/2022 • 2 minutes to read • [Edit Online](#)

The formatting options in this article apply to both C# and Visual Basic. These are options for code-style rule [IDE0055](#).

Using directive options

Use these options to customize how you want using directives to be sorted and grouped:

- [dotnet_sort_system_directives_first](#)
- [dotnet_separate_import_directive_groups](#)

Example `.editorconfig` file:

```
# .NET formatting rules
[*.{cs,vb}]
dotnet_sort_system_directives_first = true
dotnet_separate_import_directive_groups = true
```

TIP

If your code is C#, additional C#-specific `using` directive options are available.

`dotnet_sort_system_directives_first`

PROPERTY	VALUE	DESCRIPTION
Option name	<code>dotnet_sort_system_directives_first</code>	
Applicable languages	C# and Visual Basic	
Introduced version	Visual Studio 2017 version 15.3	
Option values	<code>true</code>	Sort <code>System.*</code> <code>using</code> directives alphabetically, and place them before other <code>using</code> directives.
	<code>false</code>	Do not place <code>System.*</code> <code>using</code> directives before other <code>using</code> directives.

Code examples:

```
// dotnet_sort_system_directives_first = true
using System.Collections.Generic;
using System.Threading.Tasks;
using Octokit;

// dotnet_sort_system_directives_first = false
using System.Collections.Generic;
using Octokit;
using System.Threading.Tasks;
```

dotnet_separate_import_directive_groups

PROPERTY	VALUE	DESCRIPTION
Option name	dotnet_separate_import_directive_groups	
Applicable languages	C# and Visual Basic	
Introduced version	Visual Studio 2017 version 15.5	
Option values	<code>true</code>	Place a blank line between <code>using</code> directive groups.
	<code>false</code>	Do not place a blank line between <code>using</code> directive groups.

Code examples:

```
// dotnet_separate_import_directive_groups = true
using System.Collections.Generic;
using System.Threading.Tasks;

using Octokit;

// dotnet_separate_import_directive_groups = false
using System.Collections.Generic;
using System.Threading.Tasks;
using Octokit;
```

Dotnet namespace options

This category contains one formatting option that concerns how namespaces are named in both C# and Visual Basic.

- [dotnet_style_namespace_match_folder](#)

Example `.editorconfig` file:

```
# .NET namespace rules
[*.{cs,vb}]
dotnet_style_namespace_match_folder = true
```

dotnet_style_namespace_match_folder

PROPERTY	VALUE	DESCRIPTION
Option name	dotnet_style_namespace_match_folder	
Applicable languages	C# and Visual Basic	
Introduced version	Visual Studio 2019 version 16.10	
Option values	<input checked="" type="checkbox"/> true <input type="checkbox"/> false	Match namespaces to folder structure Do not report on namespaces that do not match folder structure

Code examples:

```
// dotnet_style_namespace_match_folder = true
// file path: Example/Convention/C.cs
using System;

namespace Example.Convention
{
    class C
    {
    }
}

// dotnet_style_namespace_match_folder = false
// file path: Example/Convention/C.cs
using System;

namespace Example
{
    class C
    {
    }
}
```

NOTE

`dotnet_style_namespace_match_folder` requires the analyzer to have access to project properties to function correctly. For projects that target .NET Core 3.1 or an earlier version, you must manually add the following items to your project file. (They're added automatically for .NET 5 and later.)

```
<ItemGroup>
  <CompilerVisibleProperty Include="RootNamespace" />
  <CompilerVisibleProperty Include="ProjectDir" />
</ItemGroup>
```

See also

- [Formatting rule \(IDE0055\)](#)

C# formatting options

9/20/2022 • 14 minutes to read • [Edit Online](#)

The formatting options in this article apply only to C# code. These are options for code-style rule [IDE0055](#).

New-line options

The new-line options concern the use of new lines to format code.

- [csharp_new_line_before_open_brace](#)
- [csharp_new_line_before_else](#)
- [csharp_new_line_before_catch](#)
- [csharp_new_line_before_finally](#)
- [csharp_new_line_before_members_in_object_initializers](#)
- [csharp_new_line_before_members_in_anonymous_types](#)
- [csharp_new_line_between_query_expression_clauses](#)

Example `.editorconfig` file:

```
# CSharp formatting rules:  
[*.cs]  
csharp_new_line_before_open_brace = methods, properties, control_blocks, types  
csharp_new_line_before_else = true  
csharp_new_line_before_catch = true  
csharp_new_line_before_finally = true  
csharp_new_line_before_members_in_object_initializers = true  
csharp_new_line_before_members_in_anonymous_types = true  
csharp_new_line_between_query_expression_clauses = true
```

csharp_new_line_before_open_brace

This option concerns whether an open brace `{` should be placed on the same line as the preceding code, or on a new line. For this rule, you specify **all**, **none**, or one or more code elements such as **methods** or **properties**, to define when this rule should be applied. To specify multiple code elements, separate them with a comma (,).

PROPERTY	VALUE	DESCRIPTION
Option name	<code>csharp_new_line_before_open_brace</code>	
Applicable languages	C#	
Introduced version	Visual Studio 2017 version 15.3	
Option values	<code>all</code>	Require braces to be on a new line for all expressions ("Allman" style).
	<code>none</code>	Require braces to be on the same line for all expressions ("K&R").

PROPERTY	VALUE	DESCRIPTION
	accessors , anonymous_methods , anonymous_types , control_blocks , events , indexers , lambdas , local_functions , methods , object_collection_array_initializers , properties , types	Require braces to be on a new line for the specified code element ("Allman" style).

Code examples:

```
// csharp_new_line_before_open_brace = all
void MyMethod()
{
    if (...)

    ...
}

// csharp_new_line_before_open_brace = none
void MyMethod() {
    if (...) {
        ...
    }
}
```

csharp_new_line_before_else

PROPERTY	VALUE	DESCRIPTION
Option name	csharp_new_line_before_else	
Applicable languages	C#	
Introduced version	Visual Studio 2017 version 15.3	
Option values	true	Place <code>else</code> statements on a new line.
	false	Place <code>else</code> statements on the same line.

Code examples:

```
// csharp_new_line_before_else = true
if (...) {
    ...
}
else {
    ...
}

// csharp_new_line_before_else = false
if (...) {
    ...
} else {
    ...
}
```

csharp_new_line_before_catch

PROPERTY	VALUE	DESCRIPTION
Option name	csharp_new_line_before_catch	
Applicable languages	C#	
Introduced version	Visual Studio 2017 version 15.3	
Option values	<code>true</code>	Place <code>catch</code> statements on a new line.
	<code>false</code>	Place <code>catch</code> statements on the same line.

Code examples:

```
// csharp_new_line_before_catch = true
try {
    ...
}
catch (Exception e) {
    ...
}

// csharp_new_line_before_catch = false
try {
    ...
} catch (Exception e) {
    ...
}
```

csharp_new_line_before_finally

PROPERTY	VALUE	DESCRIPTION
Option name	csharp_new_line_before_finally	
Applicable languages	C#	
Introduced version	Visual Studio 2017 version 15.3	

PROPERTY	VALUE	DESCRIPTION
Option values	true	Require <code>finally</code> statements to be on a new line after the closing brace.
	false	Require <code>finally</code> statements to be on the same line as the closing brace.

Code examples:

```
// csharp_new_line_before_finally = true
try {
    ...
}
catch (Exception e) {
    ...
}
finally {
    ...
}

// csharp_new_line_before_finally = false
try {
    ...
} catch (Exception e) {
    ...
} finally {
    ...
}
```

csharp_new_line_before_members_in_object_initializers

PROPERTY	VALUE	DESCRIPTION
Option name	csharp_new_line_before_members_in_object_initializers	
Applicable languages	C#	
Introduced version	Visual Studio 2017 version 15.3	
Option values	true	Require members of object initializers to be on separate lines
	false	Require members of object initializers to be on the same line

Code examples:

```
// csharp_new_line_before_members_in_object_initializers = true
var z = new B()
{
    A = 3,
    B = 4
}

// csharp_new_line_before_members_in_object_initializers = false
var z = new B()
{
    A = 3, B = 4
}
```

csharp_new_line_before_members_in_anonymous_types

PROPERTY	VALUE	DESCRIPTION
Option name	csharp_new_line_before_members_in_a nonymous_types	
Applicable languages	C#	
Introduced version	Visual Studio 2017 version 15.3	
Option values	<input type="checkbox"/> true	Require members of anonymous types to be on separate lines
	<input type="checkbox"/> false	Require members of anonymous types to be on the same line

Code examples:

```
// csharp_new_line_before_members_in_anonymous_types = true
var z = new
{
    A = 3,
    B = 4
}

// csharp_new_line_before_members_in_anonymous_types = false
var z = new
{
    A = 3, B = 4
}
```

csharp_new_line_between_query_expression_clauses

PROPERTY	VALUE	DESCRIPTION
Option name	csharp_new_line_between_query_expre ssion_clauses	
Applicable languages	C#	
Introduced version	Visual Studio 2017 version 15.3	

PROPERTY	VALUE	DESCRIPTION
Option values	<code>true</code>	Require elements of query expression clauses to be on separate lines
	<code>false</code>	Require elements of query expression clauses to be on the same line

Code examples:

```
// csharp_new_line_between_query_expression_clauses = true
var q = from a in e
        from b in e
        select a * b;

// csharp_new_line_between_query_expression_clauses = false
var q = from a in e from b in e
        select a * b;
```

Indentation options

The indentation options concern the use of indentation to format code.

- [csharp_indent_case_contents](#)
- [csharp_indent_switch_labels](#)
- [csharp_indent_labels](#)
- [csharp_indent_block_contents](#)
- [csharp_indent_braces](#)
- [csharp_indent_case_contents_when_block](#)

Example `.editorconfig` file:

```
# CSharp formatting rules:
[*.cs]
csharp_indent_case_contents = true
csharp_indent_switch_labels = true
csharp_indent_labels = flush_left
csharp_indent_block_contents = true
csharp_indent_braces = false
csharp_indent_case_contents_when_block = true
```

csharp_indent_case_contents

PROPERTY	VALUE	DESCRIPTION
Option name	<code>csharp_indent_case_contents</code>	
Applicable languages	C#	
Introduced version	Visual Studio 2017 version 15.3	
Option values	<code>true</code>	Indent <code>switch</code> case contents
	<code>false</code>	Do not indent <code>switch</code> case contents

Code examples:

```
// csharp_indent_case_contents = true
switch(c) {
    case Color.Red:
        Console.WriteLine("The color is red");
        break;
    case Color.Blue:
        Console.WriteLine("The color is blue");
        break;
    default:
        Console.WriteLine("The color is unknown.");
        break;
}

// csharp_indent_case_contents = false
switch(c) {
    case Color.Red:
        Console.WriteLine("The color is red");
        break;
    case Color.Blue:
        Console.WriteLine("The color is blue");
        break;
    default:
        Console.WriteLine("The color is unknown.");
        break;
}
```

csharp_indent_switch_labels

PROPERTY	VALUE	DESCRIPTION
Option name	csharp_indent_switch_labels	
Applicable languages	C#	
Introduced version	Visual Studio 2017 version 15.3	
Option values	<code>true</code>	Indent <code>switch</code> labels
	<code>false</code>	Do not indent <code>switch</code> labels

Code examples:

```
// csharp_indent_switch_labels = true
switch(c) {
    case Color.Red:
        Console.WriteLine("The color is red");
        break;
    case Color.Blue:
        Console.WriteLine("The color is blue");
        break;
    default:
        Console.WriteLine("The color is unknown.");
        break;
}

// csharp_indent_switch_labels = false
switch(c) {
case Color.Red:
    Console.WriteLine("The color is red");
    break;
case Color.Blue:
    Console.WriteLine("The color is blue");
    break;
default:
    Console.WriteLine("The color is unknown.");
    break;
}
```

csharp_indent_labels

PROPERTY	VALUE	DESCRIPTION
Option name	csharp_indent_labels	
Applicable languages	C#	
Introduced version	Visual Studio 2017 version 15.3	
Option values	<input type="button" value="flush_left"/>	Labels are placed at the leftmost column
	<input type="button" value="one_less_than_current"/>	Labels are placed at one less indent to the current context
	<input type="button" value="no_change"/>	Labels are placed at the same indent as the current context

Code examples:

```

// csharp_indent_labels= flush_left
class C
{
    private string MyMethod(...)
    {
        if (...) {
            goto error;
        }
    error:
        throw new Exception(...);
    }
}

// csharp_indent_labels = one_less_than_current
class C
{
    private string MyMethod(...)
    {
        if (...) {
            goto error;
        }
    error:
        throw new Exception(...);
    }
}

// csharp_indent_labels= no_change
class C
{
    private string MyMethod(...)
    {
        if (...) {
            goto error;
        }
    error:
        throw new Exception(...);
    }
}

```

csharp_indent_block_contents

PROPERTY	VALUE	DESCRIPTION
Option name	csharp_indent_block_contents	
Applicable languages	C#	
Option values	<input type="checkbox"/> true	Indent block contents.
	<input type="checkbox"/> false	Don't indent block contents.

Code examples:

```
// csharp_indent_block_contents = true
static void Hello()
{
    Console.WriteLine("Hello");
}

// csharp_indent_block_contents = false
static void Hello()
{
    Console.WriteLine("Hello");
}
```

csharp_indent_braces

PROPERTY	VALUE	DESCRIPTION
Option name	csharp_indent_braces	
Applicable languages	C#	
Option values	<code>true</code>	Indent curly braces.
	<code>false</code>	Don't indent curly braces.

Code examples:

```
// csharp_indent_braces = true
static void Hello()
{
    Console.WriteLine("Hello");
}

// csharp_indent_braces = false
static void Hello()
{
    Console.WriteLine("Hello");
}
```

csharp_indent_case_contents_when_block

PROPERTY	VALUE	DESCRIPTION
Option name	csharp_indent_case_contents_when_block	
Applicable languages	C#	
Option values	<code>true</code>	When it's a block, indent the statement list and curly braces for a case in a switch statement.
	<code>false</code>	When it's a block, don't indent the statement list and curly braces for a case in a switch statement.

Code examples:

```
// csharp_indent_case_contents_when_block = true
case 0:
{
    Console.WriteLine("Hello");
    break;
}

// csharp_indent_case_contents_when_block = false
case 0:
{
    Console.WriteLine("Hello");
    break;
}
```

Spacing options

The spacing options concern the use of space characters to format code.

- [csharp_space_after_cast](#)
- [csharp_space_after_keywords_in_control_flow_statements](#)
- [csharp_space_between_parentheses](#)
- [csharp_space_before_colon_in_inheritance_clause](#)
- [csharp_space_after_colon_in_inheritance_clause](#)
- [csharp_space_around_binary_operators](#)
- [csharp_space_between_method_declaration_parameter_list_parentheses](#)
- [csharp_space_between_method_declaration_empty_parameter_list_parentheses](#)
- [csharp_space_between_method_declaration_name_and_open_parenthesis](#)
- [csharp_space_between_method_call_parameter_list_parentheses](#)
- [csharp_space_between_method_call_empty_parameter_list_parentheses](#)
- [csharp_space_between_method_call_name_and_opening_parenthesis](#)
- [csharp_space_after_comma](#)
- [csharp_space_before_comma](#)
- [csharp_space_after_dot](#)
- [csharp_space_before_dot](#)
- [csharp_space_after_semicolon_in_for_statement](#)
- [csharp_space_before_semicolon_in_for_statement](#)
- [csharp_space_around_declaration_statements](#)
- [csharp_space_before_open_square_brackets](#)
- [csharp_space_between_empty_square_brackets](#)
- [csharp_space_between_square_brackets](#)

Example `.editorconfig` file:

```

# CSharp formatting rules:
[*.cs]
csharp_space_after_cast = true
csharp_space_after_keywords_in_control_flow_statements = true
csharp_space_between_parentheses = control_flow_statements, type_casts
csharp_space_before_colon_in_inheritance_clause = true
csharp_space_after_colon_in_inheritance_clause = true
csharp_space_around_binary_operators = before_and_after
csharp_space_between_method_declaration_parameter_list_parentheses = true
csharp_space_between_method_declaration_empty_parameter_list_parentheses = false
csharp_space_between_method_declaration_name_and_open_parenthesis = false
csharp_space_between_method_call_parameter_list_parentheses = true
csharp_space_between_method_call_empty_parameter_list_parentheses = false
csharp_space_between_method_call_name_and_opening_parenthesis = false
csharp_space_after_comma = true
csharp_space_before_comma = false
csharp_space_after_dot = false
csharp_space_before_dot = false
csharp_space_after_semicolon_in_for_statement = true
csharp_space_before_semicolon_in_for_statement = false
csharp_space_around_declaration_statements = false
csharp_space_before_open_square_brackets = false
csharp_space_between_empty_square_brackets = false
csharp_space_between_square_brackets = false

```

csharp_space_after_cast

PROPERTY	VALUE	DESCRIPTION
Option name	csharp_space_after_cast	
Applicable languages	C#	
Introduced version	Visual Studio 2017 version 15.3	
Option values	<input checked="" type="checkbox"/> true <input type="checkbox"/> false	Place a space character between a cast and the value
	<input type="checkbox"/> false	Remove space between the cast and the value

Code examples:

```

// csharp_space_after_cast = true
int y = (int) x;

// csharp_space_after_cast = false
int y = (int)x;

```

csharp_space_after_keywords_in_control_flow_statements

PROPERTY	VALUE	DESCRIPTION
Option name	csharp_space_after_keywords_in_control_flow_statements	
Applicable languages	C#	

PROPERTY	VALUE	DESCRIPTION
Introduced version	Visual Studio 2017 version 15.3	
Option values	<code>true</code>	Place a space character after a keyword in a control flow statement such as a <code>for</code> loop
	<code>false</code>	Remove space after a keyword in a control flow statement such as a <code>for</code> loop

Code examples:

```
// csharp_space_after_keywords_in_control_flow_statements = true
for (int i;i<x;i++) { ... }

// csharp_space_after_keywords_in_control_flow_statements = false
for(int i;i<x;i++) { ... }
```

csharp_space_between_parentheses

PROPERTY	VALUE	DESCRIPTION
Option name	csharp_space_between_parentheses	
Applicable languages	C#	
Introduced version	Visual Studio 2017 version 15.3	
Option values	<code>control_flow_statements</code>	Place space between parentheses of control flow statements
	<code>expressions</code>	Place space between parentheses of expressions
	<code>type_casts</code>	Place space between parentheses in type casts

If you omit this rule or use a value other than `control_flow_statements`, `expressions`, or `type_casts`, the setting is not applied.

Code examples:

```
// csharp_space_between_parentheses = control_flow_statements
for ( int i = 0; i < 10; i++ ) { }

// csharp_space_between_parentheses = expressions
var z = ( x * y ) - ( ( y - x ) * 3 );

// csharp_space_between_parentheses = type_casts
int y = ( int )x;
```

csharp_space_before_colon_in_inheritance_clause

PROPERTY	VALUE	DESCRIPTION
Option name	csharp_space_before_colon_in_inheritance_clause	
Applicable languages	C#	
Introduced version	Visual Studio 2017 version 15.7	
Option values	<input checked="" type="checkbox"/> true <input type="checkbox"/> false	Place a space character before the colon for bases or interfaces in a type declaration
	<input type="checkbox"/> false	Remove space before the colon for bases or interfaces in a type declaration

Code examples:

```
// csharp_space_before_colon_in_inheritance_clause = true
interface I
{
}

class C : I
{
}

// csharp_space_before_colon_in_inheritance_clause = false
interface I
{
}

class C: I
{}
```

csharp_space_after_colon_in_inheritance_clause

PROPERTY	VALUE	DESCRIPTION
Option name	csharp_space_after_colon_in_inheritance_clause	
Applicable languages	C#	
Introduced version	Visual Studio 2017 version 15.7	
Option values	<input checked="" type="checkbox"/> true <input type="checkbox"/> false	Place a space character after the colon for bases or interfaces in a type declaration
	<input type="checkbox"/> false	Remove space after the colon for bases or interfaces in a type declaration

Code examples:

```
// csharp_space_after_colon_in_inheritance_clause = true
interface I
{
}

class C : I
{
}

// csharp_space_after_colon_in_inheritance_clause = false
interface I
{
}

class C :I
{}
```

csharp_space_around_binary_operators

PROPERTY	VALUE	DESCRIPTION
Option name	csharp_space_around_binary_operators	
Applicable languages	C#	
Introduced version	Visual Studio 2017 version 15.7	
Option values	<code>before_and_after</code>	Insert space before and after the binary operator
	<code>none</code>	Remove spaces before and after the binary operator
	<code>ignore</code>	Ignore spaces around binary operators

If you omit this rule, or use a value other than `before_and_after`, `none`, or `ignore`, the setting is not applied.

Code examples:

```
// csharp_space_around_binary_operators = before_and_after
return x * (x - y);

// csharp_space_around_binary_operators = none
return x*(x-y);

// csharp_space_around_binary_operators = ignore
return x * (x-y);
```

csharp_space_between_method_declaration_parameter_list_parentheses

PROPERTY	VALUE	DESCRIPTION
Option name	csharp_space_between_method_declar ation_parameter_list_parentheses	
Applicable languages	C#	
Introduced version	Visual Studio 2017 version 15.3	
Option values	<code>true</code>	Place a space character after the opening parenthesis and before the closing parenthesis of a method declaration parameter list
	<code>false</code>	Remove space characters after the opening parenthesis and before the closing parenthesis of a method declaration parameter list

Code examples:

```
// csharp_space_between_method_declaration_parameter_list_parentheses = true
void Bark( int x ) { ... }

// csharp_space_between_method_declaration_parameter_list_parentheses = false
void Bark(int x) { ... }
```

csharp_space_between_method_declaration_empty_parameter_list_parentheses

PROPERTY	VALUE	DESCRIPTION
Option name	csharp_space_between_method_declar ation_empty_parameter_list_parentheses	
Applicable languages	C#	
Introduced version	Visual Studio 2017 version 15.7	
Option values	<code>true</code>	Insert space within empty parameter list parentheses for a method declaration
	<code>false</code>	Remove space within empty parameter list parentheses for a method declaration

Code examples:

```
// csharp_space_between_method_declaration_empty_parameter_list_parentheses = true
void Goo( )
{
    Goo(1);
}

void Goo(int x)
{
    Goo();
}

// csharp_space_between_method_declaration_empty_parameter_list_parentheses = false
void Goo()
{
    Goo(1);
}

void Goo(int x)
{
    Goo();
}
```

csharp_space_between_method_declaration_name_and_open_parenthesis

PROPERTY	VALUE	DESCRIPTION
Option name	csharp_space_between_method_declaration_name_and_open_parenthesis	
Applicable languages	C#	
Option values	<input checked="" type="checkbox"/> true <input type="checkbox"/> false	Place a space character between the method name and opening parenthesis in the method declaration
	<input type="checkbox"/> false	Remove space characters between the method name and opening parenthesis in the method declaration

Code examples:

```
// csharp_space_between_method_declaration_name_and_open_parenthesis = true
void M () { }

// csharp_space_between_method_declaration_name_and_open_parenthesis = false
void M() { }
```

csharp_space_between_method_call_parameter_list_parentheses

PROPERTY	VALUE	DESCRIPTION
Option name	csharp_space_between_method_call_parameter_list_parentheses	
Applicable languages	C#	
Introduced version	Visual Studio 2017 version 15.3	

PROPERTY	VALUE	DESCRIPTION
Option values	true	Place a space character after the opening parenthesis and before the closing parenthesis of a method call
	false	Remove space characters after the opening parenthesis and before the closing parenthesis of a method call

Code examples:

```
// csharp_space_between_method_call_parameter_list_parentheses = true
MyMethod( argument );

// csharp_space_between_method_call_parameter_list_parentheses = false
MyMethod(argument);
```

csharp_space_between_method_call_empty_parameter_list_parentheses

PROPERTY	VALUE	DESCRIPTION
Option name	csharp_space_between_method_call_empty_parameter_list_parentheses	
Applicable languages	C#	
Introduced version	Visual Studio 2017 version 15.7	
Option values	true	Insert space within empty argument list parentheses
	false	Remove space within empty argument list parentheses

Code examples:

```
// csharp_space_between_method_call_empty_parameter_list_parentheses = true
void Goo()
{
    Goo(1);
}

void Goo(int x)
{
    Goo( );
}

// csharp_space_between_method_call_empty_parameter_list_parentheses = false
void Goo()
{
    Goo(1);
}

void Goo(int x)
{
    Goo();
}
```

csharp_space_between_method_call_name_and_opening_parenthesis

PROPERTY	VALUE	DESCRIPTION
Option name	csharp_space_between_method_call_name_and_opening_parenthesis	
Applicable languages	C#	
Introduced version	Visual Studio 2017 version 15.7	
Option values	<input checked="" type="checkbox"/> true <input type="checkbox"/> false	Insert space between method call name and opening parenthesis Remove space between method call name and opening parenthesis

Code examples:

```
// csharp_space_between_method_call_name_and_opening_parenthesis = true
void Goo()
{
    Goo (1);
}

void Goo(int x)
{
    Goo ();
}

// csharp_space_between_method_call_name_and_opening_parenthesis = false
void Goo()
{
    Goo(1);
}

void Goo(int x)
{
    Goo();
}
```

csharp_space_after_comma

PROPERTY	VALUE	DESCRIPTION
Option name	csharp_space_after_comma	
Applicable languages	C#	
Option values	<input checked="" type="checkbox"/> true <input type="checkbox"/> false	Insert space after a comma Remove space after a comma

Code examples:

```
// csharp_space_after_comma = true
int[] x = new int[] { 1, 2, 3, 4, 5 };

// csharp_space_after_comma = false
int[] x = new int[] { 1,2,3,4,5 }
```

csharp_space_before_comma

PROPERTY	VALUE	DESCRIPTION
Option name	csharp_space_before_comma	
Applicable languages	C#	
Option values	<input type="button" value="true"/>	Insert space before a comma
	<input type="button" value="false"/>	Remove space before a comma

Code examples:

```
// csharp_space_before_comma = true
int[] x = new int[] { 1 , 2 , 3 , 4 , 5 };

// csharp_space_before_comma = false
int[] x = new int[] { 1, 2, 3, 4, 5 };
```

csharp_space_after_dot

PROPERTY	VALUE	DESCRIPTION
Option name	csharp_space_after_dot	
Applicable languages	C#	
Option values	<input type="button" value="true"/>	Insert space after a dot
	<input type="button" value="false"/>	Remove space after a dot

Code examples:

```
// csharp_space_after_dot = true
this. Goo();

// csharp_space_after_dot = false
this.Goo();
```

csharp_space_before_dot

PROPERTY	VALUE	DESCRIPTION
Option name	csharp_space_before_dot	
Applicable languages	C#	

PROPERTY	VALUE	DESCRIPTION
Option values	true	Insert space before a dot
	false	Remove space before a dot

Code examples:

```
// csharp_space_before_dot = true
this .Goo();

// csharp_space_before_dot = false
this.Goo();
```

csharp_space_after_semicolon_in_for_statement

PROPERTY	VALUE	DESCRIPTION
Option name	csharp_space_after_semicolon_in_for_st atement	
Applicable languages	C#	
Option values	true	Insert space after each semicolon in a <code>for</code> statement
	false	Remove space after each semicolon in a <code>for</code> statement

Code examples:

```
// csharp_space_after_semicolon_in_for_statement = true
for (int i = 0; i < x.Length; i++)

// csharp_space_after_semicolon_in_for_statement = false
for (int i = 0;i < x.Length;i++)
```

csharp_space_before_semicolon_in_for_statement

PROPERTY	VALUE	DESCRIPTION
Option name	csharp_space_before_semicolon_in_for_ statement	
Applicable languages	C#	
Option values	true	Insert space before each semicolon in a <code>for</code> statement
	false	Remove space before each semicolon in a <code>for</code> statement

Code examples:

```
// csharp_space_before_semicolon_in_for_statement = true
for (int i = 0 ; i < x.Length ; i++)

// csharp_space_before_semicolon_in_for_statement = false
for (int i = 0; i < x.Length; i++)
```

csharp_space_around_declaration_statements

PROPERTY	VALUE	DESCRIPTION
Option name	csharp_space_around_declaration_statements	
Applicable languages	C#	
Option values	<code>ignore</code>	Don't remove extra space characters in declaration statements
	<code>false</code>	Remove extra space characters in declaration statements

Code examples:

```
// csharp_space_around_declaration_statements = ignore
int    x    =    0    ;

// csharp_space_around_declaration_statements = false
int x = 0;
```

csharp_space_before_open_square_brackets

PROPERTY	VALUE	DESCRIPTION
Option name	csharp_space_before_open_square_brackets	
Applicable languages	C#	
Option values	<code>true</code>	Insert space before opening square brackets [
	<code>false</code>	Remove space before opening square brackets [

Code examples:

```
// csharp_space_before_open_square_brackets = true
int [] numbers = new int [] { 1, 2, 3, 4, 5 };

// csharp_space_before_open_square_brackets = false
int[] numbers = new int[] { 1, 2, 3, 4, 5 };
```

csharp_space_between_empty_square_brackets

PROPERTY	VALUE	DESCRIPTION
Option name	csharp_space_between_empty_square_brackets	
Applicable languages	C#	
Option values	<input checked="" type="checkbox"/> true <input type="checkbox"/> false	Insert space between empty square brackets []
	<input type="checkbox"/> false	Remove space between empty square brackets []

Code examples:

```
// csharp_space_between_empty_square_brackets = true
int[] numbers = new int[] { 1, 2, 3, 4, 5 };

// csharp_space_between_empty_square_brackets = false
int[] numbers = new int[] { 1, 2, 3, 4, 5 };
```

csharp_space_between_square_brackets

PROPERTY	VALUE	DESCRIPTION
Option name	csharp_space_between_square_brackets	
Applicable languages	C#	
Option values	<input checked="" type="checkbox"/> true <input type="checkbox"/> false	Insert space characters in non-empty square brackets [0]
	<input type="checkbox"/> false	Remove space characters in non-empty square brackets [0]

Code examples:

```
// csharp_space_between_square_brackets = true
int index = numbers[ 0 ];

// csharp_space_between_square_brackets = false
int index = numbers[0];
```

Wrap options

The wrap formatting options concern the use of single lines versus separate lines for statements and code blocks.

- [csharp_preserve_single_line_statements](#)
- [csharp_preserve_single_line_blocks](#)

Example `.editorconfig` file:

```
# CSharp formatting rules:  
[*.cs]  
csharp_preserve_single_line_statements = true  
csharp_preserve_single_line_blocks = true
```

csharp_preserve_single_line_statements

PROPERTY	VALUE	DESCRIPTION
Option name	csharp_preserve_single_line_statements	
Applicable languages	C#	
Introduced version	Visual Studio 2017 version 15.3	
Option values	<code>true</code>	Leave statements and member declarations on the same line
	<code>false</code>	Leave statements and member declarations on different lines

Code examples:

```
//csharp_preserve_single_line_statements = true
int i = 0; string name = "John";

//csharp_preserve_single_line_statements = false
int i = 0;
string name = "John";
```

csharp_preserve_single_line_blocks

PROPERTY	VALUE	DESCRIPTION
Option name	csharp_preserve_single_line_blocks	
Applicable languages	C#	
Introduced version	Visual Studio 2017 version 15.3	
Option values	<code>true</code>	Leave code block on single line
	<code>false</code>	Leave code block on separate lines

Code examples:

```
//csharp_preserve_single_line_blocks = true
public int Foo { get; set; }

//csharp_preserve_single_line_blocks = false
public int MyProperty
{
    get; set;
}
```

Using directive options

This category contains one formatting option that concerns whether `using` directives are placed inside or outside a namespace.

- [csharp_using_directive_placement](#)

Example `.editorconfig` file:

```
'using' directive preferences
[*.cs]
csharp_using_directive_placement = outside_namespace
csharp_using_directive_placement = inside_namespace
```

TIP

For additional `using` directive options, see [.NET using directive options](#).

csharp_using_directive_placement

PROPERTY	VALUE	DESCRIPTION
Option name	<code>csharp_using_directive_placement</code>	
Applicable languages	C#	
Introduced version	Visual Studio 2019 version 16.1	
Option values	<code>outside_namespace</code>	Leave using directives outside namespace
	<code>inside_namespace</code>	Leave using directives inside namespace

Code examples:

```
// csharp_using_directive_placement = outside_namespace
using System;

namespace Conventions
{
}

// csharp_using_directive_placement = inside_namespace
namespace Conventions
{
    using System;
}
```

See also

- [Formatting rule \(IDE0055\)](#)

Code-style naming rules

9/20/2022 • 7 minutes to read • [Edit Online](#)

In your `.editorconfig` file, you can define naming conventions for your .NET programming language code elements—such as classes, properties, and methods—and how the compiler or IDE should enforce those conventions. For example, you could specify that a public member that isn't capitalized should be treated as a compiler error, or that if a private field doesn't begin with an `_`, a build warning should be issued.

Specifically, you can define a **naming rule**, which consists of three parts:

- The **symbol group** that the rule applies to, for example, public members or private fields.
- The **naming style** to associate with the rule, for example, that the name must be capitalized or start with an underscore.
- The severity level of the message when code elements included in the symbol group don't follow the naming style.

General syntax

To define any of the above entities—a naming rule, symbol group, or naming style—set one or more properties using the following syntax:

```
<kind>.<entityName>.<propertyName> = <propertyValue>
```

All the property settings for a given `kind` and `entityName` make up that specific entity definition.

Each property should only be set once, but some settings allow multiple, comma-separated values.

The order of the properties is not important.

<kind> values

<kind> specifies which kind of entity is being defined—naming rule, symbol group, or naming style—and must be one of the following:

TO SET A PROPERTY FOR	USE THE <KIND> VALUE	EXAMPLE
Naming rule	<code>dotnet_naming_rule</code>	<code>dotnet_naming_rule.types_should_be_pascal_case.se = suggestion</code>
Symbol group	<code>dotnet_naming_symbols</code>	<code>dotnet_naming_symbols.interface.applicable_kinds = interface</code>
Naming style	<code>dotnet_naming_style</code>	<code>dotnet_naming_style.pascal_case.capitalization = pascal_case</code>

<entityName>

<entityName> is a descriptive name you choose that associates multiple property settings into a single definition. For example, the following properties produce two symbol group definitions, `interface` and `types`, each of which has two properties set on it.

```

dotnet_naming_symbols.interface.applicable_kinds = interface
dotnet_naming_symbols.interface.applicable_accessibilities = public, internal, private, protected,
protected_internal, private_protected

dotnet_naming_symbols.types.applicable_kinds = class, struct, interface, enum, delegate
dotnet_naming_symbols.types.applicable_accessibilities = public, internal, private, protected,
protected_internal, private_protected

```

<propertyName> and <propertyValue>

Each kind of entity—[naming rule](#), [symbol group](#), or [naming style](#)—has its own supported properties, as described in the following sections.

Symbol group properties

You can set the following properties for symbol groups, to limit which symbols are included in the group. To specify multiple values for a single property, separate the values with a comma.

PROPERTY	DESCRIPTION	ALLOWED VALUES	REQUIRED
<code>applicable_kinds</code>	Kinds of symbols in the group ¹	<input type="checkbox"/> * (use this value to specify all symbols) <input type="checkbox"/> namespace <input type="checkbox"/> class <input type="checkbox"/> struct <input type="checkbox"/> interface <input type="checkbox"/> enum <input type="checkbox"/> property <input type="checkbox"/> method <input type="checkbox"/> field <input type="checkbox"/> event <input type="checkbox"/> delegate <input type="checkbox"/> parameter <input type="checkbox"/> type_parameter <input type="checkbox"/> local <input type="checkbox"/> local_function	Yes
<code>applicable_accessibilities</code>	Accessibility levels of the symbols in the group	<input type="checkbox"/> * (use this value to specify all accessibility levels) <input type="checkbox"/> public <input type="checkbox"/> internal or friend <input type="checkbox"/> private <input type="checkbox"/> protected <input type="checkbox"/> protected_internal or protected_friend <input type="checkbox"/> private_protected <input type="checkbox"/> local (for symbols defined within a method)	Yes
<code>required_modifiers</code>	Only match symbols with <i>all</i> the specified modifiers ²	<input type="checkbox"/> abstract or must_inherit <input type="checkbox"/> async <input type="checkbox"/> const <input type="checkbox"/> readonly <input type="checkbox"/> static or shared ³	No

Notes:

1. Tuple members aren't currently supported in `applicable_kinds`.
2. The symbol group matches *all* the modifiers in the `required_modifiers` property. If you omit this property, no specific modifiers are required for a match. This means a symbol's modifiers have no effect on whether or not this rule is applied.
3. If your group has `static` or `shared` in the `required_modifiers` property, the group will also include `const`.

symbols because they are implicitly `static` / `shared`. However, if you don't want the `static` naming rule to apply to `const` symbols, you can create a new naming rule with a symbol group of `const`.

4. `class` includes [C# records](#).

Naming style properties

A naming style defines the conventions you want to enforce with the rule. For example:

- Capitalize with `PascalCase`
- Starts with `m_`
- Ends with `_g`
- Separate words with `_`

You can set the following properties for a naming style:

PROPERTY	DESCRIPTION	ALLOWED VALUES	REQUIRED
<code>capitalization</code>	Capitalization style for words within the symbol	<code>pascal_case</code> <code>camel_case</code> <code>first_word_upper</code> <code>all_upper</code> <code>all_lower</code>	Yes ¹
<code>required_prefix</code>	Must begin with these characters		No
<code>required_suffix</code>	Must end with these characters		No
<code>word_separator</code>	Words within the symbol need to be separated with this character		No

Notes:

1. You must specify a capitalization style as part of your naming style, otherwise your naming style might be ignored.

Naming rule properties

All naming rule properties are required for a rule to take effect.

PROPERTY	DESCRIPTION
<code>symbols</code>	The name of a symbol group defined elsewhere; the naming rule will be applied to the symbols in this group
<code>style</code>	The name of the naming style which should be associated with this rule; the style is defined elsewhere
<code>severity</code>	Sets the severity with which to enforce the naming rule. Set the associated value to one of the available severity levels . ¹

Notes:

1. Severity specification within a naming rule is only respected inside development IDEs, such as Visual Studio. This setting is not understood by the C# or VB compilers, hence not respected during build. To enforce naming style rules on build, you should instead set the severity by using [code rule severity configuration](#). For more information, see this [GitHub issue](#).

Rule order

The order in which naming rules are defined in an EditorConfig file doesn't matter. The naming rules are automatically ordered according to the definition of the rules themselves. The [EditorConfig Language Service](#)

[extension](#) can analyze an EditorConfig file and report cases where the rule ordering in the file is different to what the compiler will use at run time.

NOTE

If you're using a version of Visual Studio earlier than Visual Studio 2019 version 16.2, naming rules should be ordered from most-specific to least-specific in the EditorConfig file. The first rule encountered that can be applied is the only rule that is applied. However, if there are multiple rule *properties* with the same name, the most recently found property with that name takes precedence. For more information, see [File hierarchy and precedence](#).

Default naming styles

If you don't specify any custom naming rules, the following default styles are used:

- For classes, structs, enumerations, properties, methods, and events with any accessibility, the default naming style is Pascal case.
- For interfaces with any accessibility, the default naming style is Pascal case with a required prefix of I.

Code Rule ID: `IDE1006 (Naming rule violation)`

All naming options have rule ID `IDE1006` and title `Naming rule violation`. You can configure the severity of naming violations globally in an EditorConfig file with the following syntax:

```
dotnet_diagnostic. IDE1006.severity = <severity value>
```

The severity value must be `warning` or `error` to be [enforced on build](#). For all possible severity values, see [severity level](#).

Example: Public member capitalization

The following `.editorconfig` file contains a naming convention that specifies that public properties, methods, fields, events, and delegates must be capitalized. Notice that this naming convention specifies multiple kinds of symbol to apply the rule to, using a comma to separate the values.

```
[*.{cs,vb}]

# Defining the 'public_symbols' symbol group
dotnet_naming_symbols.public_symbols.applicable_kinds      = property,method,field,event,delegate
dotnet_naming_symbols.public_symbols.applicable_accessibilities = public
dotnet_naming_symbols.public_symbols.required_modifiers     = readonly

# Defining the 'first_word_upper_case_style' naming style
dotnet_naming_style.first_word_upper_case_style.capitalization = first_word_upper

# Defining the 'public_members_must_be_capitalized' naming rule, by setting the
# symbol group to the 'public_symbols' symbol group,
dotnet_naming_rule.public_members_must_be_capitalized.symbols = public_symbols
# setting the naming style to the 'first_word_upper_case_style' naming style,
dotnet_naming_rule.public_members_must_be_capitalized.style   = first_word_upper_case_style
# and setting the severity.
dotnet_naming_rule.public_members_must_be_capitalized.severity = suggestion
```

Example: Private instance fields with underscore

This `.editorconfig` file snippet enforces that private instance fields should start with an `_`; if that convention is not followed, the IDE will treat it as a compiler error. Private static fields are ignored.

Because you can only define a symbol group based on the identifiers it has (for example, `static` or `readonly`), and not by the identifiers it doesn't have (for example, an instance field because it doesn't have `static`), you need to define two naming rules:

1. All private fields—`static` or not—should have the `underscored` naming style applied to them as a compiler `error`.
2. Private fields with `static` should have the `underscored` naming style applied to them with a severity level of `none`; in other words, ignore this case.

```
[*.{cs,vb}]

# Define the 'private_fields' symbol group:
dotnet_naming_symbols.private_fields.applicable_kinds = field
dotnet_naming_symbols.private_fields.applicable_accessibilities = private

# Define the 'private_static_fields' symbol group
dotnet_naming_symbols.private_static_fields.applicable_kinds = field
dotnet_naming_symbols.private_static_fields.applicable_accessibilities = private
dotnet_naming_symbols.private_static_fields.required_modifiers = static

# Define the 'underscored' naming style
dotnet_naming_style.underscored.capitalization = pascal_case
dotnet_naming_style.underscored.required_prefix = _

# Define the 'private_fields_underscored' naming rule
dotnet_naming_rule.private_fields_underscored.symbols = private_fields
dotnet_naming_rule.private_fields_underscored.style = underscored
dotnet_naming_rule.private_fields_underscored.severity = error

# Define the 'private_static_fields_none' naming rule
dotnet_naming_rule.private_static_fields_none.symbols = private_static_fields
dotnet_naming_rule.private_static_fields_none.style = underscored
dotnet_naming_rule.private_static_fields_none.severity = none
```

This example also demonstrates that entity definitions can be reused. The `underscored` naming style is used by both the `private_fields_underscored` and `private_static_fields_none` naming rules.

See also

- [Language rules](#)
- [Formatting rules](#)
- [Roslyn naming rules](#)
- [.NET code style rules reference](#)

Platform compatibility analyzer

9/20/2022 • 17 minutes to read • [Edit Online](#)

You've probably heard the motto of "One .NET": a single, unified platform for building any type of application. The .NET 5 SDK includes ASP.NET Core, Entity Framework Core, WinForms, WPF, Xamarin, and ML.NET, and will add support for more platforms over time. .NET 5 strives to provide an experience where you don't have to reason about the different flavors of .NET, but doesn't attempt to fully abstract away the underlying operating system (OS). You'll continue to be able to call platform-specific APIs, for example, P/Invokes, WinRT, or the Xamarin bindings for iOS and Android.

But using platform-specific APIs on a component means the code no longer works across all platforms. We needed a way to detect this at design time so developers get diagnostics when they inadvertently use platform-specific APIs. To achieve this goal, .NET 5 introduces the [platform compatibility analyzer](#) and complementary APIs to help developers identify and use platform-specific APIs where appropriate.

The new APIs include:

- [SupportedOSPlatformAttribute](#) to annotate APIs as being platform-specific and [UnsupportedOSPlatformAttribute](#) to annotate APIs as being unsupported on a particular OS. These attributes can optionally include the version number, and have already been applied to some platform-specific APIs in the core .NET libraries.
- `Is<Platform>()` and `Is<Platform>VersionAtLeast(int major, int minor = 0, int build = 0, int revision = 0)` static methods in the [System.OperatingSystem](#) class for safely calling platform-specific APIs. For example, `OperatingSystem.IsWindows()` can be used to guard a call to a Windows-specific API, and `OperatingSystem.IsWindowsVersionAtLeast()` can be used to guard a versioned Windows-specific API call. See these [examples](#) of how these methods can be used as guards of platform-specific API references.

Prerequisites

The platform compatibility analyzer is one of the Roslyn code quality analyzers. Starting in .NET 5, these analyzers are [included with the .NET SDK](#). The platform compatibility analyzer is enabled by default only for projects that target `net5.0` or a later version. However, you can [enable it](#) for projects that target other frameworks.

How the analyzer determines platform dependency

- An **unattributed API** is considered to work on **all OS platforms**.
- An API marked with `[SupportedOSPlatform("platform")]` is considered only portable to the specified platform and any platforms it's a subset of.
 - The attribute can be applied multiple times to indicate **multiple platform support**, for example `[SupportedOSPlatform("windows"), SupportedOSPlatform("Android29.0")]`.
 - If the platform is a [subset of another platform](#), the attribute implies that the superset platform is also supported. For example, `[SupportedOSPlatform("ios")]` implies that the API is supported on `ios` and also on its superset platform, `MacCatalyst`.
 - The analyzer will produce a **warning** if platform-specific APIs are referenced without a proper **platform context**:
 - **Warns** if the project doesn't target the supported platform (for example, a Windows-specific API called from a project targeting iOS `<TargetFramework>net5.0-ios14.0</TargetFramework>`).

- **Warns** if the project is cross-platform and calls platform-specific APIs (for example, a Windows-specific API called from cross platform TFM
`<TargetFramework>net5.0</TargetFramework>`).
- **Doesn't warn** if the platform-specific API is referenced within a project that targets any of the specified platforms (for example, for a Windows-specific API called from a project targeting windows
`<TargetFramework>net5.0-windows</TargetFramework>` and the *AssemblyInfo.cs* file generation is enabled for the project).
- **Doesn't warn** if the platform-specific API call is guarded by corresponding platform-check methods (for example, a Windows-specific API call guarded by
`OperatingSystem.IsWindows()`).
- **Doesn't warn** if the platform-specific API is referenced from the same platform-specific context (call site also attributed with
`[SupportedOSPlatform("platform")]`).
- An API marked with
`[UnsupportedOSPlatform("platform")]` is considered to be unsupported on the specified platform and any platforms it's a subset of, but supported for all other platforms.
 - The attribute can be applied multiple times with different platforms, for example,
`[UnsupportedOSPlatform("iOS"), UnsupportedOSPlatform("Android29.0")]`.
 - If the platform is a **subset of another platform**, the attribute implies that the superset platform is also unsupported. For example,
`[UnsupportedOSPlatform("iOS")]` implies that the API is unsupported on
`iOS` and also on its superset platform, `MacCatalyst`.
 - The analyzer produces a **warning** only if the `platform` is effective for the call site:
 - **Warns** if the project targets the platform that's attributed as unsupported (for example, if the API is attributed with
`[UnsupportedOSPlatform("windows")]` and the call site targets
`<TargetFramework>net5.0-windows</TargetFramework>`).
 - **Warns** if the project is multi-targeted and the `platform` is included in the default **MSBuild**
`<SupportedPlatform>` items group, or the `platform` is manually included within the
`MSBuild <SupportedPlatform>` items group:

```

<ItemGroup>
  <SupportedPlatform Include="platform" />
</ItemGroup>

```
 - **Doesn't warn** if you're building an app that doesn't target the unsupported platform or is multi-targeted and the platform is not included in the default **MSBuild**
`<SupportedPlatform>` items group.
- Both attributes can be instantiated with or without version numbers as part of the platform name. Version numbers are in the format of
`major.minor[.build[.revision]]`; `major.minor` is required and the `build` and `revision` parts are optional. For example, "Windows6.1" indicates Windows version 6.1, but "Windows" is interpreted as Windows 0.0.

For more information, see [examples of how the attributes work and what diagnostics they cause](#).

How the analyzer recognizes TFM target platforms

The analyzer does not check target framework moniker (TFM) target platforms from MSBuild properties, such as
`<TargetFramework>` or
`<TargetFrameworks>`. If the TFM has a target platform, MSBuild injects a
`SupportedOSPlatform` attribute with the targeted platform name in the *AssemblyInfo.cs* file, which is consumed by the analyzer. For example, if the TFM is
`net5.0-windows10.0.19041`, MSBuild injects the
`[assembly: System.Runtime.Versioning.SupportedOSPlatform("windows10.0.19041")]` attribute into the *AssemblyInfo.cs* file, and the entire assembly is considered to be Windows only. Therefore, calling Windows-only APIs versioned with 7.0 or below would not cause any warnings in the project.

NOTE

If the `AssemblyInfo.cs` file generation is disabled for the project (that is, the `<GenerateAssemblyInfo>` property is set to `false`), the required assembly level `SupportedOSPlatform` attribute can't be added by MSBuild. In this case, you could see warnings for a platform-specific APIs usage even if you're targeting that platform. To resolve the warnings, enable the `AssemblyInfo.cs` file generation or add the attribute manually in your project.

Platform inclusion

.NET 6 introduces the concept of *platform inclusion*, where one platform can be a subset of another platform. An annotation for the subset platform implies the same support (or lack thereof) for the superset platform. If a platform check method in the `OperatingSystem` type has a `[SupportedOSPlatformGuard("supersetPlatform")]` attribute, then `supersetPlatform` is considered a superset of the OS platform that the method checks for.

For example, the `OperatingSystem.IsiOS()` method is attributed `[SupportedOSPlatformGuard("MacCatalyst")]`. Therefore, the following statements apply:

- The `OperatingSystem.IsiOS()` and `OperatingSystem.IsiOSVersionAtLeast` methods check not only the `ios` platform, but also the `MacCatalyst` platform.
- `[SupportedOSPlatform("ios")]` implies that the API is supported on `ios` and also on its superset platform, `MacCatalyst`. You can use the `[UnsupportedOSPlatform("MacCatalyst")]` attribute to exclude this implied support.
- `[UnsupportedOSPlatform("ios")]` implies that the API is not supported on `ios` and `MacCatalyst`. You can use the `[SupportedOSPlatform("MacCatalyst")]` attribute to exclude this implied lack of support.

Consider the following coverage matrix, where \checkmark indicates that the platform is supported, and \times indicates that the platform is *not* supported.

PLATFORM	SUPPORTEDOSPLATFORM(SUBSET)	SUPPORTEDOSPLATFORM(SUPERSET)	UNSUPPORTEDOSPLATFORM(SUBSET)	UNSUPPORTEDOSPLATFORM(SUPERSET)
Subset	\checkmark	\times	\checkmark	\times
Superset	\checkmark	\checkmark	\checkmark	\checkmark

TIP

The same rules apply for the `SupportedOSPlatformGuard` and `UnsupportedOSPlatformGuard` attributes.

The following code snippet shows how you can combine attributes to set the right level of support.

```

// MacCatalyst is a superset of iOS therefore supported on iOS and MacCatalyst
[SupportedOSPlatform("iOS")]
public void ApiOnlySupportedOnIOSAndMacCatalyst() { }

// Does not imply iOS, only supported on MacCatalyst
[SupportedOSPlatform("MacCatalyst")]
public void ApiOnlySupportedOnMacCatalyst() { }

[SupportedOSPlatform("iOS")] // Supported on iOS and MacCatalyst
[UnsupportedOSPlatform("MacCatalyst")] // Removes implied MacCatalyst support
public void ApiOnlySupportedOnIos() { }

// Unsupported on iOS and MacCatalyst
[UnsupportedOSPlatform("iOS")]
public void ApiUnsupportedOnIOSAndMacCatalyst();

// Does not imply iOS, only unsupported on MacCatalyst
[UnsupportedOSPlatform("MacCatalyst")]
public void ApiUnsupportedOnMacCatalyst() { }

[UnsupportedOSPlatform("iOS")] // Unsupported on iOS and MacCatalyst
[SupportedOSPlatform("MacCatalyst")] // Removes implied MacCatalyst unsupportedness
public void ApiUnsupportedOnIos() { }

```

Advanced scenarios for attribute combinations

- If a combination of `[SupportedOSPlatform]` and `[UnsupportedOSPlatform]` attributes are present, all attributes are grouped by OS platform identifier:
 - **Supported only list.** If the lowest version for each OS platform is a `[SupportedOSPlatform]` attribute, the API is considered to only be supported by the listed platforms and unsupported by all other platforms. The optional `[UnsupportedOSPlatform]` attributes for each platform can only have higher version of the minimum supported version, which denotes that the API is removed starting from the specified version.

```

// API is only supported on Windows from version 6.2 to 10.0.19041.0 and all versions of Linux
// The API is considered not supported for all other platforms.
[SupportedOSPlatform("windows6.2")]
[UnsupportedOSPlatform("windows10.0.19041.0")]
[SupportedOSPlatform("linux")]
public void ApiSupportedFromWindows80SupportFromCertainVersion();

```

- **Unsupported only list.** If the lowest version for each OS platform is an `[UnsupportedOSPlatform]` attribute, then the API is considered to only be unsupported by the listed platforms and supported by all other platforms. The list could have `[SupportedOSPlatform]` attribute with the same platform but a higher version, which denotes that the API is supported starting from that version.

```

// The API is unsupported on all Linux versions was unsupported on Windows until version
10.0.19041.0.
// The API is considered supported everywhere else without constraints.
[UnsupportedOSPlatform("windows")]
[SupportedOSPlatform("windows10.0.19041.0")]
[UnsupportedOSPlatform("linux")]
public void ApiSupportedFromWindows8UnsupportedFromWindows10();

```

- **Inconsistent list.** If the lowest version for some platforms is `[SupportedOSPlatform]` while it is `[UnsupportedOSPlatform]` for other platforms, it's considered to be inconsistent, which is not supported for the analyzer. If inconsistency occurs, the analyzer ignores the `[UnsupportedOSPlatform]` platforms.

- If the lowest versions of `[SupportedOSPlatform]` and `[UnsupportedOSPlatform]` attributes are equal, the analyzer considers the platform as part of the **Supported only** list.
- Platform attributes can be applied to types, members (methods, fields, properties, and events) and assemblies with different platform names or versions.
 - Attributes applied at the top level `target` affect all of its members and types.
 - Child-level attributes only apply if they adhere to the rule "child annotations can narrow the platforms support, but they cannot widen it".
 - When parent has **Supported only** list, then child member attributes cannot add a new platform support, as that would be extending parent support. Support for a new platform can only be added to the parent itself. But the child can have the `Supported` attribute for the same platform with later versions as that narrows the support. Also, the child can have the `Unsupported` attribute with the same platform as that also narrows parent support.
 - When parent has **Unsupported only** list, then child member attributes can add support for a new platform, as that narrows parent support. But it cannot have the `Supported` attribute for the same platform as the parent, because that extends parent support. Support for the same platform can only be added to the parent where the original `Unsupported` attribute was applied.
 - If `[SupportedOSPlatform("platformVersion")]` is applied more than once for an API with the same `platform` name, the analyzer only considers the one with the minimum version.
 - If `[UnsupportedOSPlatform("platformVersion")]` is applied more than twice for an API with the same `platform` name, the analyzer only considers the two with the earliest versions.

NOTE

An API that was supported initially but unsupported (removed) in a later version is not expected to get re-supported in an even later version.

Examples of how the attributes work and what diagnostics they produce

```
// An API supported only on Windows all versions.
[SupportedOSPlatform("Windows")]
public void WindowsOnlyApi() { }

// an API supported on Windows and Linux.
[SupportedOSPlatform("Windows")]
[SupportedOSPlatform("Linux")]
public void SupportedOnWindowsAndLinuxOnly() { }

// an API only supported on Windows 6.2 and later, not supported for all other.
// an API is removed/unsupported from version 10.0.19041.0.
[SupportedOSPlatform("windows6.2")]
[UnsupportedOSPlatform("windows10.0.19041.0")]
public void ApiSupportedFromWindows8UnsupportedFromWindows10() { }

// an Assembly supported on Windows, the API added from version 10.0.19041.0.
[assembly: SupportedOSPlatform("Windows")]
[SupportedOSPlatform("windows10.0.19041.0")]
public void AssemblySupportedOnWindowsApiSupportedFromWindows10() { }

public void Caller()
{
    WindowsOnlyApi(); // warns: This call site is reachable on all platforms. 'WindowsOnlyApi()' is only
                      // supported on: 'windows'

    // This call site is reachable on all platforms. 'SupportedOnWindowsAndLinuxOnly()' is only supported
    // on: 'Windows', 'Linux'
    SupportedOnWindowsAndLinuxOnly();

    // This call site is reachable on all platforms. 'ApiSupportedFromWindows8UnsupportedFromWindows10()' is
    // only supported on: 'Windows', 'Linux'
    ApiSupportedFromWindows8UnsupportedFromWindows10();
}
```

```

only supported on: 'windows' from version 6.2 to 10.0.19041.0
    ApiSupportedFromWindows8UnsupportedFromWindows10();

    // for same platform analyzer only warn for the latest version.
    // This call site is reachable on all platforms. 'AssemblySupportedOnWindowsApiSupportedFromWindows10()'
is only supported on: 'windows' 10.0.19041.0 and later
    AssemblySupportedOnWindowsApiSupportedFromWindows10();
}

// an API not supported on android but supported on all other.
[UnsupportedOSPlatform("android")]
public void DoesNotWorkOnAndroid() { }

// an API was unsupported on Windows until version 6.2.
// The API is considered supported everywhere else without constraints.
[UnsupportedOSPlatform("windows")]
[SupportedOSPlatform("windows6.2")]
public void StartedWindowsSupportFromVersion8() { }

// an API was unsupported on Windows until version 6.2.
// Then the API is removed (unsupported) from version 10.0.19041.0.
// The API is considered supported everywhere else without constraints.
[UnsupportedOSPlatform("windows")]
[SupportedOSPlatform("windows6.2")]
[UnsupportedOSPlatform("windows10.0.19041.0")]
public void StartedWindowsSupportFrom8UnsupportedFrom10() { }

public void Caller2()
{
    DoesNotWorkOnAndroid(); // This call site is reachable on all platforms. 'DoesNotWorkOnAndroid()' is
unsupported on: 'android'

    // This call site is reachable on all platforms. 'StartedWindowsSupportFromVersion8()' is unsupported
on: 'windows' 6.2 and before.
    StartedWindowsSupportFromVersion8();

    // This call site is reachable on all platforms. 'StartedWindowsSupportFrom8UnsupportedFrom10()' is
supported on: 'windows' from version 6.2 to 10.0.19041.0
    StartedWindowsSupportFrom8UnsupportedFrom10();
}

```

Handle reported warnings

The recommended way to deal with these diagnostics is to make sure you only call platform-specific APIs when running on an appropriate platform. Following are the options you can use to address the warnings; choose whichever is most appropriate for your situation:

- **Guard the call.** You can achieve this by conditionally calling the code at run time. Check whether you're running on a desired `Platform` by using one of platform-check methods, for example,
`OperatingSystem.Is<Platform>()` or
`OperatingSystem.Is<Platform>VersionAtLeast(int major, int minor = 0, int build = 0, int revision = 0)`.
[Example](#).
- **Mark the call site as platform-specific.** You can also choose to mark your own APIs as being platform-specific, thus effectively just forwarding the requirements to your callers. Mark the containing method or type or the entire assembly with the same attributes as the referenced platform-dependent call. [Examples](#).
- **Assert the call site with platform check.** If you don't want the overhead of an additional `if` statement at run time, use `Debug.Assert(Boolean)`. [Example](#).
- **Delete the code.** Generally not what you want because it means you lose fidelity when your code is

used by Windows users. For cases where a cross-platform alternative exists, you're likely better off using that over platform-specific APIs.

- **SUPPRESS THE WARNING.** You can also simply suppress the warning, either via an [EditorConfig](#) entry or `#pragma warning disable CA1416`. However, this option should be a last resort when using platform-specific APIs.

TIP

When disabling warnings using the `#pragma` pre-compiler directives, the identifiers you're targeting are case sensitive. For example, `ca1416` would not actually disable warning CA1416.

Guard platform-specific APIs with guard methods

The guard method's platform name should match with the calling platform-dependent API platform name. If the platform string of the calling API includes the version:

- For the `[SupportedOSPlatform("platformVersion")]` attribute, the guard method platform `version` should be greater than or equal to the calling platform's `Version`.
- For the `[UnsupportedOSPlatform("platformVersion")]` attribute, the guard method's platform `version` should be less than or equal to the calling platform's `Version`.

```

public void CallingSupportedOnlyApis() // Allow list calls
{
    if (OperatingSystem.IsWindows())
    {
        WindowsOnlyApi(); // will not warn
    }

    if (OperatingSystem.IsLinux())
    {
        SupportedOnWindowsAndLinuxOnly(); // will not warn, within one of the supported context
    }

    // Can use &&, || logical operators to guard combined attributes
    if (OperatingSystem.IsWindowsVersionAtLeast(6, 2) && !OperatingSystem.IsWindowsVersionAtLeast(10,
0, 19041))
    {
        ApiSupportedFromWindows8UnsupportedFromWindows10();
    }

    if (OperatingSystem.IsWindowsVersionAtLeast(10, 0, 19041, 0))
    {
        AssemblySupportedOnWindowsApiSupportedFromWindows10(); // Only need to check latest supported
version
    }
}

public void CallingUnsupportedApis()
{
    if (!OperatingSystem.IsAndroid())
    {
        DoesNotWorkOnAndroid(); // will not warn
    }

    if (!OperatingSystem.IsWindows() || OperatingSystem.IsWindowsVersionAtLeast(6, 2))
    {
        StartedWindowsSupportFromVersion8(); // will not warn
    }

    if (!OperatingSystem.IsWindows() || // supported all other platforms
        (OperatingSystem.IsWindowsVersionAtLeast(6, 2) && !OperatingSystem.IsWindowsVersionAtLeast(10,
0, 19041)))
    {
        StartedWindowsSupportFrom8UnsupportedFrom10(); // will not warn
    }
}

```

- If you need to guard code that targets `netstandard` or `netcoreapp` where new `OperatingSystem` APIs are not available, the `RuntimeInformation.OSPlatform` API can be used and will be respected by the analyzer. But it's not as optimized as the new APIs added in `OperatingSystem`. If the platform is not supported in the `OSPlatform` struct, you can call `OSPlatform.Create(String)` and pass in the platform name, which the analyzer also respects.

```
public void CallingSupportedOnlyApis()
{
    if (RuntimeInformation.IsOSPlatform(OSPlatform.Windows))
    {
        SupportedOnWindowsAndLinuxOnly(); // will not warn
    }

    if (RuntimeInformation.IsOSPlatform(OSPlatform.Create("browser")))
    {
        ApiOnlySupportedOnBrowser(); // call of browser specific API
    }
}
```

Annotate APIs with platform guard attributes and use it as a custom guard

As shown previously, the analyzer recognizes the platform-guard static methods in the [OperatingSystem](#) type, such as `OperatingSystem.IsWindows`, and also [RuntimeInformation.IsOSPlatform](#). However, you might want to cache the guard result in a field and reuse it, or use custom guard methods for checking a platform. The analyzer needs to recognize such APIs as a custom guard and should not warn for the APIs guarded by them. The guard attributes were introduced in .NET 6 to support this scenario:

- `SupportedOSPlatformGuardAttribute` annotates APIs that can be used as a guard for APIs annotated with [SupportedOSPlatformAttribute](#).
- `UnsupportedOSPlatformGuardAttribute` annotates APIs that can be used as a guard for APIs annotated with [UnsupportedOSPlatformAttribute](#).

These attributes can optionally include a version number. They can be applied multiple times to guard more than one platform and can be used for annotating a field, property, or method.

```

class Test
{
    [UnsupportedOSPlatformGuard("browser")] // The platform guard attribute
#if TARGET_BROWSER
    internal bool IsSupported => false;
#else
    internal bool IsSupported => true;
#endif

    [UnsupportedOSPlatform("browser")]
    void ApiNotSupportedOnBrowser() { }

    void M1()
    {
        ApiNotSupportedOnBrowser(); // Warns: This call site is reachable on all
        platforms.'ApiNotSupportedOnBrowser()' is unsupported on: 'browser'

        if (IsSupported)
        {
            ApiNotSupportedOnBrowser(); // Not warn
        }
    }

    [SupportedOSPlatform("Windows")]
    [SupportedOSPlatform("Linux")]
    void ApiOnlyWorkOnWindowsLinux() { }

    [SupportedOSPlatformGuard("Linux")]
    [SupportedOSPlatformGuard("Windows")]
    private readonly bool _isWindowOrLinux = OperatingSystem.IsLinux() || OperatingSystem.IsWindows();

    void M2()
    {
        ApiOnlyWorkOnWindowsLinux(); // This call site is reachable on all
        platforms.'ApiOnlyWorkOnWindowsLinux()' is only supported on: 'Linux', 'Windows'.

        if (_isWindowOrLinux)
        {
            ApiOnlyWorkOnWindowsLinux(); // Not warn
        }
    }
}

```

Mark call site as platform-specific

Platform names should match the calling platform-dependent API. If the platform string includes a version:

- For the `[SupportedOSPlatform("platformVersion")]` attribute, the call site platform `version` should be greater than or equal to the calling platform's `Version`
- For the `[UnsupportedOSPlatform("platformVersion")]` attribute, the call site platform `version` should be less than or equal to the calling platform's `Version`

```

// an API supported only on Windows.
[SupportedOSPlatform("windows")]
public void WindowsOnlyApi() { }

// an API supported on Windows and Linux.
[SupportedOSPlatform("Windows")]
[SupportedOSPlatform("Linux")]
public void SupportedOnWindowsAndLinuxOnly() { }

// an API only supported on Windows 6.2 and later, not supported for all other.
// an API is removed/unsupported from version 10.0.19041.0.
[SupportedOSPlatform("windows6.2")]

```

```
[UnsupportedOSPlatform("windows10.0.19041.0")]
public void ApiSupportedFromWindows8UnsupportedFromWindows10() { }

// an Assembly supported on Windows, the API added from version 10.0.19041.0.
[assembly: SupportedOSPlatform("Windows")]
[SupportedOSPlatform("windows10.0.19041.0")]
public void AssemblySupportedOnWindowsApiSupportedFromWindows10() { }

[SupportedOSPlatform("windows6.2")] // call site attributed Windows 6.2 or above.
public void Caller()
{
    WindowsOnlyApi(); // will not warn as call site is for Windows.

    // will not warn as call site is for Windows all versions.
    SupportedOnWindowsAndLinuxOnly();

    // will not warn for the [SupportedOSPlatform("windows6.2")] attribute, but warns for
    [UnsupportedOSPlatform("windows10.0.19041.0")]
    // This call site is reachable on: 'windows' 6.2 and later.
    'ApiSupportedFromWindows8UnsupportedFromWindows10()' is unsupported on: 'windows' 10.0.19041.0 and
    later.
    ApiSupportedFromWindows8UnsupportedFromWindows10();

    // The call site version is lower than the calling version, so warns:
    // This call site is reachable on: 'windows' 6.2 and later.
    'AssemblySupportedOnWindowsApiSupportedFromWindows10()' is only supported on: 'windows' 10.0.19041.0
    and later
    AssemblySupportedOnWindowsApiSupportedFromWindows10();
}

[SupportedOSPlatform("windows10.0.22000")] // call site attributed with windows 10.0.22000 or above.
public void Caller2()
{
    // This call site is reachable on: 'windows' 10.0.22000 and later.
    'ApiSupportedFromWindows8UnsupportedFromWindows10()' is unsupported on: 'windows' 10.0.19041.0 and
    later.
    ApiSupportedFromWindows8UnsupportedFromWindows10();

    // will not warn as call site version higher than calling API.
    AssemblySupportedOnWindowsApiSupportedFromWindows10();
}

[SupportedOSPlatform("windows6.2")]
[UnsupportedOSPlatform("windows10.0.19041.0")] // call site supports Windows from version 6.2 to
10.0.19041.0.
public void Caller3()
{
    // will not warn as caller has exact same attributes.
    ApiSupportedFromWindows8UnsupportedFromWindows10();

    // The call site reachable for the version not supported in the calling API, therefore warns:
    // This call site is reachable on: 'windows' from version 6.2 to 10.0.19041.0.
    'AssemblySupportedOnWindowsApiSupportedFromWindows10()' is only supported on: 'windows' 10.0.19041.0
    and later.
    AssemblySupportedOnWindowsApiSupportedFromWindows10();
}

// an API not supported on Android but supported on all other.
[UnsupportedOSPlatform("android")]
public void DoesNotWorkOnAndroid() { }

// an API was unsupported on Windows until version 6.2.
// The API is considered supported everywhere else without constraints.
[UnsupportedOSPlatform("windows")]
[SupportedOSPlatform("windows6.2")]
public void StartedWindowsSupportFromVersion8() { }

// an API was unsupported on Windows until version 6.2.
// Then the API is removed (unsupported) from version 10.0.19041.0.
```

```

// The API is considered supported everywhere else without constraints.
[UnsupportedOSPlatform("windows")]
[SupportedOSPlatform("windows6.2")]
[UnsupportedOSPlatform("windows10.0.19041.0")]
public void StartedWindowsSupportFrom8UnsupportedFrom10() { }

[UnsupportedOSPlatform("windows")] // Caller no support Windows for any version.
public void Caller4()
{
    // This call site is reachable on all platforms.'DoesNotWorkOnAndroid()' is unsupported on:
    'android'
    DoesNotWorkOnAndroid();

    // will not warns as the call site not support Windows at all, but supports all other.
    StartedWindowsSupportFromVersion8();

    // same, will not warns as the call site not support Windows at all, but supports all other.
    StartedWindowsSupportFrom8UnsupportedFrom10();
}

[UnsupportedOSPlatform("windows")]
[UnsupportedOSPlatform("android")] // Caller not support Windows and Android for any version.
public void Caller4()
{
    DoesNotWorkOnAndroid(); // will not warn as call site not supports Android.

    // will not warns as the call site not support Windows at all, but supports all other.
    StartedWindowsSupportFromVersion8();

    // same, will not warns as the call site not support Windows at all, but supports all other.
    StartedWindowsSupportFrom8UnsupportedFrom10();
}

```

Assert the call-site with platform check

All the conditional checks used in the [platform guard examples](#) can also be used as the condition for `Debug.Assert(Boolean)`.

```

// An API supported only on Linux.
[SupportedOSPlatform("linux")]
public void LinuxOnlyApi() { }

public void Caller()
{
    Debug.Assert(OperatingSystem.IsLinux());

    LinuxOnlyApi(); // will not warn
}

```

See also

- [Target Framework Names in .NET 5](#)
- [Annotating platform-specific APIs and detecting its use](#)
- [Annotating APIs as unsupported on specific platforms](#)
- [CA1416 Platform compatibility analyzer](#)

The .NET Portability Analyzer

9/20/2022 • 5 minutes to read • [Edit Online](#)

NOTE

We're in the process of deprecating API Port in favor of integrating binary analysis directly into [.NET Upgrade Assistant](#). In the upcoming months, we're going to shut down the backend service of API Port, which will require using the tool offline. For more information, see [GitHub: .NET API Port repository](#).*

Want to make your libraries support multi-platform? Want to see how much work is required to make your .NET Framework application run on .NET Core? The [.NET Portability Analyzer](#) is a tool that analyzes assemblies and provides a detailed report on .NET APIs that are missing for the applications or libraries to be portable on your specified targeted .NET platforms. The Portability Analyzer is offered as a [Visual Studio Extension](#), which analyzes one assembly per project, and as an [ApiPort console app](#), which analyzes assemblies by specified files or directory.

Once you've converted your project to target the new platform, like .NET Core, you can use the Roslyn-based [Platform compatibility analyzer](#) to identify APIs that throw [PlatformNotSupportedException](#) exceptions and other compatibility issues.

Common targets

- [.NET Core](#): Has a modular design, supports side-by-side installation, and targets cross-platform scenarios. Side-by-side installation allows you to adopt new .NET Core versions without breaking other apps. If your goal is to port your app to .NET Core and support multiple platforms, this is the recommended target.
- [.NET Standard](#): Includes the .NET Standard APIs available on all .NET implementations. If your goal is to make your library run on all .NET supported platforms, this is recommended target.
- [ASP.NET Core](#): A modern web-framework built on .NET Core. If your goal is to port your web app to .NET Core to support multiple platforms, this is the recommended target.
- .NET Core + [Platform Extensions](#): Includes the .NET Core APIs in addition to the Windows Compatibility Pack, which provides many of the .NET Framework available technologies. This is a recommended target for porting your app from .NET Framework to .NET Core on Windows.
- .NET Standard + [Platform Extensions](#): Includes the .NET Standard APIs in addition to the Windows Compatibility Pack, which provides many of the .NET Framework available technologies. This is a recommended target for porting your library from .NET Framework to .NET Core on Windows.

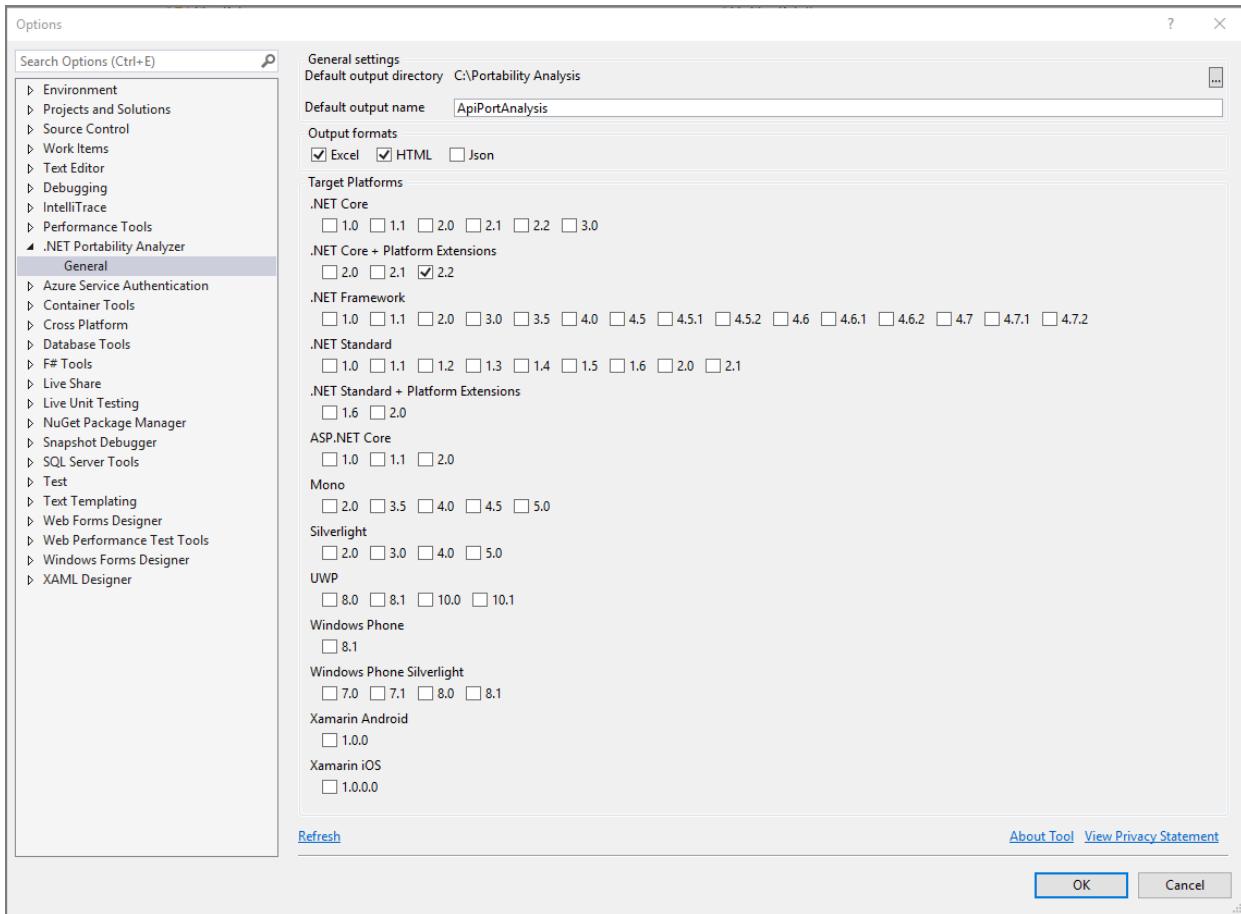
How to use the .NET Portability Analyzer

To begin using the .NET Portability Analyzer in Visual Studio, you first need to download and install the extension from the [Visual Studio Marketplace](#). It works on Visual Studio 2017 and Visual Studio 2019 versions.

IMPORTANT

The .NET Portability Analyzer is not supported in Visual Studio 2022.

Configure it in Visual Studio via [Analyze > Portability Analyzer Settings](#) and select your Target Platforms, which are the .NET platforms/versions that you want to evaluate the portability gaps comparing with the platform/version that your current assembly is built with.



You can also use the ApiPort console application, download it from [ApiPort repository](#). You can use `listTargets` command option to display the available target list, then pick target platforms by specifying `-t` or `--target` command option.

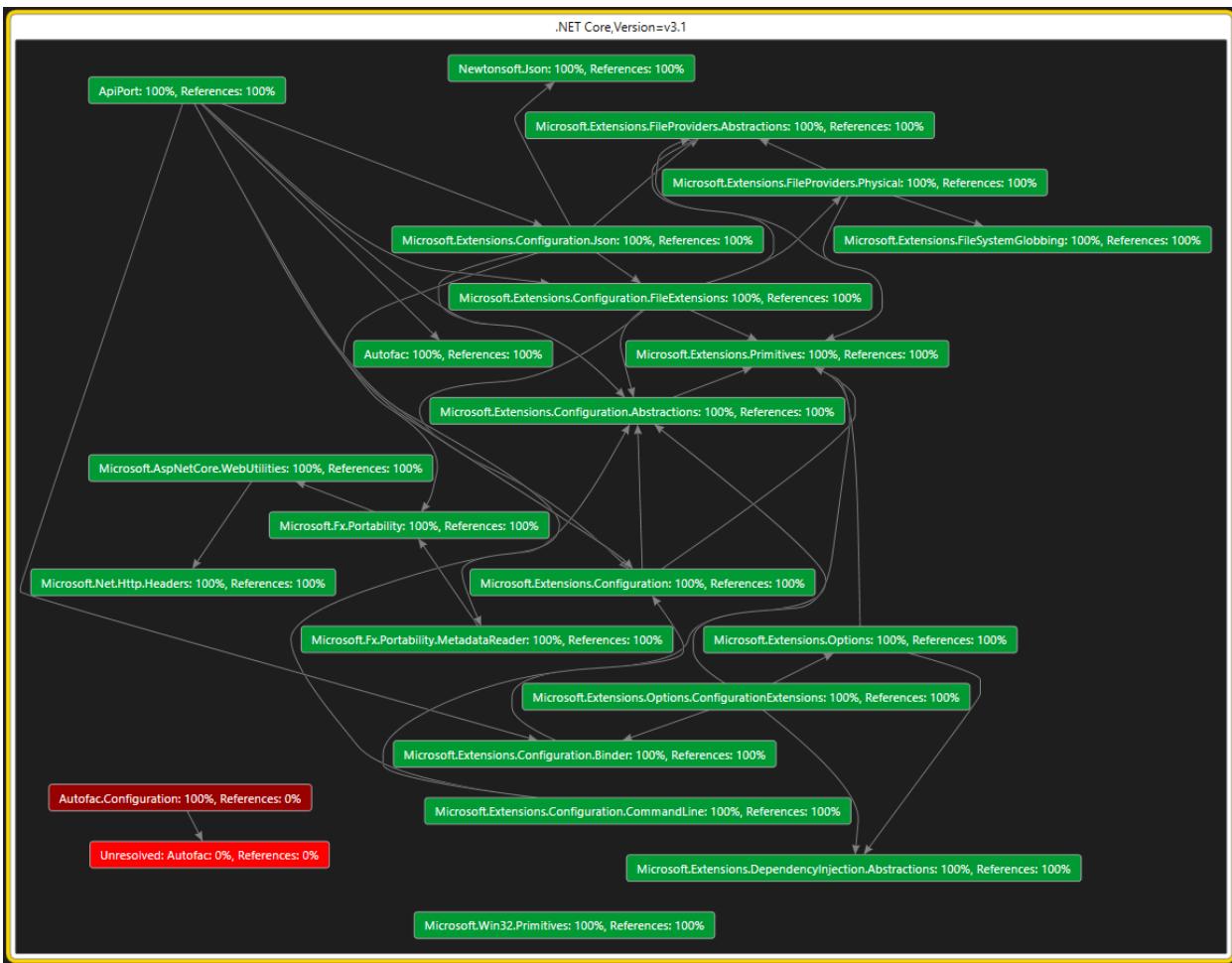
Solution wide view

A useful step in analyzing a solution with many projects would be to visualize the dependencies to understand which subset of assemblies depend on what. The general recommendation is to apply the results of the analysis in a bottom-up approach starting with the leaf nodes in a dependency graph.

To retrieve this, you may run the following command:

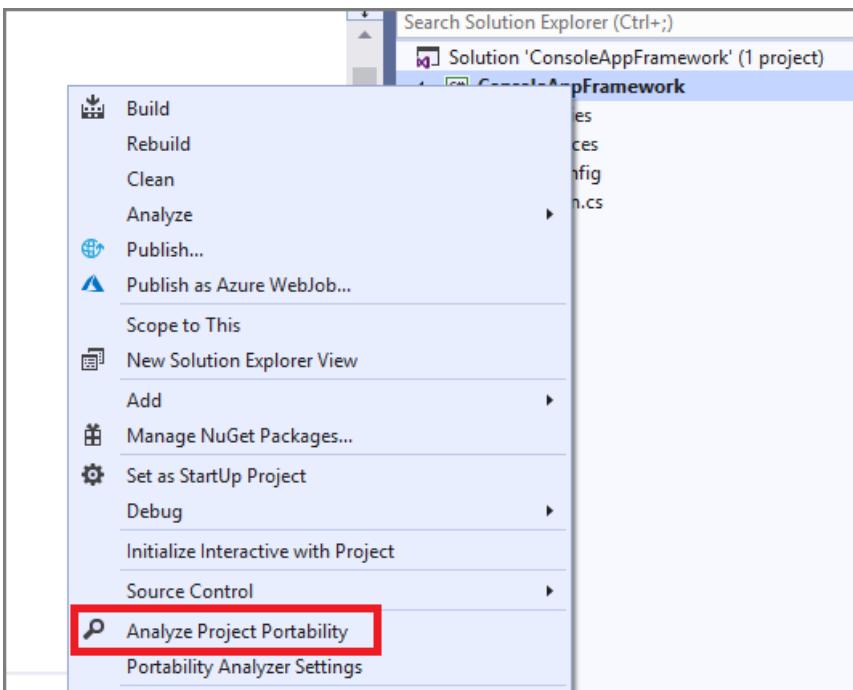
```
ApiPort.exe analyze -r DGML -f [directory or file]
```

A result of this would look like the following when opened in Visual Studio:



Analyze portability

To analyze your entire project in Visual Studio, right-click on your project in Solution Explorer and select **Analyze Assembly Portability**. Otherwise, go to the Analyze menu and select **Analyze Assembly Portability**. From there, select your project's executable or DLL.



You can also use the [ApiPort console app](#).

Type the following command to analyze the current directory:

```
ApiPort.exe analyze -f .
```

To analyze a specific list of .dll files, type the following command:

```
ApiPort.exe analyze -f first.dll -f second.dll -f third.dll
```

To target a specific version, use the `-t` parameter:

```
ApiPort.exe analyze -t ".NET, Version=5.0" -f .
```

Run `ApiPort.exe -?` to get more help.

It is recommended that you include all the related exe and dll files that you own and want to port, and exclude the files that your app depends on, but you don't own and can't port. This will give you most relevant portability report.

View and interpret portability result

Only APIs that are unsupported by a Target Platform appear in the report. After running the analysis in Visual Studio, you'll see your .NET Portability report file link pops up. If you used the [ApiPort console app](#), your .NET Portability report is saved as a file in the format you specified. The default is in an Excel file (.x/sx) in your current directory.

Portability Summary

Submission Id	5552b6d0-b7ed-4fa8-9a3c-e3d8099c7ab1					
Description						
Targets	.NET Core + Platform Extensions,.NET Core,.NET Framework,.NET Standard + Platform Extensions					
Assembly Name	Target Framework	.NET Core + Platform	.NET Core	.NET Framework	.NET Standard	
svutil	.NETFramework,Version=v4.5	71.24	71.48	100	71.24	
API Catalog last updated on	Tuesday, March 5, 2019					
See ' http://go.microsoft.com/fwlink/?LinkId=397652 ' to learn how to read this table						
	Portability Summary	Details				

The Portability Summary section of the report shows the portability percentage for each assembly included in the run. In the previous example, 71.24% of the .NET Framework APIs used in the `svutil` app are available in .NET Core + Platform Extensions. If you run the .NET Portability Analyzer tool against multiple assemblies, each assembly should have a row in the Portability Summary report.

Details

Target type	Target member	Assembly name	.NET Core + Platform Extensions	Recommended
T:System.AppDomain	M:System.AppDomain.get_SetupInformation	svutil	Not supported	Remove usage.
T:System.AppDomainSetup	I:T:System.AppDomainSetup	svutil	Not supported	Remove usage.
T:System.AppDomainSetup	M:System.AppDomainSetup.get_ConfigurationFile	svutil	Not supported	Remove usage.
T:System.Data.DataSetSchemaImporterExtension	I:T:System.Data.DataSetSchemaImporterExtension	svutil	Not supported	
T:System.Data.Design.TypedDataSetSchemaImporterExtension	T:System.Data.Design.TypedDataSetSchemaImporterExtension	svutil	Not supported	
T:System.Data.Design.TypedDataSetSchemaImporterExtensionFx3	I:T:System.Data.Design.TypedDataSetSchemaImporterExtensionFx3	svutil	Not supported	

The Details section of the report lists the APIs missing from any of the selected Targeted Platforms.

- Target type: the type has missing API from a Target Platform
- Target member: the method is missing from a Target Platform
- Assembly name: the .NET Framework assembly that the missing API lives in.
- Each of the selected Target Platforms is one column, such as ".NET Core": "Not supported" value means the API is not supported on this Target Platform.
- Recommended Changes: the recommended API or technology to change to. Currently, this field is empty or out of date for many APIs. Due to the large number of APIs, we have a significant challenge to keep it up to date. We're looking at alternative solutions to provide helpful information to customers.

Missing Assemblies

Header for assembly name entries	Used By	Reason
Autofac, Version=4.6.2.0, Culture=neutral, PublicKeyToken=17863af14b0044da	ApiPort	Unresolved assembly
System.CommandLine, Version=0.1.0.0, Culture=neutral, PublicKeyToken=adb9793829ddae60	ApiPort	Unresolved assembly

You may find a Missing Assemblies section in your report. This section contains a list of assemblies that are referenced by your analyzed assemblies and were not analyzed. If it's an assembly that you own, include it in the API portability analyzer run so that you can get a detailed, API-level portability report for it. If it's a third-party library, check if there is a newer version that supports your target platform, and consider moving to the newer version. Eventually, the list should include all the third-party assemblies that your app depends on that have a version supporting your target platform.

For more information on the .NET Portability Analyzer, visit the [GitHub documentation](#).

Package validation overview

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cross-platform compatibility has become a mainstream requirement for .NET library authors. However, without validation tooling for these packages, they often don't work well. This is especially problematic for emerging platforms where adoption isn't high enough to warrant special attention by library authors.

Until package validation was introduced, the .NET SDK tooling provided almost no validation that multi-targeted packages were well formed. For example, a package that multi-targets for .NET 6 and .NET Standard 2.0 needs to ensure that code compiled against the .NET Standard 2.0 binary can run against the .NET 6 binary.

You might think that a change is safe and compatible if source consuming that change continues to compile without changes. However, the changes can still cause problems at run time if the consumer wasn't recompiled. For example, adding an optional parameter to a method or changing the value of a constant can cause these kinds of compatibility issues.

Package validation tooling allows library developers to validate that their packages are consistent and well formed. It provides the following checks:

- Validates that there are no breaking changes across versions.
- Validates that the package has the same set of public APIs for all the different runtime-specific implementations.
- Helps developers catch any applicability holes.

Enable package validation

You enable package validation in your .NET project by setting the `EnablePackageValidation` property to `true`.

```
<Project Sdk="Microsoft.NET.Sdk">

<PropertyGroup>
  <TargetFrameworks>netstandard2.0;net6.0</TargetFrameworks>
  <EnablePackageValidation>true</EnablePackageValidation>
</PropertyGroup>

</Project>
```

`EnablePackageValidation` runs a series of checks after the `Pack` task. There are some additional checks that can be run by setting other MSBuild properties.

Validator types

There are three different validators that verify your package as part of the `Pack` task:

- The [Baseline version validator](#) validates your library project against a previously released, stable version of your package.
- The [Compatible runtime validator](#) validates that your runtime-specific implementation assemblies are compatible with each other and with the compile-time assemblies.
- The [Compatible framework validator](#) validates that code compiled against one framework can run against all the others in a multi-targeting package.

Suppress compatibility errors

To suppress compatibility errors for intentional changes, add a *CompatibilitySuppressions.xml* file to your project. You can generate this file automatically by passing `/p:GenerateCompatibilitySuppressionFile=true` if you're building the project from the command line, or by adding the following property to your project:

```
<GenerateCompatibilitySuppressionFile>true</GenerateCompatibilitySuppressionFile>
```

The suppression file looks like this.

```
<?xml version="1.0" encoding="utf-8"?>
<Suppressions xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Suppression>
    <DiagnosticId>CP0002</DiagnosticId>
    <Target>M:A.B.DoStringManipulation(System.String)</Target>
    <Left>lib/netstandard2.0/A.dll</Left>
    <Right>lib/net6.0/A.dll</Right>
    <IsBaselineSuppression>false</IsBaselineSuppression>
  </Suppression>
</Suppressions>
```

- `DiagnosticId` specifies the ID of the error to suppress.
- `Target` specifies where in the code to suppress the diagnostic IDs
- `Left` specifies the left operand of an APICompat comparison.
- `Right` specifies the right operand of an APICompat comparison.
- `IsBaselineSuppression` : set to `true` to apply the suppression to a baseline validation; otherwise, set to `false`.

Validate against a baseline package version

9/20/2022 • 2 minutes to read • [Edit Online](#)

Package Validation can help you validate your library project against a previously released, stable version of your package. To enable package validation, add the `PackageValidationBaselineVersion` or `PackageValidationBaselinePath` property to your project file.

Package validation detects any breaking changes on any of the shipped target frameworks. It also detects if any target framework support has been dropped.

For example, consider the following scenario. You're working on the AdventureWorks.Client NuGet package and you want to make sure that you don't accidentally make breaking changes. You configure your project to instruct package validation tooling to run API compatibility checks against the previous version of the package.

```
<Project Sdk="Microsoft.NET.Sdk">

<PropertyGroup>
    <TargetFramework>net6.0</TargetFramework>
    <PackageVersion>2.0.0</PackageVersion>
    <EnablePackageValidation>true</EnablePackageValidation>
    <PackageValidationBaselineVersion>1.0.0</PackageValidationBaselineVersion>
</PropertyGroup>

</Project>
```

A few weeks later, you're tasked with adding support for a connection timeout to your library. The `Connect` method currently looks like this:

```
public static HttpClient Connect(string url)
{
    // ...
}
```

Since a connection timeout is an advanced configuration setting, you reckon that you can just add an optional parameter:

```
public static HttpClient Connect(string url, TimeSpan timeout = default)
{
    // ...
}
```

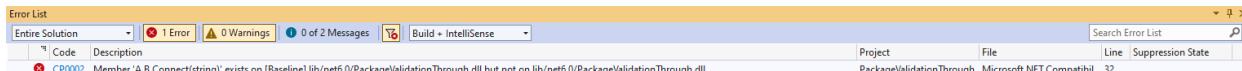
However, when you try to pack, it throws an error.

```

D:\demo>dotnet pack
Microsoft (R) Build Engine version 17.0.0-preview-21460-01+8f208e609 for .NET
Copyright (C) Microsoft Corporation. All rights reserved.

Determining projects to restore...
All projects are up-to-date for restore.
You are using a preview version of .NET. See: https://aka.ms/dotnet-core-preview
PackageValidationThrough -> D:\demo\bin\Debug\net6.0\PackageValidationThrough.dll
Successfully created package 'D:\demo\bin\Debug\PackageValidationThrough.2.0.0.nupkg'.
C:\Program Files\dotnet\sdk\6.0.100-
rc.1.21463.6\Sdks\Microsoft.NET.Sdk\targets\Microsoft.NET.Compatibility.Common.targets(32,5): error CP0002:
Member 'A.B.Connect(string)' exists on [Baseline] lib/net6.0/PackageValidationThrough.dll but not on
lib/net6.0/PackageValidationThrough.dll [D:\demo\PackageValidationThrough.csproj]

```



You realize that while this is not a [source breaking change](#), it is a [binary breaking change](#). You solve this problem by adding a new overload instead of adding a parameter to the existing method:

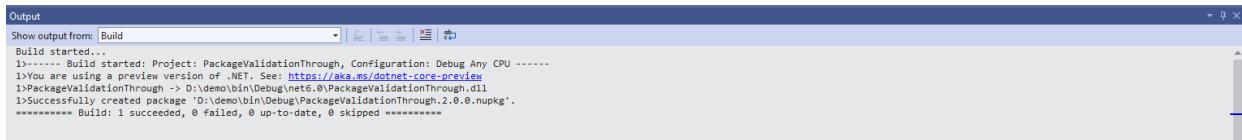
```

public static HttpClient Connect(string url)
{
    return Connect(url, Timeout.InfiniteTimeSpan);
}

public static HttpClient Connect(string url, TimeSpan timeout)
{
    // ...
}

```

Now when you pack the project, it succeeds.



Validate compatible frameworks

9/20/2022 • 2 minutes to read • [Edit Online](#)

Packages containing compatible frameworks need to ensure that code compiled against one can run against another. Examples of compatible framework pairs are:

- .NET Standard 2.0 and .NET 6
- .NET 5 and .NET 6

In both of these cases, consumers can build against .NET Standard 2.0 or NET 5 and run on .NET 6. If your binaries are not compatible between these frameworks, consumers could end up with compile-time or run-time errors.

Package validation catches these errors at pack time. Here is an example scenario:

Suppose you're writing a game that manipulates strings. You need to support both .NET Framework and .NET (.NET Core) consumers. Originally, your project targeted .NET Standard 2.0, but now you want to take advantage of `Span<T>` in .NET 6 to avoid unnecessary string allocations. To do that, you want to multi-target for .NET Standard 2.0 and .NET 6.

You've written the following code:

```
#if NET6_0_OR_GREATER
    public void DoStringManipulation(ReadOnlySpan<char> input)
    {
        // use spans to do string operations.
    }
#else
    public void DoStringManipulation(string input)
    {
        // Do some string operations.
    }
#endif
```

You then try to pack the project (using either `dotnet pack` or Visual Studio), and it fails with the following error:

```
D:\demo>dotnet pack
Microsoft (R) Build Engine version 17.0.0-preview-21460-01+8f208e609 for .NET
Copyright (C) Microsoft Corporation. All rights reserved.

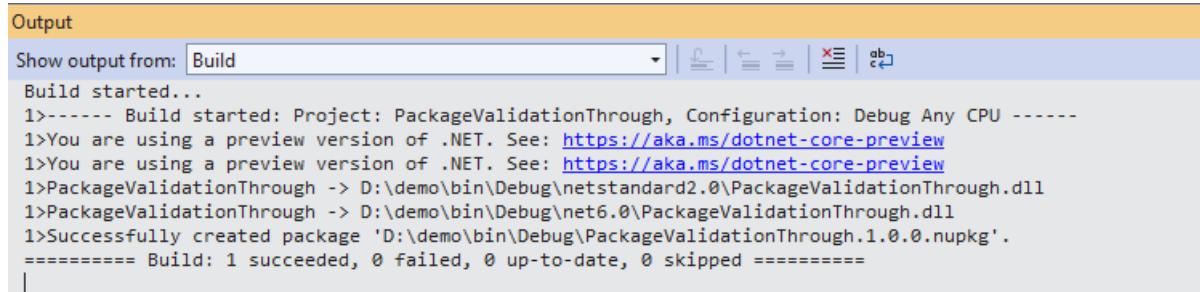
Determining projects to restore...
All projects are up-to-date for restore.
You are using a preview version of .NET. See: https://aka.ms/dotnet-core-preview
You are using a preview version of .NET. See: https://aka.ms/dotnet-core-preview
PackageValidationThrough -> D:\demo\bin\Debug\netstandard2.0\PackageValidationThrough.dll
PackageValidationThrough -> D:\demo\bin\Debug\net6.0\PackageValidationThrough.dll
Successfully created package 'D:\demo\bin\Debug\PackageValidationThrough.1.0.0.nupkg'.
C:\Program Files\dotnet\sdk\6.0.100-
rc.1.21463.6\Sdks\Microsoft.NET.Sdk\targets\Microsoft.NET.Compatibility.Common.targets(32,5): error CP0002:
Member 'A.B.DoStringManipulation(string)' exists on lib/netstandard2.0/PackageValidationThrough.dll but not
on lib/net6.0/PackageValidationThrough.dll [D:\demo\PackageValidationThrough.csproj]
```



You realize that instead of excluding `DoStringManipulation(string)` for .NET 6, you should just provide an additional `DoStringManipulation(ReadOnlySpan<char>)` method for .NET 6:

```
#if NET6_0_OR_GREATER
    public void DoStringManipulation(ReadOnlySpan<char> input)
    {
        // use spans to do string operations.
    }
#endif
    public void DoStringManipulation(string input)
    {
        // Do some string operations.
    }
```

You try to pack the project again, and it succeeds.



The screenshot shows the Visual Studio Output window with the title 'Output' at the top. A dropdown menu 'Show output from:' is set to 'Build'. Below the dropdown are several icons for filtering and sorting the output. The main area contains the following text:

```
Build started...
1>----- Build started: Project: PackageValidationThrough, Configuration: Debug Any CPU -----
1>You are using a preview version of .NET. See: https://aka.ms/dotnet-core-preview
1>You are using a preview version of .NET. See: https://aka.ms/dotnet-core-preview
1>PackageValidationThrough -> D:\demo\bin\Debug\netstandard2.0\PackageValidationThrough.dll
1>PackageValidationThrough -> D:\demo\bin\Debug\net6.0\PackageValidationThrough.dll
1>Successfully created package 'D:\demo\bin\Debug\PackageValidationThrough.1.0.0.nupkg'.
===== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped =====
```

You can enable *strict mode* for this validator by setting the `EnableStrictModeForCompatibleFrameworksInPackage` property in your project file. Enabling strict mode changes some rules and executes some other rules when getting the differences. This is useful when you want both sides you're comparing to be strictly the same on their surface area and identity.

Validate packages against different runtimes

9/20/2022 • 2 minutes to read • [Edit Online](#)

You may choose to have different implementation assemblies for different runtimes in your NuGet package. In that case, you'll need to make sure that these assemblies are compatible with each other and with the compile-time assemblies.

For example, consider the following scenario. You're working on a library involving some interop calls to Unix and Windows APIs, respectively. You have written the following code:

```
#if Unix
    public static void Open(string path, bool securityDescriptor)
    {
        // call unix specific stuff
    }
#else
    public static void Open(string path)
    {
        // call windows specific stuff
    }
#endif
```

The resulting package structure looks as follows.

```
lib/net6.0/A.dll
runtimes/unix/lib/net6.0/A.dll
```

`lib\net6.0\A.dll` will always be used at compile time, regardless of the underlying operating system.

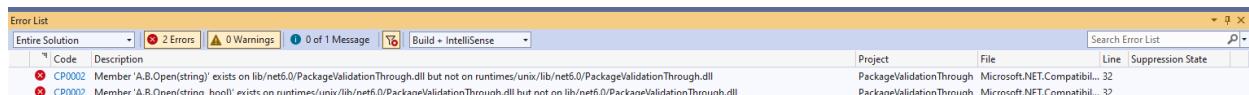
`lib\net6.0\A.dll` will also be used at run time for non-Unix systems. However, `runtimes\unix\lib\net6.0\A.dll` will be used at run time for Unix systems.

When you try to pack this project, you'll get the following error:

```
D:\demo>dotnet pack
Microsoft (R) Build Engine version 17.0.0-preview-21460-01+8f208e609 for .NET
Copyright (C) Microsoft Corporation. All rights reserved.

Determining projects to restore...
All projects are up-to-date for restore.
You are using a preview version of .NET. See: https://aka.ms/dotnet-core-preview
PackageValidationThrough -> D:\demo\bin\Debug\net6.0\PackageValidationThrough.dll
Successfully created package 'D:\demo\bin\Debug\PackageValidationThrough.1.0.0.nupkg'.

C:\Program Files\dotnet\sdk\6.0.100-
rc.1.21463.6\Sdks\Microsoft.NET.Sdk\targets\Microsoft.NET.Compatibility.Common.targets(32,5): error CP0002:
Member 'A.B.Open(string)' exists on lib/net6.0/PackageValidationThrough.dll but not on
runtimes/unix/lib/net6.0/PackageValidationThrough.dll [D:\demo\PackageValidationThrough.csproj]
C:\Program Files\dotnet\sdk\6.0.100-
rc.1.21463.6\Sdks\Microsoft.NET.Sdk\targets\Microsoft.NET.Compatibility.Common.targets(32,5): error CP0002:
Member 'A.B.Open(string, bool)' exists on runtimes/unix/lib/net6.0/PackageValidationThrough.dll but not on
lib/net6.0/PackageValidationThrough.dll [D:\demo\PackageValidationThrough.csproj]
```



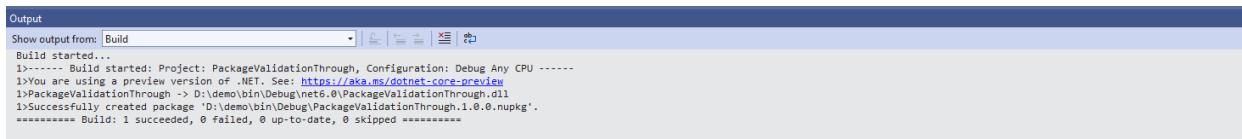
You realize your mistake and add `A.B.Open(string)` to the Unix runtime as well.

```
#if Unix
    public static void Open(string path, bool securityDescriptor)
    {
        // call unix specific stuff
    }

    public static void Open(string path)
    {
        throw new PlatformNotSupportedException();
    }
#else
    public static void Open(string path)
    {
        // call windows specific stuff
    }

    public static void Open(string path, bool securityDescriptor)
    {
        throw new PlatformNotSupportedException();
    }
#endif
```

You try to pack the project again, and it succeeds.



You can enable *strict mode* for this validator by setting the `EnableStrictModeForCompatibleTfms` property in your project file. Enabling strict mode changes some rules, and some other rules will be executed when getting the differences. This is useful when you want both sides we are comparing to be strictly the same on their surface area and identity.

Error codes returned by package validation

9/20/2022 • 3 minutes to read • [Edit Online](#)

This reference article lists all the error codes generated by package validation.

List of error codes

DIAGNOSTIC ID	DESCRIPTION	RECOMMENDED ACTION
PKV0001	A compile-time asset for a compatible framework is missing.	Add the appropriate target framework to the project.
PKV0002	A run-time asset for a compatible framework and runtime is missing.	Add the appropriate asset for corresponding runtime to the package.
PKV0003	A run-time independent asset for a compatible framework is missing.	Add the appropriate run-time independent target framework to the project.
PKV0004	A compatible run-time asset for a compile-time asset is missing.	Add the appropriate run-time asset to the package.
PKV0005	A compatible run-time asset for a compile-time asset and a supported runtime identifier is missing.	Add the appropriate run-time asset to the package.
PKV0006	The target framework is dropped in the latest version.	Add the appropriate target framework to the project.
PKV0007	The target framework and runtime identifier pair is dropped in the latest version.	Add the appropriate target framework and RID to the project.
CP0001	A type, enum, record, or struct visible outside the assembly is missing in the compared assembly when required to be present.	Add the missing type to the assembly where it is missing.
CP0002	A member that is visible outside of the assembly is missing in the compared assembly when required to be present.	Add the missing member to the assembly where it is missing.
CP0003	Some part of the assembly identity (name, public key token, culture, retargetable attribute, or version) does not match on both sides of the comparison.	Update the assembly identity so that both sides match.
CP0004	A matching assembly was not found on one side of the comparison when creating the assembly mapping.	Make sure the missing assembly is added to the package.

DIAGNOSTIC ID	DESCRIPTION	RECOMMENDED ACTION
CP0005	An <code>abstract</code> member was added to the right side of the comparison to an unsealed type.	Remove the member or don't annotate it as <code>abstract</code> .
CP0006	A member was added to an interface without a default implementation.	If the target framework and language version support default implementations, add one, or just remove the member from the interface.
CP0007	A base type on the class hierarchy was removed from one of the compared sides.	Add the base type back. (A new base type can be introduced in the hierarchy if that's intended.)
CP0008	A base interface was removed from the interface hierarchy from one of the compared sides.	Add the interface back to the hierarchy.
CP0009	A type that was unsealed on one side was annotated as <code>sealed</code> on the other compared side.	Remove the <code>sealed</code> annotation from the type.
CP0010	The underlying type of an enum changed from one side to the other.	Change the underlying type back to what it was previously.
CP0011	The value of a member in an enum changed from one side to the other.	Change the value of the member back to what it was previously.
CP0012	The <code>virtual</code> keyword was removed from a member that was previously virtual.	Add the <code>virtual</code> keyword back to the member.
CP0013	The <code>virtual</code> keyword was added to a member that was previously not virtual.	Remove the <code>virtual</code> keyword from the member.
CP0014	An attribute was removed from a member that previously had it.	Add the attribute back to the member.
CP0015	The arguments passed to an attribute changed from one side to the other.	Change the arguments to the attribute back to what they were previously.
CP0016	An attribute was added to a member that previously did not have it.	Remove the attribute from the member.
CP0017	The name of a method's parameter changed from one side to the other.	Change the parameter's name back to what it was previously.
CP0018	The <code>sealed</code> keyword was added to an interface member that was previously not sealed.	Remove the <code>sealed</code> keyword from the interface member.

DIAGNOSTIC ID	DESCRIPTION	RECOMMENDED ACTION
CP1001	A matching assembly could not be found in the search directories. (Not applicable for package validation, only when using API Compat directly.)	Provide the search directory when loading matching assemblies using <code>AssemblySymbolLoader</code> .
CP1002	A reference assembly was not found when loading the assemblies to compare in the resolved directories for the current target framework.	Include the directory path where that assembly can be found using the following MSBuild item: <pre><PackageValidationReferencePath Include=<path> TargetFramework=<tfm> /></pre>
CP1003	There was no search directory provided for the target framework moniker that the package validation is running API Compat for.	Provide the search directory to find references for that target framework using the following MSBuild item: <pre><PackageValidationReferencePath Include=<path> TargetFramework=<tfm> /></pre>

Common Language Runtime (CLR) overview

9/20/2022 • 4 minutes to read • [Edit Online](#)

.NET provides a run-time environment called the common language runtime that runs the code and provides services that make the development process easier.

Compilers and tools expose the common language runtime's functionality and enable you to write code that benefits from the managed execution environment. Code that you develop with a language compiler that targets the runtime is called managed code. Managed code benefits from features such as cross-language integration, cross-language exception handling, enhanced security, versioning and deployment support, a simplified model for component interaction, and debugging and profiling services.

NOTE

Compilers and tools can produce output that the common language runtime can consume because the type system, the format of metadata, and the run-time environment (the virtual execution system) are all defined by a public standard, the ECMA Common Language Infrastructure specification. For more information, see [ECMA C# and Common Language Infrastructure Specifications](#).

To enable the runtime to provide services to managed code, language compilers must emit metadata that describes the types, members, and references in your code. Metadata is stored with the code; every loadable common language runtime portable executable (PE) file contains metadata. The runtime uses metadata to locate and load classes, lay out instances in memory, resolve method invocations, generate native code, enforce security, and set run-time context boundaries.

The runtime automatically handles object layout and manages references to objects, releasing them when they're no longer being used. Objects whose lifetimes are managed in this way are called managed data. Garbage collection eliminates memory leaks and some other common programming errors. If your code is managed, you can use managed, unmanaged, or both managed and unmanaged data in your .NET application. Because language compilers supply their own types, such as primitive types, you might not always know or need to know whether your data is being managed.

The common language runtime makes it easy to design components and applications whose objects interact across languages. Objects written in different languages can communicate with each other, and their behaviors can be tightly integrated. For example, you can define a class and then use a different language to derive a class from your original class or call a method on the original class. You can also pass an instance of a class to a method of a class written in a different language. This cross-language integration is possible because language compilers and tools that target the runtime use a common type system defined by the runtime. They follow the runtime's rules for defining new types and for creating, using, persisting, and binding to types.

As part of their metadata, all managed components carry information about the components and resources they were built against. The runtime uses this information to ensure that your component or application has the specified versions of everything it needs, which makes your code less likely to break because of some unmet dependency. Registration information and state data are no longer stored in the registry, where they can be difficult to establish and maintain. Instead, information about the types you define and their dependencies is stored with the code as metadata. This way, the task of component replication and removal is less complicated.

Language compilers and tools expose the runtime's functionality in ways that are intended to be useful and intuitive to developers. Some features of the runtime might be more noticeable in one environment than in another. How you experience the runtime depends on which language compilers or tools you use. For example, if you're a Visual Basic developer, you might notice that with the common language runtime, the Visual Basic

language has more object-oriented features than before. The runtime provides the following benefits:

- Performance improvements.
- The ability to easily use components developed in other languages.
- Extensible types provided by a class library.
- Language features such as inheritance, interfaces, and overloading for object-oriented programming.
- Support for explicit free threading that allows creation of multithreaded and scalable applications.
- Support for structured exception handling.
- Support for custom attributes.
- Garbage collection.
- Use of delegates instead of function pointers for increased type safety and security. For more information about delegates, see [Common Type System](#).

CLR versions

.NET Core and .NET 5+ releases have a single product version, that is, there's no separate CLR version. For a list of .NET Core versions, see [Download .NET Core](#).

However, the .NET Framework version number doesn't necessarily correspond to the version number of the CLR it includes. For a list of .NET Framework versions and their corresponding CLR versions, see [.NET Framework versions and dependencies](#).

Related articles

TITLE	DESCRIPTION
Managed Execution Process	Describes the steps required to take advantage of the common language runtime.
Automatic Memory Management	Describes how the garbage collector allocates and releases memory.
Overview of .NET Framework	Describes key .NET Framework concepts, such as the common type system, cross-language interoperability, managed execution, application domains, and assemblies.
Common Type System	Describes how types are declared, used, and managed in the runtime in support of cross-language integration.

Managed Execution Process

9/20/2022 • 7 minutes to read • [Edit Online](#)

The managed execution process includes the following steps, which are discussed in detail later in this topic:

1. [Choosing a compiler.](#)

To obtain the benefits provided by the common language runtime, you must use one or more language compilers that target the runtime.

2. [Compiling your code to MSIL.](#)

Compiling translates your source code into Microsoft intermediate language (MSIL) and generates the required metadata.

3. [Compiling MSIL to native code.](#)

At execution time, a just-in-time (JIT) compiler translates the MSIL into native code. During this compilation, code must pass a verification process that examines the MSIL and metadata to find out whether the code can be determined to be type safe.

4. [Running code.](#)

The common language runtime provides the infrastructure that enables execution to take place and services that can be used during execution.

Choosing a Compiler

To obtain the benefits provided by the common language runtime (CLR), you must use one or more language compilers that target the runtime, such as Visual Basic, C#, Visual C++, F#, or one of many third-party compilers such as an Eiffel, Perl, or COBOL compiler.

Because it is a multilanguage execution environment, the runtime supports a wide variety of data types and language features. The language compiler you use determines which runtime features are available, and you design your code using those features. Your compiler, not the runtime, establishes the syntax your code must use. If your component must be completely usable by components written in other languages, your component's exported types must expose only language features that are included in the Common Language Specification (CLS). You can use the [CLSCCompliantAttribute](#) attribute to ensure that your code is CLS-compliant. For more information, see [Language independence and language-independent components](#).

[Back to top](#)

Compiling to MSIL

When compiling to managed code, the compiler translates your source code into Microsoft intermediate language (MSIL), which is a CPU-independent set of instructions that can be efficiently converted to native code. MSIL includes instructions for loading, storing, initializing, and calling methods on objects, as well as instructions for arithmetic and logical operations, control flow, direct memory access, exception handling, and other operations. Before code can be run, MSIL must be converted to CPU-specific code, usually by a [just-in-time \(JIT\) compiler](#). Because the common language runtime supplies one or more JIT compilers for each computer architecture it supports, the same set of MSIL can be JIT-compiled and run on any supported architecture.

When a compiler produces MSIL, it also produces metadata. Metadata describes the types in your code, including the definition of each type, the signatures of each type's members, the members that your code references, and other data that the runtime uses at execution time. The MSIL and metadata are contained in a portable executable (PE) file that is based on and that extends the published Microsoft PE and common object file format (COFF) used historically for executable content. This file format, which accommodates MSIL or native code as well as metadata, enables the operating system to recognize common language runtime images. The presence of metadata in the file together with MSIL enables your code to describe itself, which means that there is no need for type libraries or Interface Definition Language (IDL). The runtime locates and extracts the metadata from the file as needed during execution.

[Back to top](#)

Compiling MSIL to Native Code

Before you can run Microsoft intermediate language (MSIL), it must be compiled against the common language runtime to native code for the target machine architecture. .NET provides two ways to perform this conversion:

- A .NET just-in-time (JIT) compiler.
- [Ngen.exe \(Native Image Generator\)](#).

Compilation by the JIT Compiler

JIT compilation converts MSIL to native code on demand at application run time, when the contents of an assembly are loaded and executed. Because the common language runtime supplies a JIT compiler for each supported CPU architecture, developers can build a set of MSIL assemblies that can be JIT-compiled and run on different computers with different machine architectures. However, if your managed code calls platform-specific native APIs or a platform-specific class library, it will run only on that operating system.

JIT compilation takes into account the possibility that some code might never be called during execution. Instead of using time and memory to convert all the MSIL in a PE file to native code, it converts the MSIL as needed during execution and stores the resulting native code in memory so that it is accessible for subsequent calls in the context of that process. The loader creates and attaches a stub to each method in a type when the type is loaded and initialized. When a method is called for the first time, the stub passes control to the JIT compiler, which converts the MSIL for that method into native code and modifies the stub to point directly to the generated native code. Therefore, subsequent calls to the JIT-compiled method go directly to the native code.

Install-Time Code Generation Using NGen.exe

Because the JIT compiler converts an assembly's MSIL to native code when individual methods defined in that assembly are called, it affects performance adversely at run time. In most cases, that diminished performance is acceptable. More importantly, the code generated by the JIT compiler is bound to the process that triggered the compilation. It cannot be shared across multiple processes. To allow the generated code to be shared across multiple invocations of an application or across multiple processes that share a set of assemblies, the common language runtime supports an ahead-of-time compilation mode. This ahead-of-time compilation mode uses the [Ngen.exe \(Native Image Generator\)](#) to convert MSIL assemblies to native code much like the JIT compiler does. However, the operation of Ngen.exe differs from that of the JIT compiler in three ways:

- It performs the conversion from MSIL to native code before running the application instead of while the application is running.
- It compiles an entire assembly at a time, instead of one method at a time.
- It persists the generated code in the Native Image Cache as a file on disk.

Code Verification

As part of its compilation to native code, the MSIL code must pass a verification process unless an administrator

has established a security policy that allows the code to bypass verification. Verification examines MSIL and metadata to find out whether the code is type safe, which means that it accesses only the memory locations it is authorized to access. Type safety helps isolate objects from each other and helps protect them from inadvertent or malicious corruption. It also provides assurance that security restrictions on code can be reliably enforced.

The runtime relies on the fact that the following statements are true for code that is verifiably type safe:

- A reference to a type is strictly compatible with the type being referenced.
- Only appropriately defined operations are invoked on an object.
- Identities are what they claim to be.

During the verification process, MSIL code is examined in an attempt to confirm that the code can access memory locations and call methods only through properly defined types. For example, code cannot allow an object's fields to be accessed in a manner that allows memory locations to be overrun. Additionally, verification inspects code to determine whether the MSIL has been correctly generated, because incorrect MSIL can lead to a violation of the type safety rules. The verification process passes a well-defined set of type-safe code, and it passes only code that is type safe. However, some type-safe code might not pass verification because of some limitations of the verification process, and some languages, by design, do not produce verifiably type-safe code. If type-safe code is required by the security policy but the code does not pass verification, an exception is thrown when the code is run.

[Back to top](#)

Running Code

The common language runtime provides the infrastructure that enables managed execution to take place and services that can be used during execution. Before a method can be run, it must be compiled to processor-specific code. Each method for which MSIL has been generated is JIT-compiled when it is called for the first time, and then run. The next time the method is run, the existing JIT-compiled native code is run. The process of JIT-compiling and then running the code is repeated until execution is complete.

During execution, managed code receives services such as garbage collection, security, interoperability with unmanaged code, cross-language debugging support, and enhanced deployment and versioning support.

In Microsoft Windows Vista, the operating system loader checks for managed modules by examining a bit in the COFF header. The bit being set denotes a managed module. If the loader detects managed modules, it loads mscoree.dll, and `_CorValidateImage` and `_CorImageUnloading` notify the loader when the managed module images are loaded and unloaded. `_CorValidateImage` performs the following actions:

1. Ensures that the code is valid managed code.
2. Changes the entry point in the image to an entry point in the runtime.

On 64-bit Windows, `_CorValidateImage` modifies the image that is in memory by transforming it from PE32 to PE32+ format.

[Back to top](#)

See also

- [Overview](#)
- [Language independence and language-independent components](#)
- [Metadata and Self-Describing Components](#)
- [Ilasm.exe \(IL Assembler\)](#)

- Security
- Interoperating with Unmanaged Code
- Deployment
- Assemblies in .NET
- Application Domains

Assemblies in .NET

9/20/2022 • 6 minutes to read • [Edit Online](#)

Assemblies are the fundamental units of deployment, version control, reuse, activation scoping, and security permissions for .NET-based applications. An assembly is a collection of types and resources that are built to work together and form a logical unit of functionality. Assemblies take the form of executable (.exe) or dynamic link library (.dll) files, and are the building blocks of .NET applications. They provide the common language runtime with the information it needs to be aware of type implementations.

In .NET and .NET Framework, you can build an assembly from one or more source code files. In .NET Framework, assemblies can contain one or more modules. This way, larger projects can be planned so that several developers can work on separate source code files or modules, which are combined to create a single assembly. For more information about modules, see [How to: Build a multifile assembly](#).

Assemblies have the following properties:

- Assemblies are implemented as .exe or .dll files.
- For libraries that target .NET Framework, you can share assemblies between applications by putting them in the [global assembly cache \(GAC\)](#). You must strong-name assemblies before you can include them in the GAC. For more information, see [Strong-named assemblies](#).
- Assemblies are only loaded into memory if they're required. If they aren't used, they aren't loaded. Therefore, assemblies can be an efficient way to manage resources in larger projects.
- You can programmatically obtain information about an assembly by using reflection. For more information, see [Reflection \(C#\)](#) or [Reflection \(Visual Basic\)](#).
- You can load an assembly just to inspect it by using the [MetadataLoadContext](#) class on .NET and .NET Framework. [MetadataLoadContext](#) replaces the [Assembly.ReflectionOnlyLoad](#) methods.

Assemblies in the common language runtime

Assemblies provide the common language runtime with the information it needs to be aware of type implementations. To the runtime, a type doesn't exist outside the context of an assembly.

An assembly defines the following information:

- **Code** that the common language runtime executes. Each assembly can have only one entry point: `DllMain`, `WinMain`, or `Main`.
- The **security boundary**. An assembly is the unit at which permissions are requested and granted. For more information about security boundaries in assemblies, see [Assembly security considerations](#).
- The **type boundary**. Every type's identity includes the name of the assembly in which it resides. A type called `MyType` that's loaded in the scope of one assembly isn't the same as a type called `MyType` that's loaded in the scope of another assembly.
- The **reference-scope boundary**: The [assembly manifest](#) has metadata that's used for resolving types and satisfying resource requests. The manifest specifies the types and resources to expose outside the assembly and enumerates other assemblies on which it depends. Microsoft intermediate language (MSIL) code in a portable executable (PE) file won't be executed unless it has an associated [assembly manifest](#).
- The **version boundary**. The assembly is the smallest versionable unit in the common language runtime.

All types and resources in the same assembly are versioned as a unit. The [assembly manifest](#) describes the version dependencies you specify for any dependent assemblies. For more information about versioning, see [Assembly versioning](#).

- The **deployment unit**: When an application starts, only the assemblies that the application initially calls must be present. Other assemblies, such as assemblies containing localization resources or utility classes, can be retrieved on demand. This process allows apps to be simple and thin when first downloaded. For more information about deploying assemblies, see [Deploy applications](#).
- A **side-by-side execution unit**: For more information about running multiple versions of an assembly, see [Assemblies and side-by-side execution](#).

Create an assembly

Assemblies can be static or dynamic. Static assemblies are stored on a disk in portable executable (PE) files. Static assemblies can include interfaces, classes, and resources like bitmaps, JPEG files, and other resource files. You can also create dynamic assemblies, which are run directly from memory and aren't saved to disk before execution. You can save dynamic assemblies to disk after they've been executed.

There are several ways to create assemblies. You can use development tools, such as Visual Studio that can create `.dll` or `.exe` files. You can use tools in the Windows SDK to create assemblies with modules from other development environments. You can also use common language runtime APIs, such as [System.Reflection.Emit](#), to create dynamic assemblies.

Compile assemblies by building them in Visual Studio, building them with .NET Core command-line interface tools, or building .NET Framework assemblies with a command-line compiler. For more information about building assemblies using .NET CLI, see [.NET CLI overview](#).

NOTE

To build an assembly in Visual Studio, on the **Build** menu, select **Build**.

Assembly manifest

Every assembly has an *assembly manifest* file. Similar to a table of contents, the assembly manifest contains:

- The assembly's identity (its name and version).
- A file table describing all the other files that make up the assembly, such as other assemblies you created that your `.exe` or `.dll` file relies on, bitmap files, or Readme files.
- An *assembly reference list*, which is a list of all external dependencies, such as `.dlls` or other files. Assembly references contain references to both global and private objects. Global objects are available to all other applications. In .NET Core, global objects are coupled with a particular .NET Core runtime. In .NET Framework, global objects reside in the global assembly cache (GAC). `System.IO.dll` is an example of an assembly in the GAC. Private objects must be in a directory level at or below the directory in which your app is installed.

Assemblies contain information about content, versioning, and dependencies. So the applications that use them don't need to rely on external sources, such as the registry on Windows systems, to function properly.

Assemblies reduce `.dll` conflicts and make your applications more reliable and easier to deploy. In many cases, you can install a .NET-based application simply by copying its files to the target computer. For more information, see [Assembly manifest](#).

Add a reference to an assembly

To use an assembly in an application, you must add a reference to it. When an assembly is referenced, all the accessible types, properties, methods, and other members of its namespaces are available to your application as if their code were part of your source file.

NOTE

Most assemblies from the .NET Class Library are referenced automatically. If a system assembly isn't automatically referenced, add a reference in one of the following ways:

- For .NET and .NET Core, add a reference to the NuGet package that contains the assembly. Either use the NuGet Package Manager in Visual Studio or add a `<PackageReference>` element for the assembly to the `.csproj` or `.vbproj` project.
- For .NET Framework, add a reference to the assembly by using the **Add Reference** dialog in Visual Studio or the `-reference` command line option for the [C#](#) or [Visual Basic](#) compilers.

In C#, you can use two versions of the same assembly in a single application. For more information, see [extern alias](#).

Related content

TITLE	DESCRIPTION
Assembly contents	Elements that make up an assembly.
Assembly manifest	Data in the assembly manifest, and how it's stored in assemblies.
Global assembly cache	How the GAC stores and uses assemblies.
Strong-named assemblies	Characteristics of strong-named assemblies.
Assembly security considerations	How security works with assemblies.
Assembly versioning	Overview of the .NET Framework versioning policy.
Assembly placement	Where to locate assemblies.
Assemblies and side-by-side execution	Use multiple versions of the runtime or an assembly simultaneously.
Emit dynamic methods and assemblies	How to create dynamic assemblies.
How the runtime locates assemblies	How the .NET Framework resolves assembly references at run time.

Reference

[System.Reflection.Assembly](#)

See also

- [.NET assembly file format](#)
- [Friend assemblies](#)

- [Reference assemblies](#)
- [How to: Load and unload assemblies](#)
- [How to: Use and debug assembly unloadability in .NET Core](#)
- [How to: Determine if a file is an assembly](#)
- [How to: Inspect assembly contents using MetadataLoadContext](#)

Metadata and Self-Describing Components

9/20/2022 • 8 minutes to read • [Edit Online](#)

In the past, a software component (.exe or .dll) that was written in one language could not easily use a software component that was written in another language. COM provided a step towards solving this problem. .NET makes component interoperability even easier by allowing compilers to emit additional declarative information into all modules and assemblies. This information, called metadata, helps components to interact seamlessly.

Metadata is binary information describing your program that is stored either in a common language runtime portable executable (PE) file or in memory. When you compile your code into a PE file, metadata is inserted into one portion of the file, and your code is converted to Microsoft intermediate language (MSIL) and inserted into another portion of the file. Every type and member that is defined and referenced in a module or assembly is described within metadata. When code is executed, the runtime loads metadata into memory and references it to discover information about your code's classes, members, inheritance, and so on.

Metadata describes every type and member defined in your code in a language-neutral manner. Metadata stores the following information:

- Description of the assembly.
 - Identity (name, version, culture, public key).
 - The types that are exported.
 - Other assemblies that this assembly depends on.
 - Security permissions needed to run.
- Description of types.
 - Name, visibility, base class, and interfaces implemented.
 - Members (methods, fields, properties, events, nested types).
- Attributes.
 - Additional descriptive elements that modify types and members.

Benefits of Metadata

Metadata is the key to a simpler programming model, and eliminates the need for Interface Definition Language (IDL) files, header files, or any external method of component reference. Metadata enables .NET languages to describe themselves automatically in a language-neutral manner, unseen by both the developer and the user. Additionally, metadata is extensible through the use of attributes. Metadata provides the following major benefits:

- Self-describing files.

Common language runtime modules and assemblies are self-describing. A module's metadata contains everything needed to interact with another module. Metadata automatically provides the functionality of IDL in COM, so you can use one file for both definition and implementation. Runtime modules and assemblies do not even require registration with the operating system. As a result, the descriptions used by the runtime always reflect the actual code in your compiled file, which increases application reliability.

- Language interoperability and easier component-based design.

Metadata provides all the information required about compiled code for you to inherit a class from a PE file written in a different language. You can create an instance of any class written in any managed language (any language that targets the common language runtime) without worrying about explicit marshalling or using custom interoperability code.

- Attributes.

.NET lets you declare specific kinds of metadata, called attributes, in your compiled file. Attributes can be found throughout .NET and are used to control in more detail how your program behaves at run time. Additionally, you can emit your own custom metadata into .NET files through user-defined custom attributes. For more information, see [Attributes](#).

Metadata and the PE File Structure

Metadata is stored in one section of a .NET portable executable (PE) file, while Microsoft intermediate language (MSIL) is stored in another section of the PE file. The metadata portion of the file contains a series of table and heap data structures. The MSIL portion contains MSIL and metadata tokens that reference the metadata portion of the PE file. You might encounter metadata tokens when you use tools such as the [MSIL Disassembler \(ildasm.exe\)](#) to view your code's MSIL, for example.

Metadata Tables and Heaps

Each metadata table holds information about the elements of your program. For example, one metadata table describes the classes in your code, another table describes the fields, and so on. If you have ten classes in your code, the class table will have tens rows, one for each class. Metadata tables reference other tables and heaps. For example, the metadata table for classes references the table for methods.

Metadata also stores information in four heap structures: string, blob, user string, and GUID. All the strings used to name types and members are stored in the string heap. For example, a method table does not directly store the name of a particular method, but points to the method's name stored in the string heap.

Metadata Tokens

Each row of each metadata table is uniquely identified in the MSIL portion of the PE file by a metadata token. Metadata tokens are conceptually similar to pointers, persisted in MSIL, that reference a particular metadata table.

A metadata token is a four-byte number. The top byte denotes the metadata table to which a particular token refers (method, type, and so on). The remaining three bytes specify the row in the metadata table that corresponds to the programming element being described. If you define a method in C# and compile it into a PE file, the following metadata token might exist in the MSIL portion of the PE file:

0x06000004

The top byte (0x06) indicates that this is a **MethodDef** token. The lower three bytes (000004) tells the common language runtime to look in the fourth row of the **MethodDef** table for the information that describes this method definition.

Metadata within a PE File

When a program is compiled for the common language runtime, it is converted to a PE file that consists of three parts. The following table describes the contents of each part.

PE SECTION	CONTENTS OF PE SECTION
------------	------------------------

PE SECTION	CONTENTS OF PE SECTION
PE header	The index of the PE file's main sections and the address of the entry point. The runtime uses this information to identify the file as a PE file and to determine where execution starts when loading the program into memory.
MSIL instructions	The Microsoft intermediate language instructions (MSIL) that make up your code. Many MSIL instructions are accompanied by metadata tokens.
Metadata	Metadata tables and heaps. The runtime uses this section to record information about every type and member in your code. This section also includes custom attributes and security information.

Run-Time Use of Metadata

To better understand metadata and its role in the common language runtime, it might be helpful to construct a simple program and illustrate how metadata affects its run-time life. The following code example shows two methods inside a class called `MyApp`. The `Main` method is the program entry point, while the `Add` method simply returns the sum of two integer arguments.

```
Public Class MyApp
    Public Shared Sub Main()
        Dim ValueOne As Integer = 10
        Dim ValueTwo As Integer = 20
        Console.WriteLine("The Value is: {0}", Add(ValueOne, ValueTwo))
    End Sub

    Public Shared Function Add(One As Integer, Two As Integer) As Integer
        Return (One + Two)
    End Function
End Class
```

```
using System;
public class MyApp
{
    public static int Main()
    {
        int ValueOne = 10;
        int ValueTwo = 20;
        Console.WriteLine("The Value is: {0}", Add(ValueOne, ValueTwo));
        return 0;
    }
    public static int Add(int One, int Two)
    {
        return (One + Two);
    }
}
```

When the code runs, the runtime loads the module into memory and consults the metadata for this class. Once loaded, the runtime performs extensive analysis of the method's Microsoft intermediate language (MSIL) stream to convert it to fast native machine instructions. The runtime uses a just-in-time (JIT) compiler to convert the MSIL instructions to native machine code one method at a time as needed.

The following example shows part of the MSIL produced from the previous code's `Main` function. You can view the MSIL and metadata from any .NET application using the [MSIL Disassembler \(Ildasm.exe\)](#).

```
.entrypoint
.maxstack 3
.locals ([0] int32 ValueOne,
          [1] int32 ValueTwo,
          [2] int32 V_2,
          [3] int32 V_3)
IL_0000: ldc.i4.s 10
IL_0002: stloc.0
IL_0003: ldc.i4.s 20
IL_0005: stloc.1
IL_0006: ldstr     "The Value is: {0}"
IL_000b: ldloc.0
IL_000c: ldloc.1
IL_000d: call int32 ConsoleApplication.MyApp::Add(int32,int32) /* 06000003 */
```

The JIT compiler reads the MSIL for the whole method, analyzes it thoroughly, and generates efficient native instructions for the method. At `IL_000d`, a metadata token for the `Add` method (`/* 06000003 */`) is encountered and the runtime uses the token to consult the third row of the **MethodDef** table.

The following table shows part of the **MethodDef** table referenced by the metadata token that describes the `Add` method. While other metadata tables exist in this assembly and have their own unique values, only this table is discussed.

ROW	RELATIVE VIRTUAL ADDRESS (RVA)	IMPLFLAGS	FLAGS	NAME (POINTS TO STRING HEAP.)	SIGNATURE (POINTS TO BLOB HEAP.)
1	0x00002050	IL Managed	Public ReuseSlot SpecialName RTSpecialName .ctor	.ctor (constructor)	
2	0x00002058	IL Managed	Public Static ReuseSlot	Main	String
3	0x0000208c	IL Managed	Public Static ReuseSlot	Add	int, int, int

Each column of the table contains important information about your code. The **RVA** column allows the runtime to calculate the starting memory address of the MSIL that defines this method. The **ImplFlags** and **Flags** columns contain bitmasks that describe the method (for example, whether the method is public or private). The **Name** column indexes the name of the method from the string heap. The **Signature** column indexes the definition of the method's signature in the blob heap.

The runtime calculates the desired offset address from the **RVA** column in the third row and returns this address

to the JIT compiler, which then proceeds to the new address. The JIT compiler continues to process MSIL at the new address until it encounters another metadata token and the process is repeated.

Using metadata, the runtime has access to all the information it needs to load your code and process it into native machine instructions. In this manner, metadata enables self-describing files and, together with the common type system, cross-language inheritance.

Related Topics

TITLE	DESCRIPTION
Attributes	Describes how to apply attributes, write custom attributes, and retrieve information that is stored in attributes.

Dependency loading in .NET

9/20/2022 • 2 minutes to read • [Edit Online](#)

Every .NET application has dependencies. Even the simple `hello world` app has dependencies on portions of the .NET class libraries.

Understanding the default assembly loading logic in .NET can help you troubleshoot typical deployment issues.

In some applications, dependencies are dynamically determined at run time. In these situations, it's critical to understand how managed assemblies and unmanaged dependencies are loaded.

AssemblyLoadContext

The [AssemblyLoadContext](#) API is central to the .NET loading design. The [Understanding AssemblyLoadContext](#) article provides a conceptual overview of the design.

Loading details

The loading algorithm details are covered briefly in several articles:

- [Managed assembly loading algorithm](#)
- [Satellite assembly loading algorithm](#)
- [Unmanaged \(native\) library loading algorithm](#)
- [Default probing](#)

Create an app with plugins

The tutorial [Create a .NET application with plugins](#) describes how to create a custom AssemblyLoadContext. It uses an [AssemblyDependencyResolver](#) to resolve the dependencies of the plugin. The tutorial correctly isolates the plugin's dependencies from the hosting application.

Assembly unloadability

The [How to use and debug assembly unloadability in .NET](#) article is a step-by-step tutorial. It shows how to load a .NET application, execute it, and then unload it. The article also provides debugging tips.

Collect detailed assembly loading information

The [Collect detailed assembly loading information](#) article describes how to collect detailed information about managed assembly loading in the runtime. It uses the [dotnet-trace](#) tool to capture assembly loader events in a trace of a running process.

About System.Runtime.Loader.AssemblyLoadContext

9/20/2022 • 5 minutes to read • [Edit Online](#)

The [AssemblyLoadContext](#) class was introduced in .NET Core and is not available in .NET Framework. This article supplements the [AssemblyLoadContext](#) API documentation with conceptual information.

This article is relevant to developers implementing dynamic loading, especially dynamic-loading framework developers.

What is the AssemblyLoadContext?

Every .NET 5+ and .NET Core application implicitly uses [AssemblyLoadContext](#). It's the runtime's provider for locating and loading dependencies. Whenever a dependency is loaded, an [AssemblyLoadContext](#) instance is invoked to locate it.

- AssemblyLoadContext provides a service of locating, loading, and caching managed assemblies and other dependencies.
- To support dynamic code loading and unloading, it creates an isolated context for loading code and its dependencies in their own [AssemblyLoadContext](#) instance.

Versioning rules

A single [AssemblyLoadContext](#) instance is limited to loading exactly one version of an [Assembly](#) per [simple assembly name](#). When an assembly reference is resolved against an [AssemblyLoadContext](#) instance that already has an assembly of that name loaded, the requested version is compared to the loaded version. The resolution will succeed only if the loaded version is equal or higher to the requested version.

When do you need multiple AssemblyLoadContext instances?

The restriction that a single [AssemblyLoadContext](#) instance can load only one version of an assembly can become a problem when loading code modules dynamically. Each module is independently compiled, and the modules may depend on different versions of an [Assembly](#). This is often a problem when different modules depend on different versions of a commonly used library.

To support dynamically loading code, the [AssemblyLoadContext](#) API provides for loading conflicting versions of an [Assembly](#) in the same application. Each [AssemblyLoadContext](#) instance provides a unique dictionary that maps each [AssemblyName.Name](#) to a specific [Assembly](#) instance.

It also provides a convenient mechanism for grouping dependencies related to a code module for later unload.

The AssemblyLoadContext.Default instance

The [AssemblyLoadContext.Default](#) instance is automatically populated by the runtime at startup. It uses [default probing](#) to locate and find all static dependencies.

It solves the most common dependency loading scenarios.

Dynamic dependencies

[AssemblyLoadContext](#) has various events and virtual functions that can be overridden.

The `AssemblyLoadContext.Default` instance only supports overriding the events.

The articles [Managed assembly loading algorithm](#), [Satellite assembly loading algorithm](#), and [Unmanaged \(native\) library loading algorithm](#) refer to all the available events and virtual functions. The articles show each event and function's relative position in the loading algorithms. This article doesn't reproduce that information.

This section covers the general principles for the relevant events and functions.

- **Be repeatable.** A query for a specific dependency must always result in the same response. The same loaded dependency instance must be returned. This requirement is fundamental for cache consistency. For managed assemblies in particular, we're creating an `Assembly` cache. The cache key is a simple assembly name, `AssemblyName.Name`.
- **Typically don't throw.** It's expected that these functions return `null` rather than throw when unable to find the requested dependency. Throwing will prematurely end the search and propagate an exception to the caller. Throwing should be restricted to unexpected errors like a corrupted assembly or an out of memory condition.
- **Avoid recursion.** Be aware that these functions and handlers implement the loading rules for locating dependencies. Your implementation shouldn't call APIs that trigger recursion. Your code should typically call `AssemblyLoadContext` load functions that require a specific path or memory reference argument.
- **Load into the correct `AssemblyLoadContext`.** The choice of where to load dependencies is application-specific. The choice is implemented by these events and functions. When your code calls `AssemblyLoadContext` load-by-path functions call them on the instance where you want the code loaded. Sometime returning `null` and letting the `AssemblyLoadContext.Default` handle the load may be the simplest option.
- **Be aware of thread races.** Loading can be triggered by multiple threads. The `AssemblyLoadContext` handles thread races by atomically adding assemblies to its cache. The race loser's instance is discarded. In your implementation logic, don't add extra logic that doesn't handle multiple threads properly.

How are dynamic dependencies isolated?

Each `AssemblyLoadContext` instance represents a unique scope for `Assembly` instances and `Type` definitions.

There's no binary isolation between these dependencies. They're only isolated by not finding each other by name.

In each `AssemblyLoadContext`:

- `AssemblyName.Name` may refer to a different `Assembly` instance.
- `Type.GetType` may return a different type instance for the same type `name`.

Shared dependencies

Dependencies can easily be shared between `AssemblyLoadContext` instances. The general model is for one `AssemblyLoadContext` to load a dependency. The other shares the dependency by using a reference to the loaded assembly.

This sharing is required of the runtime assemblies. These assemblies can only be loaded into the `AssemblyLoadContext.Default`. The same is required for frameworks like `ASP.NET`, `WPF`, or `WinForms`.

It's recommended that shared dependencies be loaded into `AssemblyLoadContext.Default`. This sharing is the common design pattern.

Sharing is implemented in the coding of the custom `AssemblyLoadContext` instance. `AssemblyLoadContext` has various events and virtual functions that can be overridden. When any of these functions return a reference to an `Assembly` instance that was loaded in another `AssemblyLoadContext` instance, the `Assembly` instance is

shared. The standard load algorithm defers to [AssemblyLoadContext.Default](#) for loading to simplify the common sharing pattern. For more information, see [Managed assembly loading algorithm](#).

Type-conversion issues

When two [AssemblyLoadContext](#) instances contain type definitions with the same `name`, they're not the same type. They're the same type if and only if they come from the same [Assembly](#) instance.

To complicate matters, exception messages about these mismatched types can be confusing. The types are referred to in the exception messages by their simple type names. The common exception message in this case is of the form:

```
Object of type 'IsolatedType' cannot be converted to type 'IsolatedType'.
```

Debug type-conversion issues

Given a pair of mismatched types, it's important to also know:

- Each type's [Type.Assembly](#).
- Each type's [AssemblyLoadContext](#), which can be obtained via the [AssemblyLoadContext.GetLoadContext\(Assembly\)](#) function.

Given two objects `a` and `b`, evaluating the following in the debugger will be helpful:

```
// In debugger look at each assembly's instance, Location, and FullName  
a.GetType().Assembly  
b.GetType().Assembly  
// In debugger look at each AssemblyLoadContext's instance and name  
System.Runtime.Loader.AssemblyLoadContext.GetLoadContext(a.GetType().Assembly)  
System.Runtime.Loader.AssemblyLoadContext.GetLoadContext(b.GetType().Assembly)
```

Resolve type-conversion issues

There are two design patterns for solving these type conversion issues.

1. Use common shared types. This shared type can either be a primitive runtime type, or it can involve creating a new shared type in a shared assembly. Often the shared type is an [interface](#) defined in an application assembly. For more information, read about [how dependencies are shared](#).
2. Use marshalling techniques to convert from one type to another.

Default probing

9/20/2022 • 3 minutes to read • [Edit Online](#)

The [AssemblyLoadContext.Default](#) instance is responsible for locating an assembly's dependencies. This article describes the [AssemblyLoadContext.Default](#) instance's probing logic.

Host configured probing properties

When the runtime is started, the runtime host provides a set of named probing properties that configure [AssemblyLoadContext.Default](#) probe paths.

Each probing property is optional. If present, each property is a string value that contains a delimited list of absolute paths. The delimiter is ';' on Windows and ':' on all other platforms.

PROPERTY NAME	DESCRIPTION
TRUSTED_PLATFORM_ASSEMBLIES	List of platform and application assembly file paths.
PLATFORM_RESOURCE_ROOTS	List of directory paths to search for satellite resource assemblies.
NATIVE_DLL_SEARCH_DIRECTORIES	List of directory paths to search for unmanaged (native) libraries.
APP_PATHS	List of directory paths to search for managed assemblies.

How are the properties populated?

There are two main scenarios for populating the properties depending on whether the `<myapp>.deps.json` file exists.

- When the `*.deps.json` file is present, it's parsed to populate the probing properties.
- When the `*.deps.json` file isn't present, the application's directory is assumed to contain all the dependencies. The directory's contents are used to populate the probing properties.

Additionally, the `*.deps.json` files for any referenced frameworks are similarly parsed.

The environment variable `DOTNET_ADDITIONAL_DEPS` can be used to add additional dependencies. `dotnet.exe` also contains an optional `--additional-deps` parameter to set this value on application startup.

The `APP_PATHS` property is not populated by default and is omitted for most applications.

The list of all `*.deps.json` files used by the application can be accessed via

```
System.AppContext.GetData("APP_CONTEXT_DEPS_FILES") .
```

How do I see the probing properties from managed code?

Each property is available by calling the [AppContext.GetData\(String\)](#) function with the property name from the table above.

How do I debug the probing properties' construction?

The .NET Core runtime host will output useful trace messages when certain environment variables are enabled:

ENVIRONMENT VARIABLE	DESCRIPTION
COREHOST_TRACE=1	Enables tracing.
COREHOST_TRACEFILE=<path>	Traces to a file path instead of the default <code>stderr</code> .
COREHOST_TRACE_VERBOSITY	Sets the verbosity from 1 (lowest) to 4 (highest).

Managed assembly default probing

When probing to locate a managed assembly, the [AssemblyLoadContext.Default](#) looks in order at:

- Files matching the [AssemblyName.Name](#) in `TRUSTED_PLATFORM_ASSEMBLIES` (after removing file extensions).
- Assembly files in `APP_PATHS` with common file extensions.

Satellite (resource) assembly probing

To find a satellite assembly for a specific culture, construct a set of file paths.

For each path in `PLATFORM_RESOURCE_ROOTS` and then `APP_PATHS`, append the [CultureInfo.Name](#) string, a directory separator, the [AssemblyName.Name](#) string, and the extension '.dll'.

If any matching file exists, attempt to load and return it.

Unmanaged (native) library probing

The runtime's unmanaged library probing algorithm is identical on all platforms. However, since the actual load of the unmanaged library is performed by the underlying platform, the observed behavior can be slightly different. For more information, see the guidance on how to author [cross-platform P/Invokes](#).

1. Check if the supplied library name represents an absolute or relative path.
2. If the name represents an absolute path, use the name directly for all subsequent operations. Otherwise, use the name and create platform-defined combinations to consider. Combinations consist of platform specific prefixes (for example, `lib`) and/or suffixes (for example, `.dll`, `.dylib`, and `.so`). This is not an exhaustive list, and it doesn't represent the exact effort made on each platform. It's just an example of what is considered.
3. The name and, if the path is relative, each combination, is then used in the following steps. The first successful load attempt immediately returns the handle to the loaded library.
 - Append it to each path supplied in the `NATIVE_DLL_SEARCH_DIRECTORIES` property and attempt to load.
 - If `DllImportSearchPath.AssemblyDirectory` is defined on the calling assembly, use it directly and attempt to load relative to the calling assembly.
 - Use it directly to load the library.
4. Indicate that the library failed to load.

Managed assembly loading algorithm

9/20/2022 • 2 minutes to read • [Edit Online](#)

Managed assemblies are located and loaded with an algorithm that has various stages.

All managed assemblies except satellite assemblies and `WinRT` assemblies use the same algorithm.

When are managed assemblies loaded?

The most common mechanism to trigger a managed assembly load is a static assembly reference. These references are inserted by the compiler whenever code uses a type defined in another assembly. These assemblies are loaded (`load-by-name`) as needed by the runtime. The exact timing of when the static assembly references are loaded is unspecified. It can vary between runtime versions and is influenced by optimizations like inlining.

The direct use of the following APIs will also trigger loads:

API	DESCRIPTION	ACTIVE ASSEMBLYLOADCONTEXT
<code>AssemblyLoadContext.LoadFromAssemblyName</code>	<code>Load-by-name</code>	The <code>this</code> instance.
<code>AssemblyLoadContext.LoadFromAssemblyPath</code> <code>AssemblyLoadContext.LoadFromNativeImagePath</code>	Load from path.	The <code>this</code> instance.
<code>AssemblyLoadContext.LoadFromStream</code>	Load from object.	The <code>this</code> instance.
<code>Assembly.LoadFile</code>	Load from path in a new <code>AssemblyLoadContext</code> instance	The new <code>AssemblyLoadContext</code> instance.
<code>Assembly.LoadFrom</code>	Load from path in the <code>AssemblyLoadContext.Default</code> instance. Adds an <code>AppDomain.AssemblyResolve</code> handler. The handler will load the assembly's dependencies from its directory.	The <code>AssemblyLoadContext.Default</code> instance.
<code>Assembly.Load(AssemblyName)</code> <code>Assembly.Load(String)</code> <code>Assembly.LoadWithPartialName</code>	<code>Load-by-name</code> .	Inferred from caller. Prefer <code>AssemblyLoadContext</code> methods.
<code>Assembly.Load(Byte[])</code> <code>Assembly.Load(Byte[], Byte[])</code>	Load from object in a new <code>AssemblyLoadContext</code> instance.	The new <code>AssemblyLoadContext</code> instance.

API	DESCRIPTION	ACTIVE ASSEMBLYLOADCONTEXT
Type.GetType(String) Type.GetType(String, Boolean) Type.GetType(String, Boolean, Boolean)	Load-by-name .	Inferred from caller. Prefer Type.GetType methods with an <code>assemblyResolver</code> argument.
Assembly.GetType	If type <code>name</code> describes an assembly qualified generic type, trigger a Load-by-name .	Inferred from caller. Prefer Type.GetType when using assembly qualified type names.
Activator.CreateInstance(String, String) Activator.CreateInstance(String, String, Object[]) Activator.CreateInstance(String, String, Boolean, BindingFlags, Binder, Object[], CultureInfo, Object[])	Load-by-name .	Inferred from caller. Prefer Activator.CreateInstance methods taking a <code>Type</code> argument.

Algorithm

The following algorithm describes how the runtime loads a managed assembly.

1. Determine the `active AssemblyLoadContext`.
 - For a static assembly reference, the `active AssemblyLoadContext` is the instance that loaded the referring assembly.
 - Preferred APIs make the `active AssemblyLoadContext` explicit.
 - Other APIs infer the `active AssemblyLoadContext`. For these APIs, the `AssemblyLoadContext.CurrentContextualReflectionContext` property is used. If its value is `null`, then the inferred `AssemblyLoadContext` instance is used.
 - See the table in the [When are managed assemblies loaded?](#) section.
2. For the `Load-by-name` methods, the `active AssemblyLoadContext` loads the assembly in the following priority order:
 - Check its `cache-by-name` .
 - Call the `AssemblyLoadContext.Load` function.
 - Check the `AssemblyLoadContext.Default` instance's cache and run [managed assembly default probing](#) logic. If an assembly is newly loaded, a reference is added to the `AssemblyLoadContext.Default` instance's `cache-by-name` .
 - Raise the `AssemblyLoadContext.Resolving` event for the active `AssemblyLoadContext`.
 - Raise the `AppDomain.AssemblyResolve` event.
3. For the other types of loads, the `active AssemblyLoadContext` loads the assembly in the following priority order:
 - Check its `cache-by-name` .
 - Load from the specified path or raw assembly object. If an assembly is newly loaded, a reference is added to the `active AssemblyLoadContext` instance's `cache-by-name` .
4. In either case, if an assembly is newly loaded, then the `AppDomain.AssemblyLoad` event is raised.

Satellite assembly loading algorithm

9/20/2022 • 2 minutes to read • [Edit Online](#)

Satellite assemblies are used to store localized resources customized for language and culture.

Satellite assemblies use a different loading algorithm than general managed assemblies.

When are satellite assemblies loaded?

Satellite assemblies are loaded when loading a localized resource.

The basic API to load localized resources is the [System.Resources.ResourceManager](#) class. Ultimately the [ResourceManager](#) class will call the [GetSatelliteAssembly](#) method for each [CultureInfo.Name](#).

Higher-level APIs may abstract the low-level API.

Algorithm

The .NET Core resource fallback process involves the following steps:

1. Determine the `active AssemblyLoadContext` instance. In all cases, the `active` instance is the executing assembly's [AssemblyLoadContext](#).
2. The `active` instance attempts to load a satellite assembly for the requested culture in priority order by:
 - Checking its cache.
 - Checking the directory of the currently executing assembly for a subdirectory that matches the requested [CultureInfo.Name](#) (for example `es-MX`).

NOTE

This feature was not implemented in .NET Core before 3.0.

NOTE

On Linux and macOS, the subdirectory is case-sensitive and must either:

- Exactly match case.
- Be in lower case.

- If `active` is the [AssemblyLoadContext.Default](#) instance, by running the [default satellite \(resource\) assembly probing](#) logic.
- Calling the [AssemblyLoadContext.Load](#) function.
- Raising the [AssemblyLoadContext.Resolving](#) event.
- Raising the [AppDomain.AssemblyResolve](#) event.

3. If a satellite assembly is loaded:
 - The [AppDomain.AssemblyLoad](#) event is raised.
 - The assembly is searched for the requested resource. If the runtime finds the resource in the assembly,

it uses it. If it doesn't find the resource, it continues the search.

NOTE

To find a resource within the satellite assembly, the runtime searches for the resource file requested by the [ResourceManager](#) for the current [CultureInfo.Name](#). Within the resource file, it searches for the requested resource name. If either is not found, the resource is treated as not found.

4. The runtime next searches the parent culture assemblies through many potential levels, each time repeating steps 2 & 3.

Each culture has only one parent, which is defined by the [CultureInfo.Parent](#) property.

The search for parent cultures stops when a culture's [Parent](#) property is [CultureInfo.InvariantCulture](#).

For the [InvariantCulture](#), we don't return to steps 2 & 3, but rather continue with step 5.

5. If the resource is still not found, the resource for the default (fallback) culture is used.

Typically, the resources for the default culture are included in the main application assembly. However, you can specify [UltimateResourceFallbackLocation.Satellite](#) for the [NeutralResourcesLanguageAttribute.Location](#) property. This value indicates that the ultimate fallback location for resources is a satellite assembly rather than the main assembly.

NOTE

The default culture is the ultimate fallback. Therefore, we recommend that you always include an exhaustive set of resources in the default resource file. This helps prevent exceptions from being thrown. By having an exhaustive set, you provide a fallback for all resources and ensure that at least one resource is always present for the user, even if it is not culturally specific.

6. Finally,

- If the runtime doesn't find a resource file for a default (fallback) culture, a [MissingManifestResourceException](#) or [MissingSatelliteAssemblyException](#) exception is thrown.
- If the resource file is found but the requested resource isn't present, the request returns `null`.

Unmanaged (native) library loading algorithm

9/20/2022 • 2 minutes to read • [Edit Online](#)

Unmanaged libraries are located and loaded with an algorithm involving various stages.

The following algorithm describes how native libraries are loaded through `PInvoke`.

`PInvoke` load library algorithm

`PInvoke` uses the following algorithm when attempting to load an unmanaged assembly:

1. Determine the `active AssemblyLoadContext`. For an unmanaged load library, the `active AssemblyLoadContext` is the one with the assembly that defines the `PInvoke`.
2. For the `active AssemblyLoadContext`, try to find the assembly in priority order by:
 - Checking its cache.
 - Calling the current `System.Runtime.InteropServices.DllImportResolver` delegate set by the `NativeLibrary.SetDllImportResolver(Assembly, DllImportResolver)` function.
 - Calling the `AssemblyLoadContext.LoadUnmanagedDll` function on the `active AssemblyLoadContext`.
 - Checking the `AppDomain` instance's cache and running the `Unmanaged (native) library probing` logic.
 - Raising the `AssemblyLoadContext.ResolvingUnmanagedDll` event for the `active AssemblyLoadContext`.

Collect detailed assembly loading information

9/20/2022 • 5 minutes to read • [Edit Online](#)

Starting with .NET 5, the runtime can emit events through `EventPipe` with detailed information about [managed assembly loading](#) to aid in diagnosing assembly loading issues. These [events](#) are emitted by the `Microsoft-Windows-DotNETRuntime` provider under the `AssemblyLoader` keyword (`0x4`).

Prerequisites

- [.NET 5 SDK](#) or later versions
- `dotnet-trace` tool

NOTE

The scope of `dotnet-trace` capabilities is greater than collecting detailed assembly loading information. For more information on the usage of `dotnet-trace`, see [dotnet-trace](#).

Collect a trace with assembly loading events

You can use `dotnet-trace` to trace an existing process or to launch a child process and trace it from startup.

Trace an existing process

To enable assembly loading events in the runtime and collect a trace of them, use `dotnet-trace` with the following command:

```
dotnet-trace collect --providers Microsoft-Windows-DotNETRuntime:4 --process-id <pid>
```

This command collects a trace of the specified `<pid>`, enabling the `AssemblyLoader` events in the `Microsoft-Windows-DotNETRuntime` provider. The result is a `.nettrace` file.

Use dotnet-trace to launch a child process and trace it from startup

Sometimes it may be useful to collect a trace of a process from its startup. For apps running .NET 5 or later, you can use `dotnet-trace` to do this.

The following command launches `hello.exe` with `arg1` and `arg2` as its command line arguments and collects a trace from its runtime startup:

```
dotnet-trace collect --providers Microsoft-Windows-DotNETRuntime:4 -- hello.exe arg1 arg2
```

You can stop collecting the trace by pressing Enter or `Ctrl + C`. This also closes `hello.exe`.

NOTE

- Launching `hello.exe` via `dotnet-trace` redirects its input and output, and you won't be able to interact with it on the console by default. Use the `--show-child-io` switch to interact with its `stdin` and `stdout`.
- Exiting the tool via `Ctrl+C` or `SIGTERM` safely ends both the tool and the child process.
- If the child process exits before the tool, the tool exits as well and the trace should be safely viewable.

View a trace

The collected trace file can be viewed on Windows using the Events view in [PerfView](#). All the assembly loading events will be prefixed with `Microsoft-Windows-DotNETRuntime/AssemblyLoader`.

Example (on Windows)

This example uses the [assembly loading extension points sample](#). The application attempts to load an assembly `MyLibrary` - an assembly that is not referenced by the application and thus requires handling in an assembly loading extension point to be successfully loaded.

Collect the trace

1. Navigate to the directory with the downloaded sample. Build the application with:

```
dotnet build
```

2. Launch the application with arguments indicating that it should pause, waiting for a key press. On resuming, it will attempt to load the assembly in the default `AssemblyLoadContext` - without the handling necessary for a successful load. Navigate to the output directory and run:

```
AssemblyLoading.exe /d default
```

3. Find the application's process ID.

```
dotnet-trace ps
```

The output will list the available processes. For example:

```
35832 AssemblyLoading C:\src\AssemblyLoading\bin\Debug\net5.0\AssemblyLoading.exe
```

4. Attach `dotnet-trace` to the running application.

```
dotnet-trace collect --providers Microsoft-Windows-DotNETRuntime:4 --process-id 35832
```

5. In the window running the application, press any key to let the program continue. Tracing will automatically stop once the application exits.

View the trace

Open the collected trace in [PerfView](#) and open the Events view. Filter the events list to `Microsoft-Windows-DotNETRuntime/AssemblyLoader` events.

Event Types	Filter: AssemblyLoader
Microsoft-Windows-DotNETRuntime/AssemblyLoader/AppDomainAssemblyResolveHandlerInvoked	
Microsoft-Windows-DotNETRuntime/AssemblyLoader/AssemblyLoadContextResolvingHandlerInvoked	
Microsoft-Windows-DotNETRuntime/AssemblyLoader/KnownPathProbed	
Microsoft-Windows-DotNETRuntime/AssemblyLoader/ResolutionAttempted	
Microsoft-Windows-DotNETRuntime/AssemblyLoader/Start	
Microsoft-Windows-DotNETRuntime/AssemblyLoader/Stop	

All assembly loads that occurred in the application after tracing started will be shown. To inspect the load operation for the assembly of interest for this example - `MyLibrary`, we can do some more filtering.

Assembly loads

Filter the view to the `Start` and `Stop` events under `Microsoft-Windows-DotNETRuntime/AssemblyLoader` using the event list on the left. Add the columns `AssemblyName`, `ActivityID`, and `Success` to the view. Filter to events containing `MyLibrary`.

Text Filter:	MyLibrary	Columns To Display:	Cols	AssemblyName	ActivityID	Success *
Histogram:						
Event Name		AssemblyName		ActivityID	Success	
Microsoft-Windows-DotNETRuntime/AssemblyLoader/Start		MyLibrary, Culture=neutral, PublicKeyToken=null	//1/2/			
Microsoft-Windows-DotNETRuntime/AssemblyLoader/Stop		MyLibrary, Culture=neutral, PublicKeyToken=null	//1/2/		False	

EVENT NAME	ASSEMBLYNAME	ACTIVITYID	SUCCESS
AssemblyLoader/Start	MyLibrary, Culture=neutral, PublicKeyToken=null	//1/2/	
AssemblyLoader/Stop	MyLibrary, Culture=neutral, PublicKeyToken=null	//1/2/	False

You should see one `Start` / `Stop` pair with `Success=False` on the `Stop` event, indicating the load operation failed. Note that the two events have the same activity ID. The activity ID can be used to filter all the other assembly loader events to just the ones corresponding to this load operation.

Breakdown of attempt to load

For a more detailed breakdown of the load operation, filter the view to the `ResolutionAttempted` events under `Microsoft-Windows-DotNETRuntime/AssemblyLoader` using the event list on the left. Add the columns `AssemblyName`, `Stage`, and `Result` to the view. Filter to events with the activity ID from the `Start` / `Stop` pair.

Text Filter:	//1/2/	Columns To Display:	Cols	AssemblyName Stage Result *
Histogram:				
Event Name		AssemblyName	Stage	Result
Microsoft-Windows-DotNETRuntime/AssemblyLoader/ResolutionAttempted		MyLibrary, Culture=neutral, PublicKeyToken=null	FindInLoadContext	AssemblyNotFound
Microsoft-Windows-DotNETRuntime/AssemblyLoader/ResolutionAttempted		MyLibrary, Culture=neutral, PublicKeyToken=null	ApplicationAssemblies	AssemblyNotFound
Microsoft-Windows-DotNETRuntime/AssemblyLoader/ResolutionAttempted		MyLibrary, Culture=neutral, PublicKeyToken=null	AssemblyLoadContextResolvingEvent	AssemblyNotFound
Microsoft-Windows-DotNETRuntime/AssemblyLoader/ResolutionAttempted		MyLibrary, Culture=neutral, PublicKeyToken=null	AppDomainAssemblyResolveEvent	AssemblyNotFound

EVENT NAME	ASSEMBLYNAME	STAGE	RESULT
AssemblyLoader/ResolutionAtt	MyLibrary, Culture=neutral, PublicKeyToken=null	FindInLoadContext	AssemblyNotFound
AssemblyLoader/ResolutionAtt	MyLibrary, Culture=neutral, PublicKeyToken=null	ApplicationAssemblies	AssemblyNotFound
AssemblyLoader/ResolutionAtt	MyLibrary, Culture=neutral, PublicKeyToken=null	AssemblyLoadContextResolving	AssemblyNotFound
AssemblyLoader/ResolutionAtt	MyLibrary, Culture=neutral, PublicKeyToken=null	AppDomainAssemblyResolveEver	AssemblyNotFound

The events above indicate that the assembly loader attempted to resolve the assembly by looking in the current load context, running the default probing logic for managed application assemblies, invoking handlers for the `AssemblyLoadContext.Resolving` event, and invoking handlers for the `AppDomain.AssemblyResolve`. For all of these steps, the assembly was not found.

Extension points

To see which extension points were invoked, filter the view to the `AssemblyLoadContextResolvingHandlerInvoked` and `AppDomainAssemblyResolveHandlerInvoked` under `Microsoft-Windows-DotNETRuntime/AssemblyLoader` using the event list on the left. Add the columns `AssemblyName` and `HandlerName` to the view. Filter to events with the activity ID from the `Start / Stop` pair.

Event Name	AssemblyName	HandlerName
Microsoft-Windows-DotNETRuntime/AssemblyLoader/AssemblyLoadContextResolvingHandlerInvoked	MyLibrary, Culture=neutral, PublicKeyToken=null	OnAssemblyLoadContextResolving
Microsoft-Windows-DotNETRuntime/AssemblyLoader/AppDomainAssemblyResolveHandlerInvoked	MyLibrary, Culture=neutral, PublicKeyToken=null	OnAppDomainAssemblyResolve
Event Name	AssemblyName	HandlerName
AssemblyLoader/AssemblyLoadContextResolvingHandlerInvoked	MyLibrary, Culture=neutral, PublicKeyToken=null	OnAssemblyLoadContextResolving
AssemblyLoader/AppDomainAssemblyResolveHandlerInvoked	MyLibrary, Culture=neutral, PublicKeyToken=null	OnAppDomainAssemblyResolve

The events above indicate that a handler named `OnAssemblyLoadContextResolving` was invoked for the `AssemblyLoadContext.Resolving` event and a handler named `OnAppDomainAssemblyResolve` was invoked for the `AppDomain.AssemblyResolve` event.

Collect another trace

Run the application with arguments such that its handler for the `AssemblyLoadContext.Resolving` event will load the `MyLibrary` assembly.

```
AssemblyLoading /d default alc-resolving
```

Collect and open another `.nettrace` file using the [steps from above](#).

Filter to the `Start` and `Stop` events for `MyLibrary` again. You should see a `Start / Stop` pair with another `Start / Stop` between them. The inner load operation represents the load triggered by the handler for `AssemblyLoadContext.Resolving` when it called `AssemblyLoadContext.LoadFromAssemblyPath`. This time, you should see `Success=True` on the `Stop` event, indicating the load operation succeeded. The `ResultAssemblyPath` field shows the path of the resulting assembly.

Event Name	AssemblyName	ActivityID	Success	ResultAssemblyPath
Microsoft-Windows-DotNETRuntime/AssemblyLoader/Start	MyLibrary, Culture=neutral, PublicKeyToken=null	//1/2/		
Microsoft-Windows-DotNETRuntime/AssemblyLoader/Start	MyLibrary, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null	//1/2/1		
Microsoft-Windows-DotNETRuntime/AssemblyLoader/Stop	MyLibrary, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null	//1/2/1	True	C:\src\AssemblyLoading\bin\Debug\net5.0\MyLibrary.dll
Microsoft-Windows-DotNETRuntime/AssemblyLoader/Stop	MyLibrary, Culture=neutral, PublicKeyToken=null	//1/2/	True	C:\src\AssemblyLoading\bin\Debug\net5.0\MyLibrary.dll
Event Name	AssemblyName	ActivityID	Success	ResultAssemblyPath
AssemblyLoader/Start	MyLibrary, Culture=neutral, PublicKeyToken=null	//1/2/		
AssemblyLoader/Start	MyLibrary, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null	//1/2/1		

EVENT NAME	ASSEMBLYNAME	ACTIVITYID	SUCCESS	RESULTASSEMBLYPATH
AssemblyLoader/Stop	MyLibrary, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null	//1/2/1/	True	C:\src\AssemblyLoading\bin\Debug\net5.0\MyLibrary.dll
AssemblyLoader/Stop	MyLibrary, Culture=neutral, PublicKeyToken=null	//1/2/	True	C:\src\AssemblyLoading\bin\Debug\net5.0\MyLibrary.dll

We can then look at the `ResolutionAttempted` events with the activity ID from the outer load to determine the step at which the assembly was successfully resolved. This time, the events will show that the `AssemblyLoadContextResolvingEvent` stage was successful. The `ResultAssemblyPath` field shows the path of the resulting assembly.

Event Name	AssemblyName	Stage	Result	ResultAssemblyPath
Microsoft-Windows-DotNETRuntime/AssemblyLoader/ResolutionAttempted	MyLibrary, Culture=neutral	FindInLoadContext	AssemblyNotFound	
Microsoft-Windows-DotNETRuntime/AssemblyLoader/ResolutionAttempted	MyLibrary, Culture=neutral	ApplicationAssemblies	AssemblyNotFound	
Microsoft-Windows-DotNETRuntime/AssemblyLoader/ResolutionAttempted	MyLibrary, Culture=neutral	AssemblyLoadContextResolvingEvent	Success	C:\src\AssemblyLoading\bin\Debug\net5.0\MyLibrary.dll

EVENT NAME	ASSEMBLYNAME	STAGE	RESULT	RESULTASSEMBLYPATH
AssemblyLoader/ResolveHandler	MyLibrary, Culture=neutral, PublicKeyToken=null	FindInLoadContext	AssemblyNotFound	
AssemblyLoader/ResolveHandler	MyLibrary, Culture=neutral, PublicKeyToken=null	ApplicationAssemblies	AssemblyNotFound	
AssemblyLoader/ResolveHandler	MyLibrary, Culture=neutral, PublicKeyToken=null	AssemblyLoadContextResolvingEvent	Success	C:\src\AssemblyLoading\bin\Debug\net5.0\MyLibrary.dll

Looking at `AssemblyLoadContextResolvingHandlerInvoked` events will show that the handler named `OnAssemblyLoadContextResolving` was invoked. The `ResultAssemblyPath` field shows the path of the assembly returned by the handler.

Event Name	AssemblyName	HandlerName	ResultAssemblyPath
Microsoft-Windows-DotNETRuntime/AssemblyLoader/AssemblyLoadContextResolvingHandlerInvoked	MyLibrary, Culture=neutral	OnAssemblyLoadContextResolving	C:\src\AssemblyLoading\bin\Debug\net5.0\MyLibrary.dll

EVENT NAME	ASSEMBLYNAME	HANDLERNAME	RESULTASSEMBLYPATH
AssemblyLoader/AssemblyLoadContextResolvingHandlerInvoked	MyLibrary, Culture=neutral, PublicKeyToken=null	OnAssemblyLoadContextResolving	C:\src\AssemblyLoading\bin\Debug\net5.0\MyLibrary.dll

Note that there is no longer a `ResolutionAttempted` event with the `AppDomainAssemblyResolveEvent` stage or any `AppDomainAssemblyResolveHandlerInvoked` events, as the assembly was successfully loaded before reaching the step of the loading algorithm that raises the `AppDomain.AssemblyResolve` event.

See also

- [Assembly loader events](#)

- [dotnet-trace](#)
- [PerfView](#)

Create a .NET Core application with plugins

9/20/2022 • 8 minutes to read • [Edit Online](#)

This tutorial shows you how to create a custom [AssemblyLoadContext](#) to load plugins. An [AssemblyDependencyResolver](#) is used to resolve the dependencies of the plugin. The tutorial correctly isolates the plugin's dependencies from the hosting application. You'll learn how to:

- Structure a project to support plugins.
- Create a custom [AssemblyLoadContext](#) to load each plugin.
- Use the [System.Runtime.Loader.AssemblyDependencyResolver](#) type to allow plugins to have dependencies.
- Author plugins that can be easily deployed by just copying the build artifacts.

Prerequisites

- Install the [.NET 5 SDK](#) or a newer version.

NOTE

The sample code targets .NET 5, but all the features it uses were introduced in .NET Core 3.0 and are available in all .NET releases since then.

Create the application

The first step is to create the application:

1. Create a new folder, and in that folder run the following command:

```
dotnet new console -o AppWithPlugin
```

2. To make building the project easier, create a Visual Studio solution file in the same folder. Run the following command:

```
dotnet new sln
```

3. Run the following command to add the app project to the solution:

```
dotnet sln add AppWithPlugin/AppWithPlugin.csproj
```

Now we can fill in the skeleton of our application. Replace the code in the *AppWithPlugin/Program.cs* file with the following code:

```

using PluginBase;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Reflection;

namespace AppWithPlugin
{
    class Program
    {
        static void Main(string[] args)
        {
            try
            {
                if (args.Length == 1 && args[0] == "/d")
                {
                    Console.WriteLine("Waiting for any key...");
                    Console.ReadLine();
                }

                // Load commands from plugins.

                if (args.Length == 0)
                {
                    Console.WriteLine("Commands: ");
                    // Output the loaded commands.
                }
                else
                {
                    foreach (string commandName in args)
                    {
                        Console.WriteLine($"-- {commandName} --");

                        // Execute the command with the name passed as an argument.

                        Console.WriteLine();
                    }
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex);
            }
        }
    }
}

```

Create the plugin interfaces

The next step in building an app with plugins is defining the interface the plugins need to implement. We suggest that you make a class library that contains any types that you plan to use for communicating between your app and plugins. This division allows you to publish your plugin interface as a package without having to ship your full application.

In the root folder of the project, run `dotnet new classlib -o PluginBase`. Also, run `dotnet sln add PluginBase/PluginBase.csproj` to add the project to the solution file. Delete the `PluginBase/Class1.cs` file, and create a new file in the `PluginBase` folder named `ICommand.cs` with the following interface definition:

```

namespace PluginBase
{
    public interface ICommand
    {
        string Name { get; }
        string Description { get; }

        int Execute();
    }
}

```

This `ICommand` interface is the interface that all of the plugins will implement.

Now that the `ICommand` interface is defined, the application project can be filled in a little more. Add a reference from the `AppWithPlugin` project to the `PluginBase` project with the

```
dotnet add AppWithPlugin/AppWithPlugin.csproj reference PluginBase/PluginBase.csproj
```

command from the root folder.

Replace the `// Load commands from plugins` comment with the following code snippet to enable it to load plugins from given file paths:

```

string[] pluginPaths = new string[]
{
    // Paths to plugins to load.
};

IEnumerable<ICommand> commands = pluginPaths.SelectMany(pluginPath =>
{
    Assembly pluginAssembly = LoadPlugin(pluginPath);
    return CreateCommands(pluginAssembly);
}).ToList();

```

Then replace the `// Output the loaded commands` comment with the following code snippet:

```

foreach (ICommand command in commands)
{
    Console.WriteLine($"{command.Name}\t - {command.Description}");
}

```

Replace the `// Execute the command with the name passed as an argument` comment with the following snippet:

```

ICommand command = commands.FirstOrDefault(c => c.Name == commandName);
if (command == null)
{
    Console.WriteLine("No such command is known.");
    return;
}

command.Execute();

```

And finally, add static methods to the `Program` class named `LoadPlugin` and `CreateCommands`, as shown here:

```

static Assembly LoadPlugin(string relativePath)
{
    throw new NotImplementedException();
}

static IEnumerable< ICommand> CreateCommands(Assembly assembly)
{
    int count = 0;

    foreach (Type type in assembly.GetTypes())
    {
        if (typeof(ICommand).IsAssignableFrom(type))
        {
            ICommand result = Activator.CreateInstance(type) as ICommand;
            if (result != null)
            {
                count++;
                yield return result;
            }
        }
    }

    if (count == 0)
    {
        string availableTypes = string.Join(", ", assembly.GetTypes().Select(t => t.FullName));
        throw new ApplicationException(
            $"Can't find any type which implements ICommand in {assembly} from {assembly.Location}.\\n" +
            $"Available types: {availableTypes}");
    }
}

```

Load plugins

Now the application can correctly load and instantiate commands from loaded plugin assemblies, but it's still unable to load the plugin assemblies. Create a file named *PluginLoadContext.cs* in the *AppWithPlugin* folder with the following contents:

```

using System;
using System.Reflection;
using System.Runtime.Loader;

namespace AppWithPlugin
{
    class PluginLoadContext : AssemblyLoadContext
    {
        private AssemblyDependencyResolver _resolver;

        public PluginLoadContext(string pluginPath)
        {
            _resolver = new AssemblyDependencyResolver(pluginPath);
        }

        protected override Assembly Load(AssemblyName assemblyName)
        {
            string assemblyPath = _resolver.ResolveAssemblyToPath(assemblyName);
            if (assemblyPath != null)
            {
                return LoadFromAssemblyPath(assemblyPath);
            }

            return null;
        }

        protected override IntPtr LoadUnmanagedDll(string unmanagedDllName)
        {
            string libraryPath = _resolver.ResolveUnmanagedDllToPath(unmanagedDllName);
            if (libraryPath != null)
            {
                return LoadUnmanagedDllFromPath(libraryPath);
            }

            return IntPtr.Zero;
        }
    }
}

```

The `PluginLoadContext` type derives from `AssemblyLoadContext`. The `AssemblyLoadContext` type is a special type in the runtime that allows developers to isolate loaded assemblies into different groups to ensure that assembly versions don't conflict. Additionally, a custom `AssemblyLoadContext` can choose different paths to load assemblies from and override the default behavior. The `PluginLoadContext` uses an instance of the `AssemblyDependencyResolver` type introduced in .NET Core 3.0 to resolve assembly names to paths. The `AssemblyDependencyResolver` object is constructed with the path to a .NET class library. It resolves assemblies and native libraries to their relative paths based on the `.deps.json` file for the class library whose path was passed to the `AssemblyDependencyResolver` constructor. The custom `AssemblyLoadContext` enables plugins to have their own dependencies, and the `AssemblyDependencyResolver` makes it easy to correctly load the dependencies.

Now that the `AppWithPlugin` project has the `PluginLoadContext` type, update the `Program.LoadPlugin` method with the following body:

```

static Assembly LoadPlugin(string relativePath)
{
    // Navigate up to the solution root
    string root = Path.GetFullPath(Path.Combine(
        Path.GetDirectoryName(
            Path.GetDirectoryName(
                Path.GetDirectoryName(
                    Path.GetDirectoryName(
                        Path.GetDirectoryName(typeof(Program).Assembly.Location))))));
}

string pluginLocation = Path.GetFullPath(Path.Combine(root, relativePath.Replace('\\',
Path.DirectorySeparatorChar)));
Console.WriteLine($"Loading commands from: {pluginLocation}");
PluginLoadContext loadContext = new PluginLoadContext(pluginLocation);
return loadContext.LoadFromAssemblyName(new
AssemblyName(Path.GetFileNameWithoutExtension(pluginLocation)));
}

```

By using a different `PluginLoadContext` instance for each plugin, the plugins can have different or even conflicting dependencies without issue.

Simple plugin with no dependencies

Back in the root folder, do the following:

1. Run the following command to create a new class library project named `HelloPlugin`:

```
dotnet new classlib -o HelloPlugin
```

2. Run the following command to add the project to the `AppWithPlugin` solution:

```
dotnet sln add HelloPlugin/HelloPlugin.csproj
```

3. Replace the `HelloPlugin/Class1.cs` file with a file named `HelloCommand.cs` with the following contents:

```

using PluginBase;
using System;

namespace HelloPlugin
{
    public class HelloCommand : ICommand
    {
        public string Name { get => "hello"; }
        public string Description { get => "Displays hello message." }

        public int Execute()
        {
            Console.WriteLine("Hello !!!");
            return 0;
        }
    }
}

```

Now, open the `HelloPlugin.csproj` file. It should look similar to the following:

```
<Project Sdk="Microsoft.NET.Sdk">

<PropertyGroup>
  <TargetFramework>net5.0</TargetFramework>
</PropertyGroup>

</Project>
```

In between the `<PropertyGroup>` tags, add the following element:

```
<EnableDynamicLoading>true</EnableDynamicLoading>
```

The `<EnableDynamicLoading>true</EnableDynamicLoading>` prepares the project so that it can be used as a plugin. Among other things, this will copy all of its dependencies to the output of the project. For more details see [EnableDynamicLoading](#).

In between the `<Project>` tags, add the following elements:

```
<ItemGroup>
  <ProjectReference Include="..\PluginBase\PluginBase.csproj">
    <Private>false</Private>
    <ExcludeAssets>runtime</ExcludeAssets>
  </ProjectReference>
</ItemGroup>
```

The `<Private>false</Private>` element is important. This tells MSBuild to not copy *PluginBase.dll* to the output directory for *HelloPlugin*. If the *PluginBase.dll* assembly is present in the output directory, `PluginLoadContext` will find the assembly there and load it when it loads the *HelloPlugin.dll* assembly. At this point, the `HelloPlugin.HelloCommand` type will implement the `ICommand` interface from the *PluginBase.dll* in the output directory of the `HelloPlugin` project, not the `ICommand` interface that is loaded into the default load context. Since the runtime sees these two types as different types from different assemblies, the `AppWithPlugin.Program.CreateCommand` method won't find the commands. As a result, the `<Private>false</Private>` metadata is required for the reference to the assembly containing the plugin interfaces.

Similarly, the `<ExcludeAssets>runtime</ExcludeAssets>` element is also important if the `PluginBase` references other packages. This setting has the same effect as `<Private>false</Private>` but works on package references that the `PluginBase` project or one of its dependencies may include.

Now that the `HelloPlugin` project is complete, you should update the `AppWithPlugin` project to know where the `HelloPlugin` plugin can be found. After the `// Paths to plugins to load` comment, add `@"HelloPlugin\bin\Debug\net5.0\HelloPlugin.dll"` (this path could be different based on the .NET Core version you use) as an element of the `pluginPaths` array.

Plugin with library dependencies

Almost all plugins are more complex than a simple "Hello World", and many plugins have dependencies on other libraries. The `JsonPlugin` and `OldJsonPlugin` projects in the sample show two examples of plugins with NuGet package dependencies on `Newtonsoft.Json`. Because of this, all plugin projects should add `<EnableDynamicLoading>true</EnableDynamicLoading>` to the project properties so that they copy all of their dependencies to the output of `dotnet build`. Publishing the class library with `dotnet publish` will also copy all of its dependencies to the publish output.

Other examples in the sample

The complete source code for this tutorial can be found in [the dotnet/samples repository](#). The completed sample includes a few other examples of `AssemblyDependencyResolver` behavior. For example, the `AssemblyDependencyResolver` object can also resolve native libraries as well as localized satellite assemblies included in NuGet packages. The `UVPlugin` and `FrenchPlugin` in the samples repository demonstrate these scenarios.

Reference a plugin interface from a NuGet package

Let's say that there is an app A that has a plugin interface defined in the NuGet package named `A.PluginBase`. How do you reference the package correctly in your plugin project? For project references, using the `<Private>false</Private>` metadata on the `ProjectReference` element in the project file prevented the dll from being copied to the output.

To correctly reference the `A.PluginBase` package, you want to change the `<PackageReference>` element in the project file to the following:

```
<PackageReference Include="A.PluginBase" Version="1.0.0">
  <ExcludeAssets>runtime</ExcludeAssets>
</PackageReference>
```

This prevents the `A.PluginBase` assemblies from being copied to the output directory of your plugin and ensures that your plugin will use A's version of `A.PluginBase`.

Plugin target framework recommendations

Because plugin dependency loading uses the `.deps.json` file, there is a gotcha related to the plugin's target framework. Specifically, your plugins should target a runtime, such as .NET 5, instead of a version of .NET Standard. The `.deps.json` file is generated based on which framework the project targets, and since many .NET Standard-compatible packages ship reference assemblies for building against .NET Standard and implementation assemblies for specific runtimes, the `.deps.json` may not correctly see implementation assemblies, or it may grab the .NET Standard version of an assembly instead of the .NET Core version you expect.

Plugin framework references

Currently, plugins can't introduce new frameworks into the process. For example, you can't load a plugin that uses the `Microsoft.AspNetCore.App` framework into an application that only uses the root `Microsoft.NETCore.App` framework. The host application must declare references to all frameworks needed by plugins.

How to use and debug assembly unloadability in .NET

9/20/2022 • 12 minutes to read • [Edit Online](#)

.NET Core 3.0 introduced the ability to load and later unload a set of assemblies. In .NET Framework, custom app domains were used for this purpose, but .NET [Core] only supports a single default app domain.

Unloadability is supported through [AssemblyLoadContext](#). You can load a set of assemblies into a collectible [AssemblyLoadContext](#), execute methods in them or just inspect them using reflection, and finally unload the [AssemblyLoadContext](#). That unloads the assemblies loaded into the [AssemblyLoadContext](#).

There's one noteworthy difference between the unloading using [AssemblyLoadContext](#) and using AppDomains. With AppDomains, the unloading is forced. At unload time, all threads running in the target AppDomain are aborted, managed COM objects created in the target AppDomain are destroyed, and so on. With [AssemblyLoadContext](#), the unload is "cooperative". Calling the [AssemblyLoadContext.Unload](#) method just initiates the unloading. The unloading finishes after:

- No threads have methods from the assemblies loaded into the [AssemblyLoadContext](#) on their call stacks.
- None of the types from the assemblies loaded into the [AssemblyLoadContext](#), instances of those types, and the assemblies themselves are referenced by:
 - References outside of the [AssemblyLoadContext](#), except for weak references ([WeakReference](#) or [WeakReference<T>](#)).
 - Strong garbage collector (GC) handles ([GCHandleType.Normal](#) or [GCHandleType.Pinned](#)) from both inside and outside of the [AssemblyLoadContext](#).

Use collectible AssemblyLoadContext

This section contains a detailed step-by-step tutorial that shows a simple way to load a .NET Core application into a collectible [AssemblyLoadContext](#), execute its entry point, and then unload it. You can find a complete sample at <https://github.com/dotnet/samples/tree/main/core/tutorials/Unloading>.

Create a collectible AssemblyLoadContext

Derive your class from the [AssemblyLoadContext](#) and override its [AssemblyLoadContext.Load](#) method. That method resolves references to all assemblies that are dependencies of assemblies loaded into that [AssemblyLoadContext](#).

The following code is an example of the simplest custom [AssemblyLoadContext](#):

```
class TestAssemblyLoadContext : AssemblyLoadContext
{
    public TestAssemblyLoadContext() : base(isCollectible: true)
    {
    }

    protected override Assembly Load(AssemblyName name)
    {
        return null;
    }
}
```

As you can see, the [Load](#) method returns [null](#). That means that all the dependency assemblies are loaded into

the default context, and the new context contains only the assemblies explicitly loaded into it.

If you want to load some or all of the dependencies into the `AssemblyLoadContext` too, you can use the `AssemblyDependencyResolver` in the `Load` method. The `AssemblyDependencyResolver` resolves the assembly names to absolute assembly file paths. The resolver uses the `.deps.json` file and assembly files in the directory of the main assembly loaded into the context.

```
using System.Reflection;
using System.Runtime.Loader;

namespace complex
{
    class TestAssemblyLoadContext : AssemblyLoadContext
    {
        private AssemblyDependencyResolver _resolver;

        public TestAssemblyLoadContext(string mainAssemblyToLoadPath) : base(isCollectible: true)
        {
            _resolver = new AssemblyDependencyResolver(mainAssemblyToLoadPath);
        }

        protected override Assembly Load(AssemblyName name)
        {
            string assemblyPath = _resolver.ResolveAssemblyToPath(name);
            if (assemblyPath != null)
            {
                return LoadFromAssemblyPath(assemblyPath);
            }

            return null;
        }
    }
}
```

Use a custom collectible `AssemblyLoadContext`

This section assumes the simpler version of the `TestAssemblyLoadContext` is being used.

You can create an instance of the custom `AssemblyLoadContext` and load an assembly into it as follows:

```
var alc = new TestAssemblyLoadContext();
Assembly a = alc.LoadFromAssemblyPath(assemblyPath);
```

For each of the assemblies referenced by the loaded assembly, the `TestAssemblyLoadContext.Load` method is called so that the `TestAssemblyLoadContext` can decide where to get the assembly from. In our case, it returns `null` to indicate that it should be loaded into the default context from locations that the runtime uses to load assemblies by default.

Now that an assembly was loaded, you can execute a method from it. Run the `Main` method:

```
var args = new object[1] {new string[] {"Hello"}};
int result = (int) a.EntryPoint.Invoke(null, args);
```

After the `Main` method returns, you can initiate unloading by either calling the `Unload` method on the custom `AssemblyLoadContext` or getting rid of the reference you have to the `AssemblyLoadContext`:

```
alc.Unload();
```

This is sufficient to unload the test assembly. Let's actually put all of this into a separate non-inlineable method

to ensure that the `TestAssemblyLoadContext`, `Assembly`, and `MethodInfo` (the `Assembly.EntryPoint`) can't be kept alive by stack slot references (real- or JIT-introduced locals). That could keep the `TestAssemblyLoadContext` alive and prevent the unload.

Also, return a weak reference to the `AssemblyLoadContext` so that you can use it later to detect unload completion.

```
[MethodImpl(MethodImplOptions.NoInlining)]
static int ExecuteAndUnload(string assemblyPath, out WeakReference alcWeakRef)
{
    var alc = new TestAssemblyLoadContext();
    Assembly a = alc.LoadFromAssemblyPath(assemblyPath);

    alcWeakRef = new WeakReference(alc, trackResurrection: true);

    var args = new object[1] {new string[] {"Hello"}};
    int result = (int) a.EntryPoint.Invoke(null, args);

    alc.Unload();

    return result;
}
```

Now you can run this function to load, execute, and unload the assembly.

```
WeakReference testAlcWeakRef;
int result = ExecuteAndUnload("absolute/path/to/your/assembly", out testAlcWeakRef);
```

However, the unload doesn't complete immediately. As previously mentioned, it relies on the garbage collector to collect all the objects from the test assembly. In many cases, it isn't necessary to wait for the unload completion. However, there are cases where it's useful to know that the unload has finished. For example, you may want to delete the assembly file that was loaded into the custom `AssemblyLoadContext` from disk. In such a case, the following code snippet can be used. It triggers garbage collection and waits for pending finalizers in a loop until the weak reference to the custom `AssemblyLoadContext` is set to `null`, indicating the target object was collected. In most cases, just one pass through the loop is required. However, for more complex cases where objects created by the code running in the `AssemblyLoadContext` have finalizers, more passes may be needed.

```
for (int i = 0; testAlcWeakRef.IsAlive && (i < 10); i++)
{
    GC.Collect();
    GC.WaitForPendingFinalizers();
}
```

The Unloading event

In some cases, it may be necessary for the code loaded into a custom `AssemblyLoadContext` to perform some cleanup when the unloading is initiated. For example, it may need to stop threads or clean up strong GC handles. The `Unloading` event can be used in such cases. A handler that performs the necessary cleanup can be hooked to this event.

Troubleshoot unloadability issues

Due to the cooperative nature of the unloading, it's easy to forget about references that may be keeping the stuff in a collectible `AssemblyLoadContext` alive and preventing unload. Here is a summary of entities (some of them non-obvious) that can hold the references:

- Regular references held from outside of the collectible `AssemblyLoadContext` that are stored in a stack slot or a processor register (method locals, either explicitly created by the user code or implicitly by the just-in-time

(JIT) compiler), a static variable, or a strong (pinning) GC handle, and transitively pointing to:

- An assembly loaded into the collectible `AssemblyLoadContext`.
- A type from such an assembly.
- An instance of a type from such an assembly.
- Threads running code from an assembly loaded into the collectible `AssemblyLoadContext`.
- Instances of custom, non-collectible `AssemblyLoadContext` types created inside of the collectible `AssemblyLoadContext`.
- Pending `RegisteredWaitHandle` instances with callbacks set to methods in the custom `AssemblyLoadContext`.

TIP

Object references that are stored in stack slots or processor registers and that could prevent unloading of an `AssemblyLoadContext` can occur in the following situations:

- When function call results are passed directly to another function, even though there is no user-created local variable.
- When the JIT compiler keeps a reference to an object that was available at some point in a method.

Debug unloading issues

Debugging issues with unloading can be tedious. You can get into situations where you don't know what can be holding an `AssemblyLoadContext` alive, but the unload fails. The best weapon to help with that is WinDbg (LLDB on Unix) with the SOS plugin. You need to find what's keeping a `LoaderAllocator` belonging to the specific `AssemblyLoadContext` alive. The SOS plugin allows you to look at GC heap objects, their hierarchies, and roots.

To load the plugin into the debugger, enter the following command in the debugger command line:

In WinDbg (it seems WinDbg does that automatically when breaking into .NET Core application):

```
.loadby sos coreclr
```

In LLDB:

```
plugin load /path/to/libssosplugin.so
```

Let's debug an example program that has problems with unloading. Source code is included below. When you run it under WinDbg, the program breaks into the debugger right after attempting to check for the unload success. You can then start looking for the culprits.

TIP

If you debug using LLDB on Unix, the SOS commands in the following examples don't have the `!` in front of them.

```
!dumpheap -type LoaderAllocator
```

This command dumps all objects with a type name containing `LoaderAllocator` that are in the GC heap. Here is an example:

Address	MT	Size
000002b78000ce40	00007ffadc93a288	48
000002b78000ceb0	00007ffadc93a218	24

Statistics:

MT	Count	TotalSize	Class Name
00007ffadc93a218	1	24	System.Reflection.LoaderAllocatorScout
00007ffadc93a288	1	48	System.Reflection.LoaderAllocator
Total		2 objects	

In the "Statistics:" part below, check the `MT` (`MethodTable`) belonging to the `System.Reflection.LoaderAllocator`, which is the object we care about. Then, in the list at the beginning, find the entry with `MT` matching that one and get the address of the object itself. In our case, it is "000002b78000ce40".

Now that we know the address of the `LoaderAllocator` object, we can use another command to find its GC roots:

```
!gcroot -all 0x000002b78000ce40
```

This command dumps the chain of object references that lead to the `LoaderAllocator` instance. The list starts with the root, which is the entity that keeps our `LoaderAllocator` alive and thus is the core of the problem. The root can be a stack slot, a processor register, a GC handle, or a static variable.

Here is an example of the output of the `gcroot` command:

```
Thread 4ac:
00000cf9499dd20 00007ffa7d0236bc example.Program.Main(System.String[])
[E:\unloadability\example\Program.cs @ 70]
rbp-20: 000000cf9499dd90
    -> 000002b78000d328 System.Reflection.RuntimeMethodInfo
    -> 000002b78000d1f8 System.RuntimeType+RuntimeTypeCache
    -> 000002b78000d1d0 System.RuntimeType
    -> 000002b78000ce40 System.Reflection.LoaderAllocator

HandleTable:
000002b7f8a81198 (strong handle)
-> 000002b78000d948 test.Test
-> 000002b78000ce40 System.Reflection.LoaderAllocator

000002b7f8a815f8 (pinned handle)
-> 000002b790001038 System.Object[]
-> 000002b78000d390 example.TestInfo
-> 000002b78000d328 System.Reflection.RuntimeMethodInfo
-> 000002b78000d1f8 System.RuntimeType+RuntimeTypeCache
-> 000002b78000d1d0 System.RuntimeType
-> 000002b78000ce40 System.Reflection.LoaderAllocator

Found 3 roots.
```

The next step is to figure out where the root is located so you can fix it. The easiest case is when the root is a stack slot or a processor register. In that case, the `gcroot` shows the name of the function whose frame contains the root and the thread executing that function. The difficult case is when the root is a static variable or a GC handle.

In the previous example, the first root is a local of type `System.Reflection.RuntimeMethodInfo` stored in the frame of the function `example.Program.Main(System.String[])` at address `rbp-20` (`rbp` is the processor register `rbp` and `-20` is a hexadecimal offset from that register).

The second root is a normal (strong) `GCHandle` that holds a reference to an instance of the `test.Test` class.

The third root is a pinned `GCHandle`. This one is actually a static variable, but unfortunately, there is no way to tell. Statics for reference types are stored in a managed object array in internal runtime structures.

Another case that can prevent unloading of an `AssemblyLoadContext` is when a thread has a frame of a method from an assembly loaded into the `AssemblyLoadContext` on its stack. You can check that by dumping managed call stacks of all threads:

```
~*e !clrstack
```

The command means "apply to all threads the `!clrstack` command". The following is the output of that command for the example. Unfortunately, LLDB on Unix doesn't have any way to apply a command to all threads, so you must manually switch threads and repeat the `clrstack` command. Ignore all threads where the debugger says "Unable to walk the managed stack".

```

OS Thread Id: 0x6ba8 (0)
    Child SP          IP Call Site
0000001fc697d5c8 00007ffb50d9de12 [HelperMethodFrame: 0000001fc697d5c8]
System.Diagnostics.Debugger.BreakInternal()
0000001fc697d6d0 00007ffa864765fa System.Diagnostics.Debugger.Break()
0000001fc697d700 00007ffa864736bc example.Program.Main(System.String[])
[E:\unloadability\example\Program.cs @ 70]
0000001fc697d998 00007ffae5fdc1e3 [GCFrame: 0000001fc697d998]
0000001fc697df28 00007ffae5fdc1e3 [GCFrame: 0000001fc697df28]
OS Thread Id: 0x2ae4 (1)
Unable to walk the managed stack. The current thread is likely not a
managed thread. You can run !threads to get a list of managed threads in
the process
Failed to start stack walk: 80070057
OS Thread Id: 0x61a4 (2)
Unable to walk the managed stack. The current thread is likely not a
managed thread. You can run !threads to get a list of managed threads in
the process
Failed to start stack walk: 80070057
OS Thread Id: 0x7fdc (3)
Unable to walk the managed stack. The current thread is likely not a
managed thread. You can run !threads to get a list of managed threads in
the process
Failed to start stack walk: 80070057
OS Thread Id: 0x5390 (4)
Unable to walk the managed stack. The current thread is likely not a
managed thread. You can run !threads to get a list of managed threads in
the process
Failed to start stack walk: 80070057
OS Thread Id: 0x5ec8 (5)
    Child SP          IP Call Site
0000001fc70ff6e0 00007ffb5437f6e4 [DebuggerU2MCatchHandlerFrame: 0000001fc70ff6e0]
OS Thread Id: 0x4624 (6)
    Child SP          IP Call Site
GetFrameContext failed: 1
0000000000000000 0000000000000000
OS Thread Id: 0x60bc (7)
    Child SP          IP Call Site
0000001fc727f158 00007ffb5437fce4 [HelperMethodFrame: 0000001fc727f158]
System.Threading.Thread.SleepInternal(Int32)
0000001fc727f260 00007ffb37ea7c2b System.Threading.Thread.Sleep(Int32)
0000001fc727f290 00007ffa865005b3 test.Program.ThreadProc() [E:\unloadability\test\Program.cs @ 17]
0000001fc727f2c0 00007ffb37ea6a5b System.Threading.Thread.ThreadMain_ThreadStart()
0000001fc727f2f0 00007ffadbc4cbe3
System.Threading.ExecutionContext.RunInternal(System.Threading.ExecutionContext,
System.Threading.ContextCallback, System.Object)
0000001fc727f568 00007ffae5fdc1e3 [GCFrame: 0000001fc727f568]
0000001fc727f7f0 00007ffae5fdc1e3 [DebuggerU2MCatchHandlerFrame: 0000001fc727f7f0]

```

As you can see, the last thread has `test.Program.ThreadProc()`. This is a function from the assembly loaded into the `AssemblyLoadContext`, and so it keeps the `AssemblyLoadContext` alive.

Example source with unloadability issues

The following code is used in the previous debugging example.

Main testing program

```

using System;
using System.Reflection;
using System.Runtime.CompilerServices;
using System.Runtime.Loader;

namespace example

```

```

{
    class TestAssemblyLoadContext : AssemblyLoadContext
    {
        public TestAssemblyLoadContext() : base(true)
        {
        }
        protected override Assembly Load(AssemblyName name)
        {
            return null;
        }
    }

    class TestInfo
    {
        public TestInfo(MethodInfo mi)
        {
            entryPoint = mi;
        }
        MethodInfo entryPoint;
    }

    class Program
    {
        static TestInfo entryPoint;

        [MethodImpl(MethodImplOptions.NoInlining)]
        static int ExecuteAndUnload(string assemblyPath, out WeakReference testAlcWeakRef, out MethodInfo
testEntryPoint)
        {
            var alc = new TestAssemblyLoadContext();
            testAlcWeakRef = new WeakReference(alc);

            Assembly a = alc.LoadFromAssemblyPath(assemblyPath);
            if (a == null)
            {
                testEntryPoint = null;
                Console.WriteLine("Loading the test assembly failed");
                return -1;
            }

            var args = new object[1] {new string[] {"Hello"}};

            // Issue preventing unloading #1 - we keep MethodInfo of a method for an assembly loaded into
            the TestAssemblyLoadContext in a static variable
            entryPoint = new TestInfo(a.EntryPoint);
            testEntryPoint = a.EntryPoint;

            int result = (int)a.EntryPoint.Invoke(null, args);
            alc.Unload();

            return result;
        }

        static void Main(string[] args)
        {
            WeakReference testAlcWeakRef;
            // Issue preventing unloading #2 - we keep MethodInfo of a method for an assembly loaded into
            the TestAssemblyLoadContext in a local variable
            MethodInfo testEntryPoint;
            int result = ExecuteAndUnload(@"absolute/path/to/test.dll", out testAlcWeakRef, out
testEntryPoint);

            for (int i = 0; testAlcWeakRef.IsAlive && (i < 10); i++)
            {
                GC.Collect();
                GC.WaitForPendingFinalizers();
            }

            System.Diagnostics.Debugger.Break();
        }
    }
}

```

```
System.Diagnostics.Debugger.OnBreak();  
  
        Console.WriteLine($"Test completed, result={result}, entryPoint: {testEntryPoint} unload  
success: {!testAlcWeakRef.IsAlive}");  
    }  
}  
}
```

Program loaded into the TestAssemblyLoadContext

The following code represents the *test.dll* passed to the `ExecuteAndUnload` method in the main testing program.

```
using System;  
using System.Runtime.InteropServices;  
using System.Threading;  
  
namespace test  
{  
    class Test  
    {  
        string message = "Hello";  
    }  
  
    class Program  
    {  
        public static void ThreadProc()  
        {  
            // Issue preventing unloading #4 - a thread running method inside of the TestAssemblyLoadContext  
            // at the unload time  
            Thread.Sleep(Timeout.Infinite);  
        }  
  
        static GCHandle handle;  
        static int Main(string[] args)  
        {  
            // Issue preventing unloading #3 - normal GC handle  
            handle = GCHandle.Alloc(new Test());  
            Thread t = new Thread(new ThreadStart(ThreadProc));  
            t.IsBackground = true;  
            t.Start();  
            Console.WriteLine($"Hello from the test: args[0] = {args[0]}");  
  
            return 1;  
        }  
    }  
}
```

Overview of how .NET is versioned

9/20/2022 • 3 minutes to read • [Edit Online](#)

The [.NET Runtime](#) and the [.NET SDK](#) add new features at different frequencies. In general, the SDK is updated more frequently than the Runtime. This article explains the runtime and the SDK version numbers.

.NET releases a new major version every November. Even-numbered releases, such as .NET 6 or .NET 8, are long-term supported (LTS). Odd-numbered releases are supported until the next major release. The latest release of .NET is .NET 6.

Versioning details

The .NET Runtime has a major:minor:patch approach to versioning that follows [semantic versioning](#).

The .NET SDK, however, doesn't follow semantic versioning. The .NET SDK releases faster and its version numbers must communicate both the aligned runtime and the SDK's own minor and patch releases.

The first two positions of the .NET SDK version number match the .NET Runtime version it released with. Each version of the SDK can create applications for this runtime or any lower version.

The third position of the SDK version number communicates both the minor and patch number. The minor version is multiplied by 100. The final two digits represent the patch number. Minor version 1, patch version 2 would be represented as 102. For example, here's a possible sequence of runtime and SDK version numbers:

CHANGE	.NET RUNTIME	.NET SDK (*)
Initial release	5.0.0	5.0.100
SDK patch	5.0.0	5.0.101
Runtime and SDK patch	5.0.1	5.0.102
SDK feature change	5.0.1	5.0.200

NOTES:

- If the SDK has 10 feature updates before a runtime feature update, version numbers roll into the 1000 series. Version 5.0.1000 would follow version 5.0.900. This situation isn't expected to occur.
- 99 patch releases without a feature release won't occur. If a release approaches this number, it forces a feature release.

You can see more details in the initial proposal at the [dotnet/designs](#) repository.

Semantic versioning

The .NET *Runtime* roughly adheres to [Semantic Versioning \(SemVer\)](#), adopting the use of `MAJOR.MINOR.PATCH` versioning, using the various parts of the version number to describe the degree and type of change.

`MAJOR.MINOR.PATCH[-PRERELEASE-BUILDDNUMBER]`

The optional `PRERELEASE` and `BUILDDNUMBER` parts are never part of supported releases and only exist on nightly

builds, local builds from source targets, and unsupported preview releases.

Understand runtime version number changes

- `MAJOR` is incremented once a year and may contain:
 - Significant changes in the product, or a new product direction.
 - API introduced breaking changes. There's a high bar to accepting breaking changes.
 - A newer `MAJOR` version of an existing dependency is adopted.
- Major releases happen once a year; even-numbered versions are long-term supported (LTS) releases. The first LTS release using this versioning scheme is .NET 6. The latest non-LTS version is .NET 5.
- `MINOR` is incremented when:
 - Public API surface area is added.
 - A new behavior is added.
 - A newer `MINOR` version of an existing dependency is adopted.
 - A new dependency is introduced.
- `PATCH` is incremented when:
 - Bug fixes are made.
 - Support for a newer platform is added.
 - A newer `PATCH` version of an existing dependency is adopted.
 - Any other change doesn't fit one of the previous cases.

When there are multiple changes, the highest element affected by individual changes is incremented, and the following ones are reset to zero. For example, when `MAJOR` is incremented, `MINOR.PATCH` are reset to zero. When `MINOR` is incremented, `PATCH` is reset to zero while `MAJOR` remains the same.

Version numbers in file names

The files downloaded for .NET carry the version, for example, `dotnet-sdk-5.0.301-win10-x64.exe`.

Preview versions

Preview versions have a `-preview.[number].[build]` appended to the version number. For example, `6.0.0-preview.5.21302.13`.

Servicing versions

After a release goes out, the release branches generally stop producing daily builds and instead start producing servicing builds. Servicing versions have a `-servicing-[number]` appended to the version. For example, `5.0.1-servicing-006924`.

See also

- [Target frameworks](#)
- [.NET distribution packaging](#)
- [.NET Support Lifecycle Fact Sheet](#)
- [Docker images for .NET](#)

Select the .NET version to use

9/20/2022 • 7 minutes to read • [Edit Online](#)

This article explains the policies used by the .NET tools, SDK, and runtime for selecting versions. These policies provide a balance between running applications using the specified versions and enabling ease of upgrading both developer and end-user machines. These policies enable:

- Easy and efficient deployment of .NET, including security and reliability updates.
- Use the latest tools and commands independent of target runtime.

Version selection occurs:

- When you run an SDK command, the **SDK uses the latest installed version**.
- When you build an assembly, **target framework monikers define build time APIs**.
- When you run a .NET application, **target framework dependent apps roll-forward**.
- When you publish a self-contained application, **self-contained deployments include the selected runtime**.

The rest of this document examines those four scenarios.

The SDK uses the latest installed version

SDK commands include `dotnet new` and `dotnet run`. The .NET CLI must choose an SDK version for every `dotnet` command. It uses the latest SDK installed on the machine by default, even if:

- The project targets an earlier version of the .NET runtime.
- The latest version of the .NET SDK is a preview version.

You can take advantage of the latest SDK features and improvements while targeting earlier .NET runtime versions. You can target different runtime versions of .NET using the same SDK tools.

On rare occasions, you may need to use an earlier version of the SDK. You specify that version in a [*global.json* file](#). The "use latest" policy means you only use *global.json* to specify a .NET SDK version earlier than the latest installed version.

global.json can be placed anywhere in the file hierarchy. The CLI searches upward from the project directory for the first *global.json* it finds. You control which projects a given *global.json* applies to by its place in the file system. The .NET CLI searches for a *global.json* file iteratively navigating the path upward from the current working directory. The first *global.json* file found specifies the version used. If that SDK version is installed, that version is used. If the SDK specified in the *global.json* isn't found, the .NET CLI uses [matching rules](#) to select a compatible SDK, or fails if none is found.

The following example shows the *global.json* syntax:

```
{  
  "sdk": {  
    "version": "5.0.0"  
  }  
}
```

The process for selecting an SDK version is:

1. `dotnet` searches for a *global.json* file iteratively reverse-navigating the path upward from the current working directory.

2. `dotnet` uses the SDK specified in the first `global.json` found.
3. `dotnet` uses the latest installed SDK if no `global.json` is found.

For more information about SDK version selection, see the [Matching rules](#) and [rollForward](#) sections of the [global.json overview](#) article.

Target Framework Monikers define build time APIs

You build your project against APIs defined in a **Target Framework Moniker** (TFM). You specify the [target framework](#) in the project file. Set the `TargetFramework` element in your project file as shown in the following example:

```
<TargetFramework>net5.0</TargetFramework>
```

You may build your project against multiple TFMs. Setting multiple target frameworks is more common for libraries but can be done with applications as well. You specify a `TargetFrameworks` property (plural of `TargetFramework`). The target frameworks are semicolon-delimited as shown in the following example:

```
<TargetFrameworks>net5.0;netcoreapp3.1;net47</TargetFrameworks>
```

A given SDK supports a fixed set of frameworks, capped to the target framework of the runtime it ships with. For example, the .NET 5 SDK includes the .NET 5 runtime, which is an implementation of the `net5.0` target framework. The .NET 5 SDK supports `netcoreapp2.0`, `netcoreapp2.1`, `netcoreapp3.0`, and so on, but not `net6.0` (or higher). You install the .NET 6 SDK to build for `net6.0`.

.NET Standard

.NET Standard was a way to target an API surface shared by different implementations of .NET. Starting with the release of .NET 5, which is an API standard itself, .NET Standard has little relevance, except for one scenario: .NET Standard is useful when you want to target both .NET and .NET Framework. .NET 5 implements all .NET Standard versions.

For more information, see [.NET 5 and .NET Standard](#).

Framework-dependent apps roll-forward

When you run an application from source with `dotnet run`, from a [framework-dependent deployment](#) with `dotnet myapp.dll`, or from a [framework-dependent executable](#) with `myapp.exe`, the `dotnet` executable is the **host** for the application.

The host chooses the latest patch version installed on the machine. For example, if you specified `net5.0` in your project file, and `5.0.2` is the latest .NET runtime installed, the `5.0.2` runtime is used.

If no acceptable `5.0.*` version is found, a new `5.*` version is used. For example, if you specified `net5.0` and only `5.1.0` is installed, the application runs using the `5.1.0` runtime. This behavior is referred to as "minor version roll-forward." Lower versions also won't be considered. When no acceptable runtime is installed, the application won't run.

A few usage examples demonstrate the behavior, if you target 5.0:

- ✓ 5.0 is specified. 5.0.3 is the highest patch version installed. 5.0.3 is used.
- ✗ 5.0 is specified. No 5.0.* versions are installed. 3.1.1 is the highest runtime installed. An error message is displayed.
- ✓ 5.0 is specified. No 5.0.* versions are installed. 5.1.0 is the highest runtime version installed. 5.1.0 is used.

- ✗ 3.0 is specified. No 3.x versions are installed. 5.0.0 is the highest runtime installed. An error message is displayed.

Minor version roll-forward has one side-effect that may affect end users. Consider the following scenario:

1. The application specifies that 5.0 is required.
2. When run, version 5.0.* isn't installed, however, 5.1.0 is. Version 5.1.0 will be used.
3. Later, the user installs 5.0.3 and runs the application again, 5.0.3 will now be used.

It's possible that 5.0.3 and 5.1.0 behave differently, particularly for scenarios like serializing binary data.

Control roll-forward behavior

The roll-forward behavior for an application can be configured in four different ways:

1. Project-level setting by setting the `<RollForward>` property:

```
<PropertyGroup>
  <RollForward>LatestMinor</RollForward>
</PropertyGroup>
```

2. The `*.runtimeconfig.json` file.

This file is produced when you compile your application. If the `<RollForward>` property was set in the project, it's reproduced in the `*.runtimeconfig.json` file as the `rollForward` setting. Users can edit this file to change the behavior of your application.

```
{
  "runtimeOptions": {
    "tfm": "net5.0",
    "rollForward": "LatestMinor",
    "framework": {
      "name": "Microsoft.NETCore.App",
      "version": "5.0.0"
    }
  }
}
```

3. The `dotnet` command's `--roll-forward <value>` property.

When you run an application, you can control the roll-forward behavior through the command line:

```
dotnet run --roll-forward LatestMinor
dotnet myapp.dll --roll-forward LatestMinor
myapp.exe --roll-forward LatestMinor
```

4. The `DOTNET_ROLL_FORWARD` environment variable.

Precedence

Roll forward behavior is set by the following order when your app is run, higher numbered items taking precedence over lower numbered items:

1. First the `*.runtimeconfig.json` config file is evaluated.
2. Next, the `DOTNET_ROLL_FORWARD` environment variable is considered, overriding the previous check.
3. Finally, any `--roll-forward` parameter passed to the running application overrides everything else.

Values

However you set the roll-forward setting, use one of the following values to set the behavior:

Value	Description
Minor	Default if not specified. Roll-forward to the lowest higher minor version, if requested minor version is missing. If the requested minor version is present, then the <code>LatestPatch</code> policy is used.
Major	Roll-forward to the next available higher major version, and lowest minor version, if requested major version is missing. If the requested major version is present, then the <code>Minor</code> policy is used.
LatestPatch	Roll-forward to the highest patch version. This value disables minor version roll-forward.
LatestMinor	Roll-forward to highest minor version, even if requested minor version is present.
LatestMajor	Roll-forward to highest major and highest minor version, even if requested major is present.
Disable	Don't roll-forward, only bind to the specified version. This policy isn't recommended for general use since it disables the ability to roll-forward to the latest patches. This value is only recommended for testing.

Self-contained deployments include the selected runtime

You can publish an application as a [self-contained distribution](#). This approach bundles the .NET runtime and libraries with your application. Self-contained deployments don't have a dependency on runtime environments. Runtime version selection occurs at publishing time, not run time.

The `restore` event that occurs when publishing selects the latest patch version of the given runtime family. For example, `dotnet publish` will select .NET 5.0.3 if it's the latest patch version in the .NET 5 runtime family. The target framework (including the latest installed security patches) is packaged with the application.

An error occurs if the minimum version specified for an application isn't satisfied. `dotnet publish` binds to the latest runtime patch version (within a given major.minor version family). `dotnet publish` doesn't support the roll-forward semantics of `dotnet run`. For more information about patches and self-contained deployments, see the article on [runtime patch selection](#) in deploying .NET applications.

Self-contained deployments may require a specific patch version. You can override the minimum runtime patch version (to higher or lower versions) in the project file, as shown in the following example:

```
<PropertyGroup>
  <RuntimeFrameworkVersion>5.0.7</RuntimeFrameworkVersion>
</PropertyGroup>
```

The `RuntimeFrameworkVersion` element overrides the default version policy. For self-contained deployments, the `RuntimeFrameworkVersion` specifies the *exact* runtime framework version. For framework-dependent applications, the `RuntimeFrameworkVersion` specifies the *minimum* required runtime framework version.

See also

- [Download and install .NET](#).

- How to remove the .NET Runtime and SDK.

.NET Runtime configuration settings

9/20/2022 • 3 minutes to read • [Edit Online](#)

.NET 5+ (including .NET Core versions) supports the use of configuration files and environment variables to configure the behavior of .NET applications at run time.

NOTE

The articles in this section concern configuration of the .NET Runtime itself. If you're migrating to .NET Core 3.1 or later and are looking for a replacement for the *app.config* file, or if you simply want a way to use custom configuration values in your .NET app, see the [Microsoft.Extensions.Configuration.ConfigurationBuilder](#) class and [Configuration in .NET](#).

Using these settings is an attractive option if:

- You don't own or control the source code for an application and therefore are unable to configure it programmatically.
- Multiple instances of your application run at the same time on a single system, and you want to configure each for optimum performance.

.NET provides the following mechanisms for configuring behavior of the .NET runtime:

- The [runtimeconfig.json](#) file
- [MSBuild](#) properties
- [Environment variables](#)

TIP

Configuring an option by using an environment variable applies the setting to all .NET apps. Configuring an option in the *runtimeconfig.json* or project file applies the setting to that application only.

Some configuration values can also be set programmatically by calling the [AppContext.SetSwitch](#) method.

The articles in this section of the documentation are organized by category, for example, [debugging](#) and [garbage collection](#). Where applicable, configuration options are shown for *runtimeconfig.json* files, MSBuild properties, environment variables, and, for cross-reference, *app.config* files for .NET Framework projects.

runtimeconfig.json

When a project is [built](#), an *[appname].runtimeconfig.json* file is generated in the output directory. If a *runtimeconfig.template.json* file exists in the same folder as the project file, any configuration options it contains are inserted into the *[appname].runtimeconfig.json* file. If you're building the app yourself, put any configuration options in the *runtimeconfig.template.json* file. If you're just running the app, insert them directly into the *[appname].runtimeconfig.json* file.

NOTE

- The `[appname].runtimeconfig.json` file will get overwritten on subsequent builds.
- If your app's `OutputType` is not `Exe` and you want configuration options to be copied from `runtimeconfig.template.json` to `[appname].runtimeconfig.json`, you must explicitly set `GenerateRuntimeConfigurationFiles` to `true` in your project file. For apps that require a `runtimeconfig.json` file, this property defaults to `true`.

Specify runtime configuration options in the `configProperties` section of the `runtimeconfig.json` files. This section has the form:

```
"configProperties": {  
    "config-property-name1": "config-value1",  
    "config-property-name2": "config-value2"  
}
```

Example `[appname].runtimeconfig.json` file

If you're placing the options in the output JSON file, nest them under the `runtimeOptions` property.

```
{  
    "runtimeOptions": {  
        "tfm": "netcoreapp3.1",  
        "framework": {  
            "name": "Microsoft.NETCore.App",  
            "version": "3.1.0"  
        },  
        "configProperties": {  
            "System.Globalization.UseNls": true,  
            "System.Net.DisableIPv6": true,  
            "System.GC.Concurrent": false,  
            "System.Threading.ThreadPool.MinThreads": 4,  
            "System.Threading.ThreadPool.MaxThreads": 25  
        }  
    }  
}
```

Example `runtimeconfig.template.json` file

If you're placing the options in the template JSON file, omit the `runtimeOptions` property.

```
{  
    "configProperties": {  
        "System.Globalization.UseNls": true,  
        "System.Net.DisableIPv6": true,  
        "System.GC.Concurrent": false,  
        "System.Threading.ThreadPool.MinThreads": "4",  
        "System.Threading.ThreadPool.MaxThreads": "25"  
    }  
}
```

MSBuild properties

Some runtime configuration options can be set using MSBuild properties in the `.csproj` or `.vbproj` file of SDK-style .NET Core projects. MSBuild properties take precedence over options set in the `runtimeconfig.template.json` file.

For runtime configuration settings that don't have a specific MSBuild property, you can use the

`RuntimeHostConfigurationOption` MSBuild item instead.

Here is an example SDK-style project file with MSBuild properties for configuring run-time behavior:

```
<Project Sdk="Microsoft.NET.Sdk">

<PropertyGroup>
  <OutputType>Exe</OutputType>
  <TargetFramework>netcoreapp3.1</TargetFramework>
</PropertyGroup>

<PropertyGroup>
  <ConcurrentGarbageCollection>false</ConcurrentGarbageCollection>
  <ThreadPoolMinThreads>4</ThreadPoolMinThreads>
  <ThreadPoolMaxThreads>25</ThreadPoolMaxThreads>
</PropertyGroup>

<ItemGroup>
  <RuntimeHostConfigurationOption Include="System.Globalization.UseNls" Value="true" />
  <RuntimeHostConfigurationOption Include="System.Net.DisableIPv6" Value="true" />
</ItemGroup>

</Project>
```

MSBuild properties for configuring the behavior of the runtime are noted in the individual articles for each area, for example, [garbage collection](#). They are also listed in the [Runtime configuration](#) section of the MSBuild properties reference for SDK-style projects.

Environment variables

Environment variables can be used to supply some runtime configuration information. Configuring a run-time option by using an environment variable applies the setting to all .NET Core apps. Configuration knobs specified as environment variables generally have the prefix `DOTNET_`.

NOTE

.NET 6 standardizes on the prefix `DOTNET_` instead of `COMPlus_` for environment variables that configure .NET run-time behavior. However, the `COMPlus_` prefix will continue to work. If you're using a previous version of the .NET runtime, you should still use the `COMPlus_` prefix for environment variables.

You can define environment variables from the Windows Control Panel, at the command line, or programmatically by calling the [Environment.SetEnvironmentVariable\(String, String\)](#) method on both Windows and Unix-based systems.

The following examples show how to set an environment variable at the command line:

```
# Windows
set DOTNET_GCRetainVM=1

# Powershell
$env:DOTNET_GCRetainVM="1"

# Unix
export DOTNET_GCRetainVM=1
```

See also

- [.NET environment variables](#)

Runtime configuration options for compilation

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article details the settings you can use to configure .NET compilation.

NOTE

.NET 6 standardizes on the prefix `DOTNET_` instead of `COMPlus_` for environment variables that configure .NET run-time behavior. However, the `COMPlus_` prefix will continue to work. If you're using a previous version of the .NET runtime, you should still use the `COMPlus_` prefix for environment variables.

Tiered compilation

- Configures whether the just-in-time (JIT) compiler uses [tiered compilation](#). Tiered compilation transitions methods through two tiers:
 - The first tier generates code more quickly ([quick JIT](#)) or loads pre-compiled code ([ReadyToRun](#)).
 - The second tier generates optimized code in the background ("optimizing JIT").
- In .NET Core 3.0 and later, tiered compilation is enabled by default.
- In .NET Core 2.1 and 2.2, tiered compilation is disabled by default.
- For more information, see the [Tiered compilation guide](#).

	SETTING NAME	VALUES
<code>runtimesconfig.json</code>	<code>System.Runtime.TieredCompilation</code>	<code>true</code> - enabled <code>false</code> - disabled
<code>MSBuild</code> property	<code>TieredCompilation</code>	<code>true</code> - enabled <code>false</code> - disabled
Environment variable	<code>COMPlus_TieredCompilation</code> or <code>DOTNET_TieredCompilation</code>	<code>1</code> - enabled <code>0</code> - disabled

Examples

`runtimesconfig.json` file:

```
{  
  "runtimeOptions": {  
    "configProperties": {  
      "System.Runtime.TieredCompilation": false  
    }  
  }  
}
```

Project file:

```

<Project Sdk="Microsoft.NET.Sdk">

<PropertyGroup>
    <TieredCompilation>false</TieredCompilation>
</PropertyGroup>

</Project>

```

Quick JIT

- Configures whether the JIT compiler uses *quick JIT*. For methods that don't contain loops and for which pre-compiled code is not available, quick JIT compiles them more quickly but without optimizations.
- Enabling quick JIT decreases startup time but can produce code with degraded performance characteristics. For example, the code may use more stack space, allocate more memory, and run slower.
- If quick JIT is disabled but [tiered compilation](#) is enabled, only pre-compiled code participates in tiered compilation. If a method is not pre-compiled with [ReadyToRun](#), the JIT behavior is the same as if [tiered compilation](#) were disabled.
- In .NET Core 3.0 and later, quick JIT is enabled by default.
- In .NET Core 2.1 and 2.2, quick JIT is disabled by default.

	SETTING NAME	VALUES
runtimesconfig.json	System.Runtime.TieredCompilation.QuickJit	<input type="checkbox"/> true - enabled <input type="checkbox"/> false - disabled
MSBuild property	TieredCompilationQuickJit	<input type="checkbox"/> true - enabled <input type="checkbox"/> false - disabled
Environment variable	COMPlus_TC_QuickJit or DOTNET_TC_QuickJit	<input type="checkbox"/> 1 - enabled <input type="checkbox"/> 0 - disabled

Examples

runtimesconfig.json file:

```
{
  "runtimeOptions": {
    "configProperties": {
      "System.Runtime.TieredCompilation.QuickJit": false
    }
  }
}
```

Project file:

```

<Project Sdk="Microsoft.NET.Sdk">

<PropertyGroup>
    <TieredCompilationQuickJit>false</TieredCompilationQuickJit>
</PropertyGroup>

</Project>

```

Quick JIT for loops

- Configures whether the JIT compiler uses quick JIT on methods that contain loops.
- Enabling quick JIT for loops may improve startup performance. However, long-running loops can get stuck in less-optimized code for long periods.
- If [quick JIT](#) is disabled, this setting has no effect.
- If you omit this setting, quick JIT is not used for methods that contain loops. This is equivalent to setting the value to `false`.

	SETTING NAME	VALUES
runtimeconfig.json	<code>System.Runtime.TieredCompilation.QuickJitForLoops</code>	<code>false</code> - disabled <code>true</code> - enabled
MSBuild property	<code>TieredCompilationQuickJitForLoops</code>	<code>false</code> - disabled <code>true</code> - enabled
Environment variable	<code>COMPlus_TC_QuickJitForLoops</code> or <code>DOTNET_TC_QuickJitForLoops</code>	<code>0</code> - disabled <code>1</code> - enabled

Examples

runtimeconfig.json file:

```
{
  "runtimeOptions": {
    "configProperties": {
      "System.Runtime.TieredCompilation.QuickJitForLoops": false
    }
  }
}
```

Project file:

```
<Project Sdk="Microsoft.NET.Sdk">

<PropertyGroup>
  <TieredCompilationQuickJitForLoops>true</TieredCompilationQuickJitForLoops>
</PropertyGroup>

</Project>
```

ReadyToRun

- Configures whether the .NET Core runtime uses pre-compiled code for images with available ReadyToRun data. Disabling this option forces the runtime to JIT-compile framework code.
- For more information, see [Ready to Run](#).
- If you omit this setting, .NET uses ReadyToRun data when it's available. This is equivalent to setting the value to `1`.

	SETTING NAME	VALUES
Environment variable	<code>COMPlus_ReadyToRun</code> or <code>DOTNET_ReadyToRun</code>	<code>1</code> - enabled <code>0</code> - disabled

Profile-guided optimization

This setting enables dynamic or tiered profile-guided optimization (PGO) in .NET 6 and later versions.

	SETTING NAME	VALUES
Environment variable	<code>DOTNET_TieredPGO</code>	<input checked="" type="checkbox"/> 1 - enabled <input type="checkbox"/> 0 - disabled

Runtime configuration options for debugging and profiling

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article details the settings you can use to configure .NET debugging and profiling.

NOTE

.NET 6 standardizes on the prefix `DOTNET_` instead of `COMPlus_` for environment variables that configure .NET run-time behavior. However, the `COMPlus_` prefix will continue to work. If you're using a previous version of the .NET runtime, you should still use the `COMPlus_` prefix for environment variables.

Enable diagnostics

- Configures whether the debugger, the profiler, and EventPipe diagnostics are enabled or disabled.
- If you omit this setting, diagnostics are enabled. This is equivalent to setting the value to `1`.

	SETTING NAME	VALUES
<code>runtimeconfig.json</code>	N/A	N/A
Environment variable	<code>COMPlus_EnableDiagnostics</code> or <code>DOTNET_EnableDiagnostics</code>	<code>1</code> - enabled <code>0</code> - disabled

Enable profiling

- Configures whether profiling is enabled for the currently running process.
- If you omit this setting, profiling is disabled. This is equivalent to setting the value to `0`.

	SETTING NAME	VALUES
<code>runtimeconfig.json</code>	N/A	N/A
Environment variable	<code>CORECLR_ENABLE_PROFILING</code>	<code>0</code> - disabled <code>1</code> - enabled

Profiler GUID

- Specifies the GUID of the profiler to load into the currently running process.

	SETTING NAME	VALUES
<code>runtimeconfig.json</code>	N/A	N/A
Environment variable	<code>CORECLR_PROFILER</code>	<code>string-guid</code>

Profiler location

- Specifies the path to the profiler DLL to load into the currently running process (or 32-bit or 64-bit process).
- If more than one variable is set, the bitness-specific variables take precedence. They specify which bitness of profiler to load.
- For more information, see [Finding the profiler library](#).

	SETTING NAME	VALUES
Environment variable	CORECLR_PROFILER_PATH	<i>string-path</i>
Environment variable	CORECLR_PROFILER_PATH_32	<i>string-path</i>
Environment variable	CORECLR_PROFILER_PATH_64	<i>string-path</i>

Export perf maps

- Enables or disables emitting perf maps to `/tmp/perf-$pid.map`. Perf maps allow third party tools, such as perf, to identify call sites from precompiled ReadyToRun (R2R) modules.
- If you omit this setting, writing the perf map is disabled. This is equivalent to setting the value to `0`.
- When perf maps are disabled, not all managed callsites will be properly resolved.
- Enabling perf maps causes a 10-20% overhead.

	SETTING NAME	VALUES
runtimeconfig.json	N/A	N/A
Environment variable	COMPlus_PerfMapEnabled or DOTNET_PerfMapEnabled	<code>0</code> - disabled <code>1</code> - enabled

Perf log markers

- Enables or disables the specified signal to be accepted and ignored as a marker in the perf logs.
- If you omit this setting, the specified signal is not ignored. This is equivalent to setting the value to `0`.

	SETTING NAME	VALUES
runtimeconfig.json	N/A	N/A
Environment variable	COMPlus_PerfMapIgnoreSignal or DOTNET_PerfMapIgnoreSignal	<code>0</code> - disabled <code>1</code> - enabled

NOTE

This setting is ignored if `DOTNET_PerfMapEnabled` is omitted or set to `0` (that is, disabled).

Runtime configuration options for garbage collection

9/20/2022 • 17 minutes to read • [Edit Online](#)

This page contains information about settings for the .NET runtime garbage collector (GC). If you're trying to achieve peak performance of a running app, consider using these settings. However, the defaults provide optimum performance for most applications in typical situations.

Settings are arranged into groups on this page. The settings within each group are commonly used in conjunction with each other to achieve a specific result.

NOTE

- These settings can also be changed dynamically by the app as it's running, so any configuration options you set may be overridden.
- Some settings, such as [latency level](#), are typically set only through the API at design time. Such settings are omitted from this page.
- For number values, use decimal notation for settings in the *runtimeconfig.json* file and hexadecimal notation for environment variable settings. For hexadecimal values, you can specify them with or without the "0x" prefix.
- If you're using the environment variables, .NET 6 standardizes on the prefix `DOTNET_` instead of `COMPlus_`. However, the `COMPlus_` prefix will continue to work. If you're using a previous version of the .NET runtime, you should still use the `COMPlus_` prefix, for example, `COMPlus_gcServer`.

Ways to specify the configuration

For different versions of the .NET runtime, there are different ways to specify the configuration values. The following table shows a summary.

CONFIG LOCATION	.NET VERSIONS THIS LOCATION APPLIES TO	FORMATS	HOW IT'S INTERPRETED
runtimeconfig.json file	.NET Core	n	n is interpreted as a decimal value.
Environment variable	.NET Framework, .NET Core	0xn or n	n is interpreted as a hex value in either format
app.config file	.NET Framework	0xn	n is interpreted as a hex value ¹

¹ You can specify a value without the `0x` prefix for an app.config file setting, but it's not recommended. On .NET Framework 4.8+, due to a bug, a value specified without the `0x` prefix is interpreted as hexadecimal, but on previous versions of .NET Framework, it's interpreted as decimal. To avoid having to change your config, use the `0x` prefix when specifying a value in your app.config file.

For example, to specify 12 heaps for `GCHeapCount` for a .NET Framework app named *A.exe*, add the following XML to the *A.exe.config* file.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  ...
  <runtime>
    <gcServer enabled="true"/>
    <GCHeapCount>0xc</GCHeapCount>
  </runtime>
</configuration>
```

For both .NET Core and .NET Framework, you can use environment variables.

On Windows using .NET 5 or a later version:

```
SET DOTNET_gcServer=1
SET DOTNET_GCHeapCount=c
```

On Windows using .NET Core 3.1 or earlier:

```
SET COMPlus_gcServer=1
SET COMPlus_GCHeapCount=c
```

On other operating systems:

For .NET 5 or later versions:

```
export DOTNET_gcServer=1
export DOTNET_GCHeapCount=c
```

For .NET Core 3.1 and earlier versions:

```
export COMPlus_gcServer=1
export COMPlus_GCHeapCount=c
```

For .NET Core only, you can set the value in the *runtimeconfig.json* file.

```
{
  "runtimeOptions": {
    "configProperties": {
      "System.GC.Server": true,
      "System.GC.HeapCount": 12
    }
  }
}
```

Flavors of garbage collection

The two main flavors of garbage collection are workstation GC and server GC. For more information about differences between the two, see [Workstation and server garbage collection](#).

The subflavors of garbage collection are background and non-concurrent.

Use the following settings to select flavors of garbage collection:

- [Workstation vs. server GC](#)
- [Background GC](#)

Workstation vs. server

- Configures whether the application uses workstation garbage collection or server garbage collection.
- Default: Workstation garbage collection. This is equivalent to setting the value to `false`.

	SETTING NAME	VALUES	VERSION INTRODUCED
runtimeconfig.json	<code>System.GC.Server</code>	<code>false</code> - workstation <code>true</code> - server	.NET Core 1.0
MSBuild property	<code>ServerGarbageCollection</code>	<code>false</code> - workstation <code>true</code> - server	.NET Core 1.0
Environment variable	<code>COMPlus_gcServer</code>	<code>0</code> - workstation <code>1</code> - server	.NET Core 1.0
Environment variable	<code>DOTNET_gcServer</code>	<code>0</code> - workstation <code>1</code> - server	.NET 6
app.config for .NET Framework	<code>GCSERVER</code>	<code>false</code> - workstation <code>true</code> - server	

Examples

runtimeconfig.json file:

```
{
  "runtimeOptions": {
    "configProperties": {
      "System.GC.Server": true
    }
  }
}
```

Project file:

```
<Project Sdk="Microsoft.NET.Sdk">

<PropertyGroup>
  <ServerGarbageCollection>true</ServerGarbageCollection>
</PropertyGroup>

</Project>
```

Background GC

- Configures whether background (concurrent) garbage collection is enabled.
- Default: Use background GC. This is equivalent to setting the value to `true`.
- For more information, see [Background garbage collection](#).

	SETTING NAME	VALUES	VERSION INTRODUCED
runtimeconfig.json	<code>System.GC.Concurrent</code>	<code>true</code> - background GC <code>false</code> - non-concurrent GC	.NET Core 1.0

	SETTING NAME	VALUES	VERSION INTRODUCED
MSBuild property	ConcurrentGarbageCollection	true - background GC false - non-concurrent GC	.NET Core 1.0
Environment variable	COMPlus_gcConcurrent	1 - background GC 0 - non-concurrent GC	.NET Core 1.0
Environment variable	DOTNET_gcConcurrent	1 - background GC 0 - non-concurrent GC	.NET 6
app.config for .NET Framework	gcConcurrent	true - background GC false - non-concurrent GC	

Examples

runtimeconfig.json file:

```
{
  "runtimeOptions": {
    "configProperties": {
      "System.GC.Concurrent": false
    }
  }
}
```

Project file:

```
<Project Sdk="Microsoft.NET.Sdk">

<PropertyGroup>
  <ConcurrentGarbageCollection>false</ConcurrentGarbageCollection>
</PropertyGroup>

</Project>
```

Manage resource usage

Use the following settings to manage the garbage collector's memory and processor usage:

- [Affinize](#)
- [Affinize mask](#)
- [Affinize ranges](#)
- [CPU groups](#)
- [Heap count](#)
- [Heap limit](#)
- [Heap limit percent](#)
- [High memory percent](#)
- [Per-object-heap limits](#)
- [Per-object-heap limit percents](#)
- [Retain VM](#)

For more information about some of these settings, see the [Middle ground between workstation and server GC](#)

blog entry.

Heap count

- Limits the number of heaps created by the garbage collector.
- Applies to server garbage collection only.
- If [GC processor affinity](#) is enabled, which is the default, the heap count setting affinizes n GC heaps/threads to the first n processors. (Use the [affinize mask](#) or [affinize ranges](#) settings to specify exactly which processors to affinize.)
- If [GC processor affinity](#) is disabled, this setting limits the number of GC heaps.
- For more information, see the [GCHeapCount remarks](#).

	SETTING NAME	VALUES	VERSION INTRODUCED
runtimeconfig.json	<code>System.GC.HeapCount</code>	<i>decimal value</i>	.NET Core 3.0
Environment variable	<code>COMPlus_GCHeapCount</code>	<i>hexadecimal value</i>	.NET Core 3.0
Environment variable	<code>DOTNET_GCHeapCount</code>	<i>hexadecimal value</i>	.NET 6
app.config for .NET Framework	GCHeapCount	<i>decimal value</i>	.NET Framework 4.6.2

Example:

```
{  
    "runtimeOptions": {  
        "configProperties": {  
            "System.GC.HeapCount": 16  
        }  
    }  
}
```

TIP

If you're setting the option in *runtimeconfig.json*, specify a decimal value. If you're setting the option as an environment variable, specify a hexadecimal value. For example, to limit the number of heaps to 16, the values would be 16 for the JSON file and 0x10 or 10 for the environment variable.

Affinize mask

- Specifies the exact processors that garbage collector threads should use.
- If [GC processor affinity](#) is disabled, this setting is ignored.
- Applies to server garbage collection only.
- The value is a bit mask that defines the processors that are available to the process. For example, a decimal value of 1023 (or a hexadecimal value of 0x3FF or 3FF if you're using the environment variable) is 0011 1111 1111 in binary notation. This specifies that the first 10 processors are to be used. To specify the next 10 processors, that is, processors 10-19, specify a decimal value of 1047552 (or a hexadecimal value of 0xFFC00 or FFC00), which is equivalent to a binary value of 1111 1111 1100 0000 0000.

	SETTING NAME	VALUES	VERSION INTRODUCED
runtimeconfig.json	<code>System.GC.HeapAffinizeMask</code> <i>decimal value</i>		.NET Core 3.0

	SETTING NAME	VALUES	VERSION INTRODUCED
Environment variable	COMPlus_GCHeapAffinitizeMask <i>hexadecimal value</i>		.NET Core 3.0
Environment variable	DOTNET_GCHeapAffinitizeMask <i>hexadecimal value</i>		.NET 6
app.config for .NET Framework	GCHeapAffinitizeMask	<i>decimal value</i>	.NET Framework 4.6.2

Example:

```
{
  "runtimeOptions": {
    "configProperties": {
      "System.GC.HeapAffinitizeMask": 1023
    }
  }
}
```

Affinize ranges

- Specifies the list of processors to use for garbage collector threads.
- This setting is similar to [System.GC.HeapAffinitizeMask](#), except it allows you to specify more than 64 processors.
- For Windows operating systems, prefix the processor number or range with the corresponding [CPU group](#), for example, "0:1-10,0:12,1:50-52,1:70".
- If [GC processor affinity](#) is disabled, this setting is ignored.
- Applies to server garbage collection only.
- For more information, see [Making CPU configuration better for GC on machines with > 64 CPUs](#) on Maoni Stephens' blog.

	SETTING NAME	VALUES	VERSION INTRODUCED
runtimeconfig.json	System.GC.HeapAffinitizeRanges	Comma-separated list of processor numbers or ranges of processor numbers. Unix example: "1-10,12,50-52,70" Windows example: "0:1-10,0:12,1:50-52,1:70"	.NET Core 3.0
Environment variable	COMPlus_GCHeapAffinitizeRanges	Comma-separated list of processor numbers or ranges of processor numbers. Unix example: "1-10,12,50-52,70" Windows example: "0:1-10,0:12,1:50-52,1:70"	.NET Core 3.0

	SETTING NAME	VALUES	VERSION INTRODUCED
Environment variable	DOTNET_GCHeapAffinizeRange	Comma-separated list of processor numbers or ranges of processor numbers. Unix example: "1-10,12,50-52,70" Windows example: "0:1-10,0:12,1:50-52,1:70"	.NET 6

Example:

```
{
  "runtimeOptions": {
    "configProperties": {
      "System.GC.HeapAffinizeRanges": "0:1-10,0:12,1:50-52,1:70"
    }
  }
}
```

CPU groups

- Configures whether the garbage collector uses [CPU groups](#) or not.

When a 64-bit Windows computer has multiple CPU groups, that is, there are more than 64 processors, enabling this element extends garbage collection across all CPU groups. The garbage collector uses all cores to create and balance heaps.

- Applies to server garbage collection on 64-bit Windows operating systems only.
- Default: GC does not extend across CPU groups. This is equivalent to setting the value to `0`.
- For more information, see [Making CPU configuration better for GC on machines with > 64 CPUs](#) on Maoni Stephens' blog.

	SETTING NAME	VALUES	VERSION INTRODUCED
runtimeconfig.json	System.GC.CpuGroup	<code>false</code> - disabled <code>true</code> - enabled	.NET 5
Environment variable	COMPlus_GCCpuGroup	<code>0</code> - disabled <code>1</code> - enabled	.NET Core 1.0
Environment variable	DOTNET_GCCpuGroup	<code>0</code> - disabled <code>1</code> - enabled	.NET 6
app.config for .NET Framework	GCCpuGroup	<code>false</code> - disabled <code>true</code> - enabled	

NOTE

To configure the common language runtime (CLR) to also distribute threads from the thread pool across all CPU groups, enable the [Thread_UseAllCpuGroups](#) element option. For .NET Core apps, you can enable this option by setting the value of the `DOTNET_Thread_UseAllCpuGroups` environment variable to `1`.

Affinize

- Specifies whether to *affinitize* garbage collection threads with processors. To affinize a GC thread means that it can only run on its specific CPU. A heap is created for each GC thread.
- Applies to server garbage collection only.
- Default: Affinize garbage collection threads with processors. This is equivalent to setting the value to `false`.

	SETTING NAME	VALUES	VERSION INTRODUCED
<code>runtimeconfig.json</code>	<code>System.GC.NoAffinimize</code>	<code>false</code> - affinize <code>true</code> - don't affinize	.NET Core 3.0
Environment variable	<code>COMPlus_GCNoAffinimize</code>	<code>0</code> - affinize <code>1</code> - don't affinize	.NET Core 3.0
Environment variable	<code>DOTNET_GCNaffinimize</code>	<code>0</code> - affinize <code>1</code> - don't affinize	.NET 6
app.config for .NET Framework	<code>GCNoAffinimize</code>	<code>false</code> - affinize <code>true</code> - don't affinize	.NET Framework 4.6.2

Example:

```
{
  "runtimeOptions": {
    "configProperties": {
      "System.GC.NoAffinimize": true
    }
  }
}
```

Heap limit

- Specifies the maximum commit size, in bytes, for the GC heap and GC bookkeeping.
- This setting only applies to 64-bit computers.
- This setting is ignored if the [Per-object-heap limits](#) are configured.
- The default value, which only applies in certain cases, is the greater of 20 MB or 75% of the memory limit on the container. The default value applies if:
 - The process is running inside a container that has a specified memory limit.
 - `System.GC.HeapHardLimitPercent` is not set.

	SETTING NAME	VALUES	VERSION INTRODUCED
<code>runtimeconfig.json</code>	<code>System.GC.HeapHardLimit</code>	<i>decimal value</i>	.NET Core 3.0
Environment variable	<code>COMPlus_GCHardLimit</code>	<i>hexadecimal value</i>	.NET Core 3.0
Environment variable	<code>DOTNET_GCHardLimit</code>	<i>hexadecimal value</i>	.NET 6

Example:

```
{
  "runtimeOptions": {
    "configProperties": {
      "System.GC.HeapHardLimit": 209715200
    }
  }
}
```

TIP

If you're setting the option in `runtimeconfig.json`, specify a decimal value. If you're setting the option as an environment variable, specify a hexadecimal value. For example, to specify a heap hard limit of 200 mebibytes (MiB), the values would be 209715200 for the JSON file and 0xC800000 or C800000 for the environment variable.

Heap limit percent

- Specifies the allowable GC heap usage as a percentage of the total physical memory.
- If `System.GC.HeapHardLimit` is also set, this setting is ignored.
- This setting only applies to 64-bit computers.
- If the process is running inside a container that has a specified memory limit, the percentage is calculated as a percentage of that memory limit.
- This setting is ignored if the [Per-object-heap limits](#) are configured.
- The default value, which only applies in certain cases, is the greater of 20 MB or 75% of the memory limit on the container. The default value applies if:
 - The process is running inside a container that has a specified memory limit.
 - `System.GC.HeapHardLimit` is not set.

	SETTING NAME	VALUES	VERSION INTRODUCED
<code>runtimeconfig.json</code>	<code>System.GC.HeapHardLimitPercent</code> <i>decimal value</i>		.NET Core 3.0
Environment variable	<code>COMPlus_GCHeapHardLimitPercent</code> <i>hexadecimal value</i>		.NET Core 3.0
Environment variable	<code>DOTNET_GCHardLimitPercent</code> <i>hexadecimal value</i>		.NET 6

Example:

```
{
  "runtimeOptions": {
    "configProperties": {
      "System.GC.HeapHardLimitPercent": 30
    }
  }
}
```

TIP

If you're setting the option in `runtimeconfig.json`, specify a decimal value. If you're setting the option as an environment variable, specify a hexadecimal value. For example, to limit the heap usage to 30%, the values would be 30 for the JSON file and 0x1E or 1E for the environment variable.

Per-object-heap limits

You can specify the GC's allowable heap usage on a per-object-heap basis. The different heaps are the large object heap (LOH), small object heap (SOH), and pinned object heap (POH).

- If you specify a value for any of the `DOTNET_GCHeapHardLimitSOH`, `DOTNET_GCHeapHardLimitLOH`, or `DOTNET_GCHeapHardLimitPOH` settings, you must also specify a value for `DOTNET_GCHeapHardLimitSOH` and `DOTNET_GCHeapHardLimitLOH`. If you don't, the runtime will fail to initialize.
- The default value for `DOTNET_GCHeapHardLimitPOH` is 0. `DOTNET_GCHeapHardLimitSOH` and `DOTNET_GCHeapHardLimitLOH` don't have default values.

	SETTING NAME	VALUES	VERSION INTRODUCED
runtimeconfig.json	<code>System.GC.HeapHardLimitSOH</code>	<i>decimal value</i>	.NET 5
Environment variable	<code>COMPlus_GCHeapHardLimitSOH</code>	<i>hexadecimal value</i>	.NET 5
Environment variable	<code>DOTNET_GCHeapHardLimitSOH</code>	<i>hexadecimal value</i>	.NET 6

	SETTING NAME	VALUES	VERSION INTRODUCED
runtimeconfig.json	<code>System.GC.HeapHardLimitLOH</code>	<i>decimal value</i>	.NET 5
Environment variable	<code>COMPlus_GCHeapHardLimitLOH</code>	<i>hexadecimal value</i>	.NET 5
Environment variable	<code>DOTNET_GCHeapHardLimitLOH</code>	<i>hexadecimal value</i>	.NET 6

	SETTING NAME	VALUES	VERSION INTRODUCED
runtimeconfig.json	<code>System.GC.HeapHardLimitPOH</code>	<i>decimal value</i>	.NET 5
Environment variable	<code>COMPlus_GCHeapHardLimitPOH</code>	<i>hexadecimal value</i>	.NET 5
Environment variable	<code>DOTNET_GCHeapHardLimitPOH</code>	<i>hexadecimal value</i>	.NET 6

TIP

If you're setting the option in `runtimeconfig.json`, specify a decimal value. If you're setting the option as an environment variable, specify a hexadecimal value. For example, to specify a heap hard limit of 200 mebibytes (MiB), the values would be 209715200 for the JSON file and 0xC800000 or C800000 for the environment variable.

Per-object-heap limit percents

You can specify the GC's allowable heap usage on a per-object-heap basis. The different heaps are the large object heap (LOH), small object heap (SOH), and pinned object heap (POH).

- If you specify a value for any of the `DOTNET_GCHeapHardLimitSOHPercent`, `DOTNET_GCHeapHardLimitLOHPercent`, or `DOTNET_GCHeapHardLimitPOHPercent` settings, you must also specify a value for `DOTNET_GCHeapHardLimitSOHPercent` and `DOTNET_GCHeapHardLimitLOHPercent`. If you don't, the runtime will fail to initialize.
- These settings are ignored if `DOTNET_GCHeapHardLimitSOH`, `DOTNET_GCHeapHardLimitLOH`, and `DOTNET_GCHeapHardLimitPOH` are specified.
- A value of 1 means that GC uses 1% of total physical memory for that object heap.

- Each value must be greater than zero and less than 100. Additionally, the sum of the three percentage values must be less than 100. Otherwise, the runtime will fail to initialize.

	SETTING NAME	VALUES	VERSION INTRODUCED
runtimeconfig.json	System.GC.HeapHardLimitSOHPer <code>decimal value</code>		.NET 5
Environment variable	COMplus_GCHardLimitSOHPer <code>hexadecimal value</code>		.NET 5
Environment variable	DOTNET_GCHardLimitSOHPer <code>hexadecimal value</code>		.NET 6
	SETTING NAME	VALUES	VERSION INTRODUCED
runtimeconfig.json	System.GC.HeapHardLimitLOHPer <code>decimal value</code>		.NET 5
Environment variable	COMplus_GCHardLimitLOHPer <code>hexadecimal value</code>		.NET 5
Environment variable	DOTNET_GCHardLimitLOHPer <code>hexadecimal value</code>		.NET 6
	SETTING NAME	VALUES	VERSION INTRODUCED
runtimeconfig.json	System.GC.HeapHardLimitPOHPer <code>decimal value</code>		.NET 5
Environment variable	COMplus_GCHardLimitPOHPer <code>hexadecimal value</code>		.NET 5
Environment variable	DOTNET_GCHardLimitPOHPer <code>hexadecimal value</code>		.NET 6

TIP

If you're setting the option in `runtimeconfig.json`, specify a decimal value. If you're setting the option as an environment variable, specify a hexadecimal value. For example, to limit the heap usage to 30%, the values would be 30 for the JSON file and 0x1E or 1E for the environment variable.

High memory percent

Memory load is indicated by the percentage of physical memory in use. By default, when the physical memory load reaches 90%, garbage collection becomes more aggressive about doing full, compacting garbage collections to avoid paging. When memory load is below 90%, GC favors background collections for full garbage collections, which have shorter pauses but don't reduce the total heap size by much. On machines with a significant amount of memory (80GB or more), the default load threshold is between 90% and 97%.

The high memory load threshold can be adjusted by the `DOTNET_GCHighMemPercent` environment variable or `System.GC.HighMemoryPercent` JSON configuration setting. Consider adjusting the threshold if you want to control heap size. For example, for the dominant process on a machine with 64GB of memory, it's reasonable for GC to start reacting when there's 10% of memory available. But for smaller processes, for example, a process that only consumes 1GB of memory, GC can comfortably run with less than 10% of memory available. For these smaller processes, consider setting the threshold higher. On the other hand, if you want larger processes to have smaller heap sizes (even when there's plenty of physical memory available), lowering this threshold is an effective way for GC to react sooner to compact the heap down.

NOTE

For processes running in a container, GC considers the physical memory based on the container limit.

	SETTING NAME	VALUES	VERSION INTRODUCED
runtimeconfig.json	System.GC.HighMemoryPercent	decimal value	.NET 5
Environment variable	COMPlus_GCHighMemPercent	hexadecimal value	.NET Core 3.0 .NET Framework 4.7.2
Environment variable	DOTNET_GCHighMemPercent	hexadecimal value	.NET 6

TIP

If you're setting the option in `runtimeconfig.json`, specify a decimal value. If you're setting the option as an environment variable, specify a hexadecimal value. For example, to set the high memory threshold to 75%, the values would be 75 for the JSON file and 0x4B or 4B for the environment variable.

Retain VM

- Configures whether segments that should be deleted are put on a standby list for future use or are released back to the operating system (OS).
- Default: Release segments back to the operating system. This is equivalent to setting the value to `false`.

	SETTING NAME	VALUES	VERSION INTRODUCED
runtimeconfig.json	System.GC.RetainVM	<code>false</code> - release to OS <code>true</code> - put on standby	.NET Core 1.0
MSBuild property	RetainVMGarbageCollection	<code>false</code> - release to OS <code>true</code> - put on standby	.NET Core 1.0
Environment variable	COMPlus_GCRetainVM	<code>0</code> - release to OS <code>1</code> - put on standby	.NET Core 1.0
Environment variable	DOTNET_GCRetainVM	<code>0</code> - release to OS <code>1</code> - put on standby	.NET 6

Examples

`runtimeconfig.json` file:

```
{
  "runtimeOptions": {
    "configProperties": {
      "System.GC.RetainVM": true
    }
  }
}
```

Project file:

```

<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <RetainVMGarbageCollection>true</RetainVMGarbageCollection>
  </PropertyGroup>

</Project>

```

Large pages

- Specifies whether large pages should be used when a heap hard limit is set.
- Default: Don't use large pages when a heap hard limit is set. This is equivalent to setting the value to `0`.
- This is an experimental setting.

	SETTING NAME	VALUES	VERSION INTRODUCED
<code>runtimeconfig.json</code>	N/A	N/A	N/A
Environment variable	<code>COMPlus_GCLargePages</code>	<input type="checkbox"/> <code>0</code> - disabled <input checked="" type="checkbox"/> <code>1</code> - enabled	.NET Core 3.0
Environment variable	<code>DOTNET_GCLargePages</code>	<input type="checkbox"/> <code>0</code> - disabled <input checked="" type="checkbox"/> <code>1</code> - enabled	.NET 6

Allow large objects

- Configures garbage collector support on 64-bit platforms for arrays that are greater than 2 gigabytes (GB) in total size.
- Default: GC supports arrays greater than 2-GB. This is equivalent to setting the value to `1`.
- This option may become obsolete in a future version of .NET.

	SETTING NAME	VALUES	VERSION INTRODUCED
<code>runtimeconfig.json</code>	N/A	N/A	N/A
Environment variable	<code>COMPlus_gcAllowVeryLargeObjects</code>	<input checked="" type="checkbox"/> <code>1</code> - enabled <input type="checkbox"/> <code>0</code> - disabled	.NET Core 1.0
Environment variable	<code>DOTNET_gcAllowVeryLargeObjects</code>	<input checked="" type="checkbox"/> <code>1</code> - enabled <input type="checkbox"/> <code>0</code> - disabled	.NET 6
app.config for .NET Framework	<code>gcAllowVeryLargeObjects</code>	<input checked="" type="checkbox"/> <code>1</code> - enabled <input type="checkbox"/> <code>0</code> - disabled	.NET Framework 4.5

Large object heap threshold

- Specifies the threshold size, in bytes, that causes objects to go on the large object heap (LOH).
- The default threshold is 85,000 bytes.
- The value you specify must be larger than the default threshold.

	SETTING NAME	VALUES	VERSION INTRODUCED
runtimeconfig.json	System.GC.LOHTreshold	<i>decimal value</i>	.NET Core 1.0
Environment variable	COMPlus_GCLOHTreshold	<i>hexadecimal value</i>	.NET Core 1.0
Environment variable	DOTNET_GCLOHTreshold	<i>hexadecimal value</i>	.NET 6
app.config for .NET Framework	GCLOHTreshold	<i>decimal value</i>	.NET Framework 4.8

Example:

```
{
  "runtimeOptions": {
    "configProperties": {
      "System.GC.LOHTreshold": 120000
    }
  }
}
```

TIP

If you're setting the option in *runtimeconfig.json*, specify a decimal value. If you're setting the option as an environment variable, specify a hexadecimal value. For example, to set a threshold size of 120,000 bytes, the values would be 120000 for the JSON file and 0x1D4C0 or 1D4C0 for the environment variable.

Standalone GC

- Specifies a path to the library containing the garbage collector that the runtime intends to load.
- For more information, see [Standalone GC loader design](#).

	SETTING NAME	VALUES	VERSION INTRODUCED
runtimeconfig.json	N/A	N/A	N/A
Environment variable	COMPlus_GCName	<i>string_path</i>	.NET Core 2.0
Environment variable	DOTNET_GCName	<i>string_path</i>	.NET 6

Conserve memory

- Configures the garbage collector to conserve memory at the expense of more frequent garbage collections and possibly longer pause times.
- Default value is 0 - this implies no change.
- Besides the default value 0, values between 1 and 9 (inclusive) are valid. The higher the value, the more the garbage collector tries to conserve memory and thus to keep the heap small.
- If the value is non-zero, the large object heap will be compacted automatically if it has too much fragmentation.

	SETTING NAME	VALUES	VERSION INTRODUCED
runtimeconfig.json	N/A	N/A	N/A
Environment variable	<code>COMPlus_GCConserveMemory</code>	<code>0</code> - <code>9</code>	.NET Framework 4.8
Environment variable	<code>DOTNET_GCConserveMemory</code>	<code>0</code> - <code>9</code>	.NET 6
app.config for .NET Framework	<code>GCConserveMemory</code>	<code>0</code> - <code>9</code>	.NET Framework 4.8

Example *app.config* file:

```
<configuration>
  <runtime>
    <GCConserveMemory enabled="5"/>
  </runtime>
</configuration>
```

TIP

Experiment with different numbers to see which value works best for you. Start with a value between 5 and 7.

Runtime configuration options for globalization

9/20/2022 • 2 minutes to read • [Edit Online](#)

Invariant mode

- Determines whether a .NET Core app runs in globalization-invariant mode without access to culture-specific data and behavior.
- If you omit this setting, the app runs with access to cultural data. This is equivalent to setting the value to `false`.
- For more information, see [.NET Core globalization invariant mode](#).

	SETTING NAME	VALUES
runtimesconfig.json	<code>System.Globalization.Invariant</code>	<code>false</code> - access to cultural data <code>true</code> - run in invariant mode
MSBuild property	<code>InvariantGlobalization</code>	<code>false</code> - access to cultural data <code>true</code> - run in invariant mode
Environment variable	<code>DOTNET_SYSTEM_GLOBALIZATION_INVARIANT</code>	<code>0</code> - access to cultural data <code>1</code> - run in invariant mode

Examples

runtimesconfig.json file:

```
{
  "runtimeOptions": {
    "configProperties": {
      "System.Globalization.Invariant": true
    }
  }
}
```

Project file:

```
<Project Sdk="Microsoft.NET.Sdk">

<PropertyGroup>
  <InvariantGlobalization>true</InvariantGlobalization>
</PropertyGroup>

</Project>
```

Era year ranges

- Determines whether range checks for calendars that support multiple eras are relaxed or whether dates that overflow an era's date range throw an [ArgumentOutOfRangeException](#).
- If you omit this setting, range checks are relaxed. This is equivalent to setting the value to `false`.
- For more information, see [Calendars, eras, and date ranges: Relaxed range checks](#).

	SETTING NAME	VALUES
runtimeconfig.json	Switch.System.Globalization.EnforceJapaneseYearFormat	false - relaxed range checks true - overflows cause an exception
Environment variable	N/A	N/A

Japanese date parsing

- Determines whether a string that contains either "1" or "Gannen" as the year parses successfully or whether only "1" is supported.
- If you omit this setting, strings that contain either "1" or "Gannen" as the year parse successfully. This is equivalent to setting the value to `false`.
- For more information, see [Represent dates in calendars with multiple eras](#).

	SETTING NAME	VALUES
runtimeconfig.json	Switch.System.Globalization.EnforceJapaneseYearFormat	false - "Gannen" parsing is supported true - only "1" is supported
Environment variable	N/A	N/A

Japanese year format

- Determines whether the first year of a Japanese calendar era is formatted as "Gannen" or as a number.
- If you omit this setting, the first year is formatted as "Gannen". This is equivalent to setting the value to `false`.
- For more information, see [Represent dates in calendars with multiple eras](#).

	SETTING NAME	VALUES
runtimeconfig.json	Switch.System.Globalization.FormatJapaneseYearFormat	false - format as "Gannen" true - format as number
Environment variable	N/A	N/A

NLS

- Determines whether .NET uses National Language Support (NLS) or International Components for Unicode (ICU) globalization APIs for Windows apps. .NET 5 and later versions use ICU globalization APIs by default on Windows 10 May 2019 Update and later versions.
- If you omit this setting, .NET uses ICU globalization APIs by default. This is equivalent to setting the value to `false`.
- For more information, see [Globalization APIs use ICU libraries on Windows](#).

	SETTING NAME	VALUES	INTRODUCED

	SETTING NAME	VALUES	INTRODUCED
runtimesconfig.json	<code>System.Globalization.UseNls</code>	false - Use ICU globalization APIs true - Use NLS globalization APIs	.NET 5
Environment variable	<code>DOTNET_SYSTEM_GLOBALIZATION</code>	false - Use ICU globalization APIs true - Use NLS globalization APIs	.NET 5

Predefined cultures

- Configures whether apps can create cultures other than the invariant culture when [globalization-invariant mode](#) is enabled.
- If you omit this setting, .NET restricts the creation of cultures in globalization-invariant mode. This is equivalent to setting the value to `true`.
- For more information, see [Culture creation and case mapping in globalization-invariant mode](#).

	SETTING NAME	VALUES	INTRODUCED
runtimesconfig.json	<code>System.Globalization.Predefi</code>	<code>true</code> - In globalization-invariant mode, don't allow creation of any culture except the invariant culture. <code>false</code> - Allow creation of any culture.	.NET 6
MSBuild property	<code>PredefinedCulturesOnly</code>	<code>true</code> - In globalization-invariant mode, don't allow creation of any culture except the invariant culture. <code>false</code> - Allow creation of any culture.	.NET 6
Environment variable	<code>DOTNET_SYSTEM_GLOBALIZATION</code>	<code>true</code> - In globalization ONLY invariant mode, don't allow creation of any culture except the invariant culture. <code>false</code> - Allow creation of any culture.	.NET 6

Runtime configuration options for networking

9/20/2022 • 2 minutes to read • [Edit Online](#)

HTTP/2 protocol

- Configures whether support for the HTTP/2 protocol is enabled.
- Introduced in .NET Core 3.0.
- .NET Core 3.0 only: If you omit this setting, support for the HTTP/2 protocol is disabled. This is equivalent to setting the value to `false`.
- .NET Core 3.1 and .NET 5+: If you omit this setting, support for the HTTP/2 protocol is enabled. This is equivalent to setting the value to `true`.

	SETTING NAME	VALUES
runtimesconfig.json	<code>System.Net.Http.SocketsHttpHandler.Http2Support</code>	<code>false</code> - disabled <code>true</code> - enabled
Environment variable	<code>DOTNET_SYSTEM_NET_HTTP_SOCKETSHTTPHANDLER2_SUPPORT</code>	<code>0</code> - disabled <code>1</code> - enabled

SPN creation in HttpClient (.NET 6 and later)

- Impacts generation of [service principal names](#) (SPN) for Kerberos and NTLM authentication when `Host` header is missing and target is not running on default port.
- .NET Core 2.x and 3.x do not include port in SPN.
- .NET Core 5.x does include port in SPN
- .NET 6 and later versions don't include the port, but the behavior is configurable.

	SETTING NAME	VALUES
runtimesconfig.json	<code>System.Net.Http.UsePortInSpn</code>	<code>true</code> - includes port number in SPN, for example, <code>HTTP/host:port</code> <code>false</code> - does not include port in SPN, for example, <code>HTTP/host</code>
Environment variable	<code>DOTNET_SYSTEM_NET_HTTP_USEPORTINSPN</code>	<code>1</code> - includes port number in SPN, for example, <code>HTTP/host:port</code> <code>0</code> - does not include port in SPN, for example, <code>HTTP/host</code>

UseSocketsHttpHandler (.NET Core 2.1-3.1 only)

- Configures whether `System.Net.Http.HttpClientHandler` uses `System.Net.Http.SocketsHttpHandler` or older HTTP protocol stacks (`WinHttpHandler` on Windows and `CurlHandler`, an internal class implemented on top of `libcurl`, on Linux).

NOTE

You may be using high-level networking APIs instead of directly instantiating the [HttpClientHandler](#) class. This setting also affects which HTTP protocol stack is used by high-level networking APIs, including [HttpClient](#) and [HttpClientFactory](#).

- If you omit this setting, [HttpClientHandler](#) uses [SocketsHttpHandler](#). This is equivalent to setting the value to `true`.

	SETTING NAME	VALUES
runtimesconfig.json	<code>System.Net.Http.UseSocketsHttpHandler</code>	<code>true</code> - enables the use of SocketsHttpHandler <code>false</code> - enables the use of WinHttpHandler on Windows or curl on Linux
Environment variable	<code>DOTNET_SYSTEM_NET_HTTP_USESOCKETSHTTP</code>	<code>1</code> - enables the use of SocketsHttpHandler <code>0</code> - enables the use of WinHttpHandler on Windows or curl on Linux

NOTE

Starting in .NET 5, the `System.Net.Http.UseSocketsHttpHandler` setting is no longer available.

Latin1 headers (.NET Core 3.1 only)

- This switch allows relaxing the HTTP header validation, enabling [SocketsHttpHandler](#) to send ISO-8859-1 (Latin-1) encoded characters in headers.
- If you omit this setting, an attempt to send a non-ASCII character will result in [HttpRequestException](#). This is equivalent to setting the value to `false`.

	SETTING NAME	VALUES
runtimesconfig.json	<code>System.Net.Http.SocketsHttpHandler.AllowNonAsciiHeader</code>	<code>false</code> - disabled <code>true</code> - enabled
Environment variable	<code>DOTNET_SYSTEM_NET_HTTP_SOCKETSHTTPHANDLER</code>	<code>0</code> - disabled <code>1</code> - enabled

NOTE

This option is only available in .NET Core 3.1 since version 3.1.9, and not in previous or later versions. In .NET 5 we recommend using [RequestHeaderEncodingSelector](#).

Runtime configuration options for threading

9/20/2022 • 3 minutes to read • [Edit Online](#)

This article details the settings you can use to configure threading in .NET.

NOTE

.NET 6 standardizes on the prefix `DOTNET_` instead of `COMPlus_` for environment variables that configure .NET run-time behavior. However, the `COMPlus_` prefix will continue to work. If you're using a previous version of the .NET runtime, you should still use the `COMPlus_` prefix for environment variables.

CPU groups

- Configures whether threads are automatically distributed across CPU groups.
- If you omit this setting, threads are not distributed across CPU groups. This is equivalent to setting the value to `0`.

	SETTING NAME	VALUES
runtimesconfig.json	N/A	N/A
Environment variable	<code>COMPlus_Thread_UseAllCpuGroups</code> or <code>DOTNET_Thread_UseAllCpuGroups</code>	<code>0</code> - disabled <code>1</code> - enabled

Minimum threads

- Specifies the minimum number of threads for the worker thread pool.
- Corresponds to the [ThreadPool.SetMinThreads](#) method.

	SETTING NAME	VALUES
runtimesconfig.json	<code>System.Threading.ThreadPool.MinThreads</code>	An integer that represents the minimum number of threads
MSBuild property	<code>ThreadPoolMinThreads</code>	An integer that represents the minimum number of threads
Environment variable	N/A	N/A

Examples

`runtimesconfig.json` file:

```
{  
    "runtimeOptions": {  
        "configProperties": {  
            "System.Threading.ThreadPool.MinThreads": 4  
        }  
    }  
}
```

Project file:

```
<Project Sdk="Microsoft.NET.Sdk">

<PropertyGroup>
    <ThreadPoolMinThreads>4</ThreadPoolMinThreads>
</PropertyGroup>

</Project>
```

Maximum threads

- Specifies the maximum number of threads for the worker thread pool.
- Corresponds to the [ThreadPool.SetMaxThreads](#) method.

	SETTING NAME	VALUES
runtimeconfig.json	System.Threading.ThreadPool.MaxThreads	An integer that represents the maximum number of threads
MSBuild property	ThreadPoolMaxThreads	An integer that represents the maximum number of threads
Environment variable	N/A	N/A

Examples

runtimeconfig.json file:

```
{
    "runtimeOptions": {
        "configProperties": {
            "System.Threading.ThreadPool.MaxThreads": 20
        }
    }
}
```

Project file:

```
<Project Sdk="Microsoft.NET.Sdk">

<PropertyGroup>
    <ThreadPoolMaxThreads>20</ThreadPoolMaxThreads>
</PropertyGroup>

</Project>
```

Thread injection in response to blocking work items

In some cases, the thread pool detects work items that block its threads. To compensate, it injects more threads. In .NET 6+, you can use the following [runtime configuration](#) settings to configure thread injection in response to blocking work items. Currently, these settings take effect only for work items that wait for another task to complete, such as in typical [sync-over-async](#) cases.

RUNTIMECONFIG.JSON SETTING NAME	DESCRIPTION	VERSION INTRODUCED
System.Threading.ThreadPool.Blocking. After the thread count based on MinThreads	After the thread count based on <code>MinThreads</code> is reached, this value (after it is multiplied by the processor count) specifies how many additional threads may be created without a delay.	.NET 6
System.Threading.ThreadPool.Blocking. After the thread count based on Factor	After the thread count based on <code>Factor</code> is reached, this value (after it is multiplied by the processor count) specifies after how many threads an additional <code>DelayStepMs</code> would be added to the delay before each new thread is created.	.NET 6
System.Threading.ThreadPool.Blocking. After the thread count based on ThreadsToAddWithoutDelay	After the thread count based on <code>ThreadsToAddWithoutDelay</code> is reached, this value specifies how much additional delay to add per <code>ThreadsPerDelayStep</code> threads, which would be applied before each new thread is created.	.NET 6
System.Threading.ThreadPool.Blocking. After the thread count based on MaxDelayMs	After the thread count based on <code>MaxDelayMs</code> is reached, this value specifies the max delay to use before each new thread is created.	.NET 6
System.Threading.ThreadPool.Blocking. By default, the rate of thread injection	By default, the rate of thread injection in response to blocking is limited by heuristics that determine whether there is sufficient physical memory available. In some situations, it may be preferable to inject threads more quickly even in low-memory situations. You can disable the memory usage heuristics by turning off this switch.	.NET 7

How the configuration settings take effect

- After the thread count based on `MinThreads` is reached, up to `ThreadsToAddWithoutDelay` additional threads may be created without a delay.
- After that, before each additional thread is created, a delay is induced, starting with `DelayStepMs`.
- For every `ThreadsPerDelayStep` threads that are added with a delay, an additional `DelayStepMs` is added to the delay.
- The delay may not exceed `MaxDelayMs`.
- Delays are only induced before creating threads. If threads are already available, they would be released without delay to compensate for blocking work items.
- Physical memory usage and limits are also used and, beyond a threshold, the system switches to slower thread injection.

Examples

`runtimedataconfig.json` file:

```
{
  "runtimeOptions": {
    "configProperties": {
      "System.Threading.ThreadPool.Blocking.ThreadsToAddWithoutDelay_ProcCountFactor": 5
    }
  }
}
```

AutoreleasePool for managed threads

This option configures whether each managed thread receives an implicit [NSAutoreleasePool](#) when running on a supported macOS platform.

	SETTING NAME	VALUES	VERSION INTRODUCED
runtimeconfig.json	System.Threading.Thread.EnableAutoreleasePool	true or false	.NET 6
MSBuild property	AutoreleasePoolSupport	true or false	.NET 6
Environment variable	N/A	N/A	N/A

Examples

runtimeconfig.json file:

```
{
  "runtimeOptions": {
    "configProperties": {
      "System.Threading.Thread.EnableAutoreleasePool": true
    }
  }
}
```

Project file:

```
<Project Sdk="Microsoft.NET.Sdk">

<PropertyGroup>
  <AutoreleasePoolSupport>true</AutoreleasePoolSupport>
</PropertyGroup>

</Project>
```

Troubleshoot app launch failures

9/20/2022 • 6 minutes to read • [Edit Online](#)

This article describes some common reasons and possible solutions for application launch failures. It relates to [framework-dependent applications](#), which rely on a .NET installation on your machine.

If you already know which .NET version you need, you can download it from [.NET downloads](#).

.NET installation not found

If a .NET installation is not found, the application fails to launch with a message similar to:

```
You must install .NET to run this application.

App: C:\repos\myapp\myapp.exe
Architecture: x64
Host version: 7.0.0
.NET location: Not found
```

```
You must install .NET to run this application.

App: /home/user/repos/myapp/myapp
Architecture: x64
Host version: 7.0.0
.NET location: Not found
```

The error message includes a link to download .NET. You can follow this link to get to the appropriate download page. You can also pick the .NET version (specified by [Host version](#)) from [.NET downloads](#).

On the [download page](#) for the required .NET version, find the **.NET Runtime** download that matches the architecture listed in the error message. You can then install it by downloading and running an **Installer**.

.NET is available through various Linux package managers. See [Install .NET on Linux](#) for details. Note that preview versions of .NET are typically not available through package managers.

You need to install the .NET Runtime package for the appropriate version, like `dotnet-runtime6`.

Alternatively, on the [download page](#) for the required .NET version, you can download **Binaries** for the specified architecture.

Required framework not found

If a required framework or compatible version is not found, the application fails to launch with a message similar to:

```
You must install or update .NET to run this application.
```

```
App: C:\repos\myapp\myapp.exe
Architecture: x64
Framework: 'Microsoft.NETCore.App', version '5.0.15' (x64)
.NET location: C:\Program Files\dotnet\
```

```
The following frameworks were found:
  6.0.2 at [c:\Program Files\dotnet\shared\Microsoft.NETCore.App]
```

```
You must install or update .NET to run this application.
```

```
App: /home/user/repos/myapp/myapp
Architecture: x64
Framework: 'Microsoft.NETCore.App', version '5.0.15' (x64)
.NET location: /usr/share/dotnet/
```

```
The following frameworks were found:
  6.0.2 at [/usr/share/dotnet/shared/Microsoft.NETCore.App]
```

```
You must install or update .NET to run this application.
```

```
App: /home/user/repos/myapp/myapp
Architecture: x64
Framework: 'Microsoft.NETCore.App', version '5.0.15' (x64)
.NET location: /usr/local/share/dotnet/
```

```
The following frameworks were found:
  6.0.2 at [/usr/local/share/dotnet/shared/Microsoft.NETCore.App]
```

The error indicates the name, version, and architecture of the missing framework and the location at which it is expected to be installed. To run the application, you can [install a compatible runtime](#) at the specified ".NET location". If the application is targeting a lower version than one you have installed and you would like to run it on a higher version, you can also [configure roll-forward behavior](#) for the application.

Install a compatible runtime

The error message includes a link to download the missing framework. You can follow this link to get to the appropriate download page.

Alternately, you can download a runtime from the [.NET downloads](#) page. There are multiple .NET runtime downloads.

The following table shows the frameworks that each runtime contains.

RUNTIME DOWNLOAD	INCLUDED FRAMEWORKS
ASP.NET Core Runtime	Microsoft.NETCore.App Microsoft.AspNetCore.App
.NET Desktop Runtime	Microsoft.NETCore.App Microsoft.WindowsDesktop.App
.NET Runtime	Microsoft.NETCore.App

RUNTIME DOWNLOAD	INCLUDED FRAMEWORKS
ASP.NET Core Runtime	Microsoft.NETCore.App Microsoft.AspNetCore.App
.NET Runtime	Microsoft.NETCore.App

Select a runtime download containing the missing framework, and install it.

On the [download page](#) for the required .NET version, find the runtime download that matches the architecture listed in the error message. You likely want to download an **Installer**.

.NET is available through various Linux package managers. See [Install .NET on Linux](#) for details. Note that preview versions of .NET are typically not available through package managers.

You need to install the .NET runtime package for the appropriate version, like `dotnet-runtime6` or `dotnet-aspnet6`.

Alternatively, on the [download page](#) for the required .NET version, you can download **Binaries** for the specified architecture.

In most cases, when the application that failed to launch is using such an installation, the ".NET location" in the error message points to:

```
%ProgramFiles%\dotnet
```

```
/usr/share/dotnet/
```

```
/usr/local/share/dotnet/
```

Other options

There are other installation and workaround options to consider.

Run the `dotnet-install` script

Download the [dotnet-install script](#) for your operating system. Run the script with options based on the information in the error message. The [dotnet-install script reference page](#) shows all available options.

Launch [PowerShell](#) and run:

```
dotnet-install.ps1 -Architecture <architecture> -InstallDir <directory> -Runtime <runtime> -Version <version>
```

For example, the error message in the previous section would correspond to:

```
dotnet-install.ps1 -Architecture x64 -InstallDir "C:\Program Files\dotnet\" -Runtime dotnet -Version 5.0.15
```

If you encounter an error stating that running scripts is disabled, you may need to set the [execution policy](#) to allow the script to run:

```
Set-ExecutionPolicy Bypass -Scope Process
```

For more details on installation using the script, see [Install with PowerShell automation](#).

```
./dotnet-install.sh --architecture <architecture> --install-dir <directory> --runtime <runtime> --version <version>
```

For example, the error message in the previous section would correspond to:

```
./dotnet-install.sh --architecture x64 --install-dir /usr/share/dotnet/ --runtime dotnet --version 5.0.15
```

For more details on installation using the script, see [Scripted install](#).

```
./dotnet-install.sh --architecture <architecture> --install-dir <directory> --runtime <runtime> --version <version>
```

For example, the error message in the previous section would correspond to:

```
./dotnet-install.sh --architecture x64 --install-dir /usr/local/share/dotnet/ --runtime dotnet --version 5.0.15
```

For more details on installation using the script, see [Install with bash automation](#).

Download binaries

You can download a binary archive of .NET from the [download page](#). From the **Binaries** column of the runtime download, download the binary release matching the required architecture. Extract the downloaded archive to the ".NET location" specified in the error message.

For more details on manual installation, see [Install .NET on Windows](#)

For more details on manual installation, see [Install .NET on Linux](#)

For more details on manual installation, see [Install .NET on macOS](#)

Configure roll-forward behavior

If you already have a higher version of the required framework installed, you can make the application run on that higher version by configuring its roll-forward behavior.

When running the application, you can specify the [--roll-forward command line option](#) or set the [DOTNET_ROLL_FORWARD environment variable](#). By default, an application requires a framework that matches the same major version that the application targets, but can use a higher minor or patch version. However, application developers may have specified a different behavior. For more details, see [Framework-dependent apps roll-forward](#).

NOTE

Since using this option lets the application run on a different framework version than the one for which it was designed, it may result in unintended behavior due to changes between versions of a framework.

Breaking changes

Multi-level lookup disabled for .NET 7 and later

On Windows, before .NET 7, the application could search for frameworks in multiple [install locations](#).

1. Subdirectories relative to:

- `dotnet` executable when running the application through `dotnet`

- `DOTNET_ROOT` environment variable (if set) when running the application through its executable (`apphost`)
2. Globally registered install location (if set) in
`HKLM\SOFTWARE\dotnet\Setup\InstalledVersions\<arch>\InstallLocation`.
3. Default install location of `%ProgramFiles%\dotnet` (or `%ProgramFiles(x86)%\dotnet` for 32-bit processes on 64-bit Windows).

This multi-level lookup behavior was enabled by default but could be disabled by setting the environment variable `DOTNET_MULTILEVEL_LOOKUP=0`.

For applications targeting .NET 7 and later, multi-level lookup is completely disabled and only one location—the first location where a .NET installation is found—is searched. When running an application through `dotnet`, frameworks are only searched for in subdirectories relative to `dotnet`. When running an application through its executable (`apphost`), frameworks are only searched for in the first of the above locations where .NET is found.

For more details, see [Disable multi-level lookup by default](#).

See also

- [Install .NET](#)
- [.NET install locations](#)
- [Check installed .NET versions](#)
- [Framework-dependent applications](#)

.NET application publishing overview

9/20/2022 • 7 minutes to read • [Edit Online](#)

Applications you create with .NET can be published in two different modes, and the mode affects how a user runs your app.

Publishing your app as *self-contained* produces an application that includes the .NET runtime and libraries, and your application and its dependencies. Users of the application can run it on a machine that doesn't have the .NET runtime installed.

Publishing your app as *framework-dependent* produces an application that includes only your application itself and its dependencies. Users of the application have to separately install the .NET runtime.

Both publishing modes produce a platform-specific executable by default. Framework-dependent applications can be created without an executable, and these applications are cross-platform.

When an executable is produced, you can specify the target platform with a runtime identifier (RID). For more information about RIDs, see [.NET RID Catalog](#).

The following table outlines the commands used to publish an app as framework-dependent or self-contained, per SDK version:

TYPE	SDK 2.1	SDK 3.1	SDK 5.0	SDK 6.0	COMMAND
framework-dependent executable for the current platform.		✓	✓	✓	<code>dotnet publish</code>
framework-dependent executable for a specific platform.		✓	✓	✓	<code>dotnet publish -r <RID> --self-contained false</code>
framework-dependent cross-platform binary.	✓	✓	✓	✓	<code>dotnet publish</code>
self-contained executable.	✓	✓	✓	✓	<code>dotnet publish -r <RID></code>

For more information, see [.NET dotnet publish command](#).

Produce an executable

Executables aren't cross-platform. They're specific to an operating system and CPU architecture. When publishing your app and creating an executable, you can publish the app as *self-contained* or *framework-dependent*. Publishing an app as self-contained includes the .NET runtime with the app, and users of the app don't have to worry about installing .NET before running the app. Apps published as framework-dependent don't include the .NET runtime and libraries; only the app and 3rd-party dependencies are included.

The following commands produce an executable:

TYPE	SDK 2.1	SDK 3.1	SDK 5.0	SDK 6.0	COMMAND
framework-dependent executable for the current platform.		✓	✓	✓	<code>dotnet publish</code>
framework-dependent executable for a specific platform.		✓	✓	✓	<code>dotnet publish -r <RID> --self-contained false</code>
self-contained executable.	✓	✓	✓	✓	<code>dotnet publish -r <RID></code>

Produce a cross-platform binary

Cross-platform binaries are created when you publish your app as [framework-dependent](#), in the form of a *dll* file. The *dll* file is named after your project. For example, if you have an app named `word_reader`, a file named `word_reader.dll` is created. Apps published in this way are run with the `dotnet <filename.dll>` command and can be run on any platform.

Cross-platform binaries can be run on any operating system as long as the targeted .NET runtime is already installed. If the targeted .NET runtime isn't installed, the app may run using a newer runtime if the app is configured to roll-forward. For more information, see [framework-dependent apps roll forward](#).

The following command produces a cross-platform binary:

TYPE	SDK 2.1	SDK 3.X	SDK 5.0	SDK 6.0	COMMAND
framework-dependent cross-platform binary.	✓	✓	✓	✓	<code>dotnet publish</code>

Publish framework-dependent

Apps published as framework-dependent are cross-platform and don't include the .NET runtime. The user of your app is required to install the .NET runtime.

Publishing an app as framework-dependent produces a [cross-platform binary](#) as a *dll* file, and a [platform-specific executable](#) that targets your current platform. The *dll* is cross-platform while the executable isn't. For example, if you publish an app named `word_reader` and target Windows, a `word_reader.exe` executable is created along with `word_reader.dll`. When targeting Linux or macOS, a `word_reader` executable is created along with `word_reader.dll`. For more information about RIDs, see [.NET RID Catalog](#).

IMPORTANT

.NET SDK 2.1 doesn't produce platform-specific executables when you publish an app framework-dependent.

The cross-platform binary of your app can be run with the `dotnet <filename.dll>` command, and can be run on any platform. If the app uses a NuGet package that has platform-specific implementations, all platforms'

dependencies are copied to the publish folder along with the app.

You can create an executable for a specific platform by passing the `-r <RID> --self-contained false` parameters to the `dotnet publish` command. When the `-r` parameter is omitted, an executable is created for your current platform. Any NuGet packages that have platform-specific dependencies for the targeted platform are copied to the publish folder. If you don't need a platform-specific executable, you can specify `<UseAppHost>False</UseAppHost>` in the project file. For more information, see [MSBuild reference for .NET SDK projects](#).

Advantages

- **Small deployment**

Only your app and its dependencies are distributed. The .NET runtime and libraries are installed by the user and all apps share the runtime.

- **Cross-platform**

Your app and any .NET-based library runs on other operating systems. You don't need to define a target platform for your app. For information about the .NET file format, see [.NET Assembly File Format](#).

- **Uses the latest patched runtime**

The app uses the latest runtime (within the targeted major-minor family of .NET) installed on the target system. This means your app automatically uses the latest patched version of the .NET runtime. This default behavior can be overridden. For more information, see [framework-dependent apps roll forward](#).

Disadvantages

- **Requires pre-installing the runtime**

Your app can run only if the version of .NET your app targets is already installed on the host system. You can configure roll-forward behavior for the app to either require a specific version of .NET or allow a newer version of .NET. For more information, see [framework-dependent apps roll forward](#).

- **.NET may change**

It's possible for the .NET runtime and libraries to be updated on the machine where the app is run. In rare cases, this may change the behavior of your app if you use the .NET libraries, which most apps do. You can configure how your app uses newer versions of .NET. For more information, see [framework-dependent apps roll forward](#).

The following disadvantage only applies to .NET Core 2.1 SDK.

- **Use the `dotnet` command to start the app**

Users must run the `dotnet <filename.dll>` command to start your app. .NET Core 2.1 SDK doesn't produce platform-specific executables for apps published framework-dependent.

Examples

Publish an app cross-platform framework-dependent. An executable that targets your current platform is created along with the `dll` file.

```
dotnet publish
```

Publish an app cross-platform framework-dependent. A Linux 64-bit executable is created along with the `dll` file. This command doesn't work with .NET Core SDK 2.1.

```
dotnet publish -r linux-x64 --self-contained false
```

Publish self-contained

Publishing your app as self-contained produces a platform-specific executable. The output publishing folder contains all components of the app, including the .NET libraries and target runtime. The app is isolated from other .NET apps and doesn't use a locally installed shared runtime. The user of your app isn't required to download and install .NET.

The executable binary is produced for the specified target platform. For example, if you have an app named `word_reader`, and you publish a self-contained executable for Windows, a `word_reader.exe` file is created. Publishing for Linux or macOS, a `word_reader` file is created. The target platform and architecture is specified with the `-r <RID>` parameter for the `dotnet publish` command. For more information about RIDs, see [.NET RID Catalog](#).

If the app has platform-specific dependencies, such as a NuGet package containing platform-specific dependencies, these are copied to the publish folder along with the app.

Advantages

- **Control .NET version**

You control which version of .NET is deployed with your app.

- **Platform-specific targeting**

Because you have to publish your app for each platform, you know where your app will run. If .NET introduces a new platform, users can't run your app on that platform until you release a version targeting that platform. You can test your app for compatibility problems before your users run your app on the new platform.

Disadvantages

- **Larger deployments**

Because your app includes the .NET runtime and all of your app dependencies, the download size and hard drive space required is greater than a [framework-dependent](#) version.

TIP

You can reduce the size of your deployment on Linux systems by approximately 28 MB by using [.NET globalization invariant mode](#). This forces your app to treat all cultures like the [invariant culture](#).

TIP

[IL trimming](#) can further reduce the size of your deployment.

- **Harder to update the .NET version**

.NET Runtime (distributed with your app) can only be upgraded by releasing a new version of your app.

Examples

Publish an app self-contained. A macOS 64-bit executable is created.

```
dotnet publish -r osx-x64
```

Publish an app self-contained. A Windows 64-bit executable is created.

```
dotnet publish -r win-x64
```

Publish with ReadyToRun images

Publishing with ReadyToRun images will improve the startup time of your application at the cost of increasing the size of your application. In order to publish with ReadyToRun see [ReadyToRun](#) for more details.

Advantages

- **Improved startup time**

The application will spend less time running the JIT.

Disadvantages

- **Larger size**

The application will be larger on disk.

Examples

Publish an app self-contained and ReadyToRun. A macOS 64-bit executable is created.

```
dotnet publish -c Release -r osx-x64 -p:PublishReadyToRun=true
```

Publish an app self-contained and ReadyToRun. A Windows 64-bit executable is created.

```
dotnet publish -c Release -r win-x64 -p:PublishReadyToRun=true
```

See also

- [Deploying .NET Apps with .NET CLI.](#)
- [Deploying .NET Apps with Visual Studio.](#)
- [.NET Runtime Identifier \(RID\) catalog.](#)
- [Select the .NET version to use.](#)

Deploy .NET Core apps with Visual Studio

9/20/2022 • 15 minutes to read • [Edit Online](#)

You can deploy a .NET Core application either as a *framework-dependent deployment*, which includes your application binaries but depends on the presence of .NET Core on the target system, or as a *self-contained deployment*, which includes both your application and .NET Core binaries. For an overview of .NET Core application deployment, see [.NET Core Application Deployment](#).

The following sections show how to use Microsoft Visual Studio to create the following kinds of deployments:

- Framework-dependent deployment
- Framework-dependent deployment with third-party dependencies
- Self-contained deployment
- Self-contained deployment with third-party dependencies

For information on using Visual Studio to develop .NET Core applications, see [.NET Core dependencies and requirements](#).

Framework-dependent deployment

Deploying a framework-dependent deployment with no third-party dependencies simply involves building, testing, and publishing the app. A simple example written in C# illustrates the process.

1. Create the project.

Select **File > New > Project**. In the **New Project** dialog, expand your language's (C# or Visual Basic) project categories in the **Installed** project types pane, choose **.NET Core**, and then select the **Console App (.NET Core)** template in the center pane. Enter a project name, such as "FDD", in the **Name** text box. Select the **OK** button.

2. Add the application's source code.

Open the *Program.cs* or *Program.vb* file in the editor and replace the autogenerated code with the following code. It prompts the user to enter text and displays the individual words entered by the user. It uses the regular expression `\w+` to separate the words in the input text.

```
using System;
using System.Text.RegularExpressions;

namespace Applications.ConsoleApps
{
    public class ConsoleParser
    {
        public static void Main()
        {
            Console.WriteLine("Enter any text, followed by <Enter>:\n");
            String s = Console.ReadLine();
            ShowWords(s);
            Console.Write("\nPress any key to continue... ");
            Console.ReadKey();
        }

        private static void ShowWords(String s)
        {
            String pattern = @"\w+";
            var matches = Regex.Matches(s, pattern);
            if (matches.Count == 0)
            {
                Console.WriteLine("\nNo words were identified in your input.");
            }
            else
            {
                Console.WriteLine($"\\nThere are {matches.Count} words in your string:");
                for (int ctr = 0; ctr < matches.Count; ctr++)
                {
                    Console.WriteLine($"    #{ctr,2}: '{matches[ctr].Value}' at position
{matches[ctr].Index}");
                }
            }
        }
    }
}
```

```

Imports System.Text.RegularExpressions

Namespace Applications.ConsoleApps
    Public Module ConsoleParser
        Public Sub Main()
            Console.WriteLine("Enter any text, followed by <Enter>:")
            Console.WriteLine()
            Dim s = Console.ReadLine()
            ShowWords(s)
            Console.WriteLine($"{vbCrLf}Press any key to continue... ")
            Console.ReadKey()
        End Sub

        Private Sub ShowWords(s As String)
            Dim pattern = "\w+"
            Dim matches = Regex.Matches(s, pattern)
            Console.WriteLine()
            If matches.Count = 0 Then
                Console.WriteLine("No words were identified in your input.")
            Else
                Console.WriteLine($"There are {matches.Count} words in your string:")
                For ctr = 0 To matches.Count - 1
                    Console.WriteLine($"  #{ctr,2}: '{matches(ctr).Value}' at position
{matches(ctr).Index}")
                Next
            End If
            Console.WriteLine()
        End Sub
    End Module
End Namespace

```

3. Create a Debug build of your app.

Select **Build > Build Solution**. You can also compile and run the Debug build of your application by selecting **Debug > Start Debugging**.

4. Deploy your app.

After you've debugged and tested the program, create the files to be deployed with your app. To publish from Visual Studio, do the following:

- Change the solution configuration from **Debug** to **Release** on the toolbar to build a Release (rather than a Debug) version of your app.
- Right-click on the project (not the solution) in **Solution Explorer** and select **Publish**.
- In the **Publish** tab, select **Publish**. Visual Studio writes the files that comprise your application to the local file system.
- The **Publish** tab now shows a single profile, **FolderProfile**. The profile's configuration settings are shown in the **Summary** section of the tab.

The resulting files are placed in a directory named `Publish` on Windows and `publish` on Unix systems that is in a subdirectory of your project's `.\bin\release\netcoreapp2.1` subdirectory.

Along with your application's files, the publishing process emits a program database (.pdb) file that contains debugging information about your app. The file is useful primarily for debugging exceptions. You can choose not to package it with your application's files. You should, however, save it in the event that you want to debug the Release build of your app.

Deploy the complete set of application files in any way you like. For example, you can package them in a Zip file,

use a simple `copy` command, or deploy them with any installation package of your choice. Once installed, users can then execute your application by using the `dotnet` command and providing the application filename, such as `dotnet fdd.dll`.

In addition to the application binaries, your installer should also either bundle the shared framework installer or check for it as a prerequisite as part of the application installation. Installation of the shared framework requires Administrator/root access since it is machine-wide.

Framework-dependent deployment with third-party dependencies

Deploying a framework-dependent deployment with one or more third-party dependencies requires that any dependencies be available to your project. The following additional steps are required before you can build your app:

1. Use the **NuGet Package Manager** to add a reference to a NuGet package to your project; and if the package is not already available on your system, install it. To open the package manager, select **Tools > NuGet Package Manager > Manage NuGet Packages for Solution**.
2. Confirm that your third-party dependencies (for example, `Newtonsoft.Json`) are installed on your system and, if they aren't, install them. The **Installed** tab lists NuGet packages installed on your system. If `Newtonsoft.Json` is not listed there, select the **Browse** tab and enter "Newtonsoft.Json" in the search box. Select `Newtonsoft.Json` and, in the right pane, select your project before selecting **Install**.
3. If `Newtonsoft.Json` is already installed on your system, add it to your project by selecting your project in the right pane of the **Manage Packages for Solution** tab.

A framework-dependent deployment with third-party dependencies is only as portable as its third-party dependencies. For example, if a third-party library only supports macOS, the app isn't portable to Windows systems. This happens if the third-party dependency itself depends on native code. A good example of this is [Kestrel server](#), which requires a native dependency on [libuv](#). When an FDD is created for an application with this kind of third-party dependency, the published output contains a folder for each [Runtime Identifier \(RID\)](#) that the native dependency supports (and that exists in its NuGet package).

Self-contained deployment without third-party dependencies

Deploying a self-contained deployment with no third-party dependencies involves creating the project, modifying the `csproj` file, building, testing, and publishing the app. A simple example written in C# illustrates the process. You begin by creating, coding, and testing your project just as you would a framework-dependent deployment:

1. Create the project.

Select **File > New > Project**. In the **New Project** dialog, expand your language's (C# or Visual Basic) project categories in the **Installed** project types pane, choose **.NET Core**, and then select the **Console App (.NET Core)** template in the center pane. Enter a project name, such as "SCD", in the **Name** text box, and select the **OK** button.
2. Add the application's source code.

Open the `Program.cs` or `Program.vb` file in your editor, and replace the autogenerated code with the following code. It prompts the user to enter text and displays the individual words entered by the user. It uses the regular expression `\w+` to separate the words in the input text.

```
using System;
using System.Text.RegularExpressions;

namespace Applications.ConsoleApps
{
    public class ConsoleParser
    {
        public static void Main()
        {
            Console.WriteLine("Enter any text, followed by <Enter>:\n");
            String s = Console.ReadLine();
            ShowWords(s);
            Console.Write("\nPress any key to continue... ");
            Console.ReadKey();
        }

        private static void ShowWords(String s)
        {
            String pattern = @"\w+";
            var matches = Regex.Matches(s, pattern);
            if (matches.Count == 0)
            {
                Console.WriteLine("\nNo words were identified in your input.");
            }
            else
            {
                Console.WriteLine($"\\nThere are {matches.Count} words in your string:");
                for (int ctr = 0; ctr < matches.Count; ctr++)
                {
                    Console.WriteLine($"    #{ctr,2}: '{matches[ctr].Value}' at position
{matches[ctr].Index}");
                }
            }
        }
    }
}
```

```

Imports System.Text.RegularExpressions

Namespace Applications.ConsoleApps
    Public Module ConsoleParser
        Public Sub Main()
            Console.WriteLine("Enter any text, followed by <Enter>:")
            Console.WriteLine()
            Dim s = Console.ReadLine()
            ShowWords(s)
            Console.WriteLine($"{vbCrLf}Press any key to continue... ")
            Console.ReadKey()
        End Sub

        Private Sub ShowWords(s As String)
            Dim pattern = "\w+"
            Dim matches = Regex.Matches(s, pattern)
            Console.WriteLine()
            If matches.Count = 0 Then
                Console.WriteLine("No words were identified in your input.")
            Else
                Console.WriteLine($"There are {matches.Count} words in your string:")
                For ctr = 0 To matches.Count - 1
                    Console.WriteLine($"  #{ctr,2}: '{matches(ctr).Value}' at position
{matches(ctr).Index}")
                Next
            End If
            Console.WriteLine()
        End Sub
    End Module
End Namespace

```

3. Determine whether you want to use globalization invariant mode.

Particularly if your app targets Linux, you can reduce the total size of your deployment by taking advantage of [globalization invariant mode](#). Globalization invariant mode is useful for applications that are not globally aware and that can use the formatting conventions, casing conventions, and string comparison and sort order of the [invariant culture](#).

To enable invariant mode, right-click on your project (not the solution) in **Solution Explorer**, and select **Edit SCD.csproj** or **Edit SCD.vbproj**. Then add the following highlighted lines to the file:

```

<Project Sdk="Microsoft.NET.Sdk">

    <PropertyGroup>
        <TargetFramework>netcoreapp3.1</TargetFramework>
    </PropertyGroup>

    <ItemGroup>
        <RuntimeHostConfigurationOption Include="System.Globalization.Invariant" Value="true" />
    </ItemGroup>

</Project>

```

4. Create a Debug build of your application.

Select **Build > Build Solution**. You can also compile and run the Debug build of your application by selecting **Debug > Start Debugging**. This debugging step lets you identify problems with your application when it's running on your host platform. You still will have to test it on each of your target platforms.

If you've enabled globalization invariant mode, be particularly sure to test whether the absence of

culture-sensitive data is suitable for your application.

Once you've finished debugging, you can publish your self-contained deployment:

- [Visual Studio 15.6 and earlier](#)
- [Visual Studio 15.7 and later](#)

After you've debugged and tested the program, create the files to be deployed with your app for each platform that it targets.

To publish your app from Visual Studio, do the following:

1. Define the platforms that your app will target.
 - a. Right-click on your project (not the solution) in **Solution Explorer** and select **Edit SCD.csproj**.
 - b. Create a `<RuntimeIdentifiers>` tag in the `<PropertyGroup>` section of your *csproj* file that defines the platforms your app targets, and specify the runtime identifier (RID) of each platform that you target. You also need to add a semicolon to separate the RIDs. See [Runtime identifier catalog](#) for a list of runtime identifiers.

For example, the following example indicates that the app runs on 64-bit Windows 10 operating systems and the 64-bit OS X Version 10.11 operating system.

```
<PropertyGroup>
  <RuntimeIdentifiers>win10-x64;osx.10.11-x64</RuntimeIdentifiers>
</PropertyGroup>
```

The `<RuntimeIdentifiers>` element can go into any `<PropertyGroup>` that you have in your *csproj* file. A complete sample *csproj* file appears later in this section.

2. Publish your app.

After you've debugged and tested the program, create the files to be deployed with your app for each platform that it targets.

To publish your app from Visual Studio, do the following:

- a. Change the solution configuration from **Debug** to **Release** on the toolbar to build a Release (rather than a Debug) version of your app.
- b. Right-click on the project (not the solution) in **Solution Explorer** and select **Publish**.
- c. In the **Publish** tab, select **Publish**. Visual Studio writes the files that comprise your application to the local file system.
- d. The **Publish** tab now shows a single profile, **FolderProfile**. The profile's configuration settings are shown in the **Summary** section of the tab. **Target Runtime** identifies which runtime has been published, and **Target Location** identifies where the files for the self-contained deployment were written.
- e. Visual Studio by default writes all published files to a single directory. For convenience, it's best to create separate profiles for each target runtime and to place published files in a platform-specific directory. This involves creating a separate publishing profile for each target platform. So now rebuild the application for each platform by doing the following:
 - a. Select **Create new profile** in the **Publish** dialog.
 - b. In the **Pick a publish target** dialog, change the **Choose a folder** location to

`bin\Release\PublishOutput\win10-x64`. Select **OK**.

- c. Select the new profile (**FolderProfile1**) in the list of profiles, and make sure that the **Target Runtime** is `win10-x64`. If it isn't, select **Settings**. In the **Profile Settings** dialog, change the **Target Runtime** to `win10-x64` and select **Save**. Otherwise, select **Cancel**.
- d. Select **Publish** to publish your app for 64-bit Windows 10 platforms.
- e. Follow the previous steps again to create a profile for the `osx.10.11-x64` platform. The **Target Location** is `bin\Release\PublishOutput\osx.10.11-x64`, and the **Target Runtime** is `osx.10.11-x64`. The name that Visual Studio assigns to this profile is **FolderProfile2**.

Each target location contains the complete set of files (both your app files and all .NET Core files) needed to launch your app.

Along with your application's files, the publishing process emits a program database (.pdb) file that contains debugging information about your app. The file is useful primarily for debugging exceptions. You can choose not to package it with your application's files. You should, however, save it in the event that you want to debug the Release build of your app.

Deploy the published files in any way you like. For example, you can package them in a Zip file, use a simple `copy` command, or deploy them with any installation package of your choice.

The following is the complete *csproj* file for this project.

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>netcoreapp2.1</TargetFramework>
    <RuntimeIdentifiers>win10-x64;osx.10.11-x64</RuntimeIdentifiers>
  </PropertyGroup>
</Project>
```

Self-contained deployment with third-party dependencies

Deploying a self-contained deployment with one or more third-party dependencies involves adding the dependencies. The following additional steps are required before you can build your app:

1. Use the **NuGet Package Manager** to add a reference to a NuGet package to your project; and if the package is not already available on your system, install it. To open the package manager, select **Tools** > **NuGet Package Manager** > **Manage NuGet Packages for Solution**.
2. Confirm that your third-party dependencies (for example, `Newtonsoft.Json`) are installed on your system and, if they aren't, install them. The **Installed** tab lists NuGet packages installed on your system. If `Newtonsoft.Json` is not listed there, select the **Browse** tab and enter "Newtonsoft.Json" in the search box. Select `Newtonsoft.Json` and, in the right pane, select your project before selecting **Install**.
3. If `Newtonsoft.Json` is already installed on your system, add it to your project by selecting your project in the right pane of the **Manage Packages for Solution** tab.

The following is the complete *csproj* file for this project:

- [Visual Studio 15.6 and earlier](#)
- [Visual Studio 15.7 and later](#)

```
<Project Sdk="Microsoft.NET.Sdk">
<PropertyGroup>
<OutputType>Exe</OutputType>
<TargetFramework>netcoreapp2.1</TargetFramework>
<RuntimeIdentifiers>win10-x64;osx.10.11-x64</RuntimeIdentifiers>
</PropertyGroup>
<ItemGroup>
<PackageReference Include="Newtonsoft.Json" Version="10.0.2" />
</ItemGroup>
</Project>
```

When you deploy your application, any third-party dependencies used in your app are also contained with your application files. Third-party libraries aren't required on the system on which the app is running.

You can only deploy a self-contained deployment with a third-party library to platforms supported by that library. This is similar to having third-party dependencies with native dependencies in your framework-dependent deployment, where the native dependencies won't exist on the target platform unless they were previously installed there.

See also

- [.NET Core Application Deployment](#)
- [.NET Core Runtime Identifier \(RID\) catalog](#)

Publish .NET apps with the .NET CLI

9/20/2022 • 8 minutes to read • [Edit Online](#)

This article demonstrates how you can publish your .NET application from the command line. .NET provides three ways to publish your applications. Framework-dependent deployment produces a cross-platform .dll file that uses the locally installed .NET runtime. Framework-dependent executable produces a platform-specific executable that uses the locally installed .NET runtime. Self-contained executable produces a platform-specific executable and includes a local copy of the .NET runtime.

For an overview of these publishing modes, see [.NET Application Deployment](#).

Looking for some quick help on using the CLI? The following table shows some examples of how to publish your app. You can specify the target framework with the `-f <TFM>` parameter or by editing the project file. For more information, see [Publishing basics](#).

PUBLISH MODE	SDK VERSION	COMMAND
Framework-dependent deployment	2.1	<code>dotnet publish -c Release</code>
	3.1	<code>dotnet publish -c Release -p:UseAppHost=false</code>
	5.0	<code>dotnet publish -c Release -p:UseAppHost=false</code>
	6.0	<code>dotnet publish -c Release -p:UseAppHost=false</code>
Framework-dependent executable	3.1	<code>dotnet publish -c Release -r <RID> --self-contained false</code> <code>dotnet publish -c Release</code>
	5.0	<code>dotnet publish -c Release -r <RID> --self-contained false</code> <code>dotnet publish -c Release</code>
	6.0	<code>dotnet publish -c Release -r <RID> --self-contained false</code> <code>dotnet publish -c Release</code>
	2.1	<code>dotnet publish -c Release -r <RID> --self-contained true</code>
Self-contained deployment	3.1	<code>dotnet publish -c Release -r <RID> --self-contained true</code>
	5.0	<code>dotnet publish -c Release -r <RID> --self-contained true</code>
	6.0	<code>dotnet publish -c Release -r <RID> --self-contained true</code>

* When using **SDK version 3.1** or higher, framework-dependent executable is the default publishing mode

when running the basic `dotnet publish` command.

NOTE

The `-c Release` parameter isn't required. It's provided as a reminder to publish the **Release** build of your app.

Publishing basics

The `<TargetFramework>` setting of the project file specifies the default target framework when you publish your app. You can change the target framework to any valid [Target Framework Moniker \(TFM\)](#). For example, if your project uses `<TargetFramework>netcoreapp2.1</TargetFramework>`, a binary that targets .NET Core 2.1 is created. The TFM specified in this setting is the default target used by the `dotnet publish` command.

If you want to target more than one framework, you can set the `<TargetFrameworks>` setting to multiple TFM values, separated by a semicolon. When you build your app, a build is produced for each target framework. However, when you publish your app, you must specify the target framework with the `dotnet publish -f <TFM>` command.

Unless otherwise set, the output directory of the `dotnet publish` command is

`./bin/<BUILD-CONFIGURATION>/<TFM>/publish/`. The default BUILD-CONFIGURATION mode is **Debug** unless changed with the `-c` parameter. For example, `dotnet publish -c Release -f netcoreapp2.1` publishes to `./bin/Release/netcoreapp2.1/publish/`.

If you use .NET Core SDK 3.1 or later, the default publish mode is framework-dependent *executable*.

If you use .NET Core SDK 2.1, the default publish mode is framework-dependent *deployment*.

Native dependencies

If your app has native dependencies, it may not run on a different operating system. For example, if your app uses the native Windows API, it won't run on macOS or Linux. You would need to provide platform-specific code and compile an executable for each platform.

Consider also, if a library you referenced has a native dependency, your app may not run on every platform. However, it's possible a NuGet package you're referencing has included platform-specific versions to handle the required native dependencies for you.

When distributing an app with native dependencies, you may need to use the `dotnet publish -r <RID>` switch to specify the target platform you want to publish for. For a list of runtime identifiers, see [Runtime Identifier \(RID\) catalog](#).

More information about platform-specific binaries is covered in the [Framework-dependent executable](#) and [Self-contained deployment](#) sections.

Sample app

You can use the following app to explore the publishing commands. The app is created by running the following commands in your terminal:

```
mkdir apptest1
cd apptest1
dotnet new console
dotnet add package Figgle
```

The `Program.cs` or `Program.vb` file that is generated by the console template needs to be changed to the following:

```
using System;

namespace apptest1
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine(Figgle.FiggleFonts.Standard.Render("Hello, World!"));
        }
    }
}
```

```
Module Program
Sub Main(args As String())
    Console.WriteLine(Figgle.FiggleFonts.Standard.Render("Hello, World!"))
End Sub
End Module
```

When you run the app (`dotnet run`), the following output is displayed:

Framework-dependent deployment

For the .NET Core 2.1 SDK CLI, framework-dependent deployment (FDD) is the default mode for the basic `dotnet publish` command. In newer SDKs, [Framework-dependent executable](#) is the default.

When you publish your app as an FDD, a `<PROJECT-NAME>.dll` file is created in the `./bin/<BUILD-CONFIGURATION>/<TFM>/publish/` folder. To run your app, navigate to the output folder and use the `dotnet <PROJECT-NAME>.dll` command.

Your app is configured to target a specific version of .NET. That targeted .NET runtime is required to be on any machine where your app runs. For example, if your app targets .NET Core 3.1, any machine that your app runs on must have the .NET Core 3.1 runtime installed. As stated in the [Publishing basics](#) section, you can edit your project file to change the default target framework or to target more than one framework.

Publishing an FDD creates an app that automatically rolls-forward to the latest .NET security patch available on the system that runs the app. For more information on version binding at compile time, see [Select the .NET version to use](#).

PUBLISH MODE	SDK VERSION	COMMAND
Framework-dependent deployment	2.1	<code>dotnet publish -c Release</code>
	3.1	<code>dotnet publish -c Release -p:UseAppHost=false</code>
	5.0	<code>dotnet publish -c Release -p:UseAppHost=false</code>

PUBLISH MODE	SDK VERSION	COMMAND
	6.0	<code>dotnet publish -c Release -p:UseAppHost=false</code>

Framework-dependent executable

For the .NET 5 (and .NET Core 3.1) SDK CLI, framework-dependent executable (FDE) is the default mode for the basic `dotnet publish` command. You don't need to specify any other parameters, as long as you want to target the current operating system.

In this mode, a platform-specific executable host is created to host your cross-platform app. This mode is similar to FDD, as FDD requires a host in the form of the `dotnet` command. The host executable filename varies per platform and is named something similar to `<PROJECT-FILE>.exe`. You can run this executable directly instead of calling `dotnet <PROJECT-FILE>.dll`, which is still an acceptable way to run the app.

Your app is configured to target a specific version of .NET. That targeted .NET runtime is required to be on any machine where your app runs. For example, if your app targets .NET Core 3.1, any machine that your app runs on must have the .NET Core 3.1 runtime installed. As stated in the [Publishing basics](#) section, you can edit your project file to change the default target framework or to target more than one framework.

Publishing an FDE creates an app that automatically rolls-forward to the latest .NET security patch available on the system that runs the app. For more information on version binding at compile time, see [Select the .NET version to use](#).

For .NET 2.1, you must use the following switches with the `dotnet publish` command to publish an FDE:

- `-r <RID>` This switch uses an identifier (RID) to specify the target platform. For a list of runtime identifiers, see [Runtime Identifier \(RID\) catalog](#).
- `--self-contained false` This switch disables the default behavior of the `-r` switch, which is to create a self-contained deployment (SCD). This switch creates an FDE.

PUBLISH MODE	SDK VERSION	COMMAND
Framework-dependent executable	3.1	<code>dotnet publish -c Release -r <RID> --self-contained false</code> <code>dotnet publish -c Release</code>
	5.0	<code>dotnet publish -c Release -r <RID> --self-contained false</code> <code>dotnet publish -c Release</code>
	6.0	<code>dotnet publish -c Release -r <RID> --self-contained false</code> <code>dotnet publish -c Release</code>

Whenever you use the `-r` switch, the output folder path changes to:

```
./bin/<BUILD-CONFIGURATION>/<TFM>/<RID>/publish/
```

If you use the [example app](#), run `dotnet publish -f net6.0 -r win10-x64 --self-contained false`. This command creates the following executable: `./bin/Debug/net6.0/win10-x64/publish/apptest1.exe`

NOTE

You can reduce the total size of your deployment by enabling **globalization invariant mode**. This mode is useful for applications that are not globally aware and that can use the formatting conventions, casing conventions, and string comparison and sort order of the [invariant culture](#). For more information about **globalization invariant mode** and how to enable it, see [.NET Globalization Invariant Mode](#).

Self-contained deployment

When you publish a self-contained deployment (SCD), the .NET SDK creates a platform-specific executable. Publishing an SCD includes all required .NET files to run your app but it doesn't include the [native dependencies of .NET](#). These dependencies must be present on the system before the app runs.

Publishing an SCD creates an app that doesn't roll forward to the latest available .NET security patch. For more information on version binding at compile time, see [Select the .NET version to use](#).

You must use the following switches with the `dotnet publish` command to publish an SCD:

- `-r <RID>` This switch uses an identifier (RID) to specify the target platform. For a list of runtime identifiers, see [Runtime Identifier \(RID\) catalog](#).
- `--self-contained true` This switch tells the .NET SDK to create an executable as an SCD.

PUBLISH MODE	SDK VERSION	COMMAND
Self-contained deployment	2.1	<code>dotnet publish -c Release -r <RID> --self-contained true</code>
	3.1	<code>dotnet publish -c Release -r <RID> --self-contained true</code>
	5.0	<code>dotnet publish -c Release -r <RID> --self-contained true</code>
	6.0	<code>dotnet publish -c Release -r <RID> --self-contained true</code>

TIP

In .NET 6 you can reduce the total size of compatible self-contained apps by [publishing trimmed](#). This enables the trimmer to remove parts of the framework and referenced assemblies that are not on any code path or potentially referenced in [runtime reflection](#). See [trimming incompatibilities](#) to determine if trimming makes sense for your application.

NOTE

You can reduce the total size of your deployment by enabling **globalization invariant mode**. This mode is useful for applications that are not globally aware and that can use the formatting conventions, casing conventions, and string comparison and sort order of the [invariant culture](#). For more information about **globalization invariant mode** and how to enable it, see [.NET Core Globalization Invariant Mode](#).

See also

- [.NET Application Deployment Overview](#)
- [.NET Runtime Identifier \(RID\) catalog](#)

How to create a NuGet package with the .NET CLI

9/20/2022 • 2 minutes to read • [Edit Online](#)

NOTE

The following shows command-line samples using Unix. The `dotnet pack` command as shown here works the same way on Windows.

.NET Standard and .NET Core libraries are expected to be distributed as NuGet packages. This is in fact how all of the .NET Standard libraries are distributed and consumed. This is most easily done with the `dotnet pack` command.

Imagine that you just wrote an awesome new library that you would like to distribute over NuGet. You can create a NuGet package with cross-platform tools to do exactly that! The following example assumes a library called `SuperAwesomeLibrary` that targets `netstandard1.0`.

If you have transitive dependencies, that is, a project that depends on another package, make sure to restore packages for the entire solution with the `dotnet restore` command before you create a NuGet package. Failing to do so will result in the `dotnet pack` command not working properly.

You don't have to run `dotnet restore` because it's run implicitly by all commands that require a restore to occur, such as `dotnet new`, `dotnet build`, `dotnet run`, `dotnet test`, `dotnet publish`, and `dotnet pack`. To disable implicit restore, use the `--no-restore` option.

The `dotnet restore` command is still useful in certain scenarios where explicitly restoring makes sense, such as [continuous integration builds in Azure DevOps Services](#) or in build systems that need to explicitly control when the restore occurs.

For information about how to manage NuGet feeds, see the [dotnet restore](#) documentation.

After ensuring packages are restored, you can navigate to the directory where a library lives:

```
cd src/SuperAwesomeLibrary
```

Then it's just a single command from the command line:

```
dotnet pack
```

Your `/bin/Debug` folder will now look like this:

```
$ ls bin/Debug  
netstandard1.0/  
SuperAwesomeLibrary.1.0.0.nupkg  
SuperAwesomeLibrary.1.0.0.symbols.nupkg
```

This produces a package that is capable of being debugged. If you want to build a NuGet package with release binaries, all you need to do is add the `--configuration` (or `-c`) switch and use `release` as the argument.

```
dotnet pack --configuration release
```

Your `/bin` folder will now have a `release` folder containing your NuGet package with release binaries:

```
$ ls bin/release  
netstandard1.0/  
SuperAwesomeLibrary.1.0.0.nupkg  
SuperAwesomeLibrary.1.0.0.symbols.nupkg
```

And now you have the necessary files to publish a NuGet package!

Don't confuse `dotnet pack` with `dotnet publish`

It is important to note that at no point is the `dotnet publish` command involved. The `dotnet publish` command is for deploying applications with all of their dependencies in the same bundle -- not for generating a NuGet package to be distributed and consumed via NuGet.

See also

- [Quickstart: Create and publish a package](#)

Self-contained deployment runtime roll forward

9/20/2022 • 2 minutes to read • [Edit Online](#)

.NET Core [self-contained application deployments](#) include both the .NET Core libraries and the .NET Core runtime. Starting in .NET Core 2.1 SDK (version 2.1.300), a self-contained application deployment [publishes the highest patch runtime on your machine](#). By default, `dotnet publish` for a self-contained deployment selects the latest version installed as part of the SDK on the publishing machine. This enables your deployed application to run with security fixes (and other fixes) available during `publish`. The application must be republished to obtain a new patch. Self-contained applications are created by specifying `-r <RID>` on the `dotnet publish` command or by specifying the [runtime identifier \(RID\)](#) in the project file (csproj / vbproj) or on the command line.

Patch version roll forward overview

`restore`, `build` and `publish` are `dotnet` commands that can run separately. The runtime choice is part of the `restore` operation, not `publish` or `build`. If you call `publish`, the latest patch version will be chosen. If you call `publish` with the `--no-restore` argument, then you may not get the desired patch version because a prior `restore` may not have been executed with the new self-contained application publishing policy. In this case, a build error is generated with text similar to the following:

"The project was restored using Microsoft.NETCore.App version 2.0.0, but with current settings, version 2.0.6 would be used instead. To resolve this issue, make sure the same settings are used for restore and for subsequent operations such as build or publish. Typically this issue can occur if the RuntimeIdentifier property is set during build or publish but not during restore."

NOTE

`restore` and `build` can be run implicitly as part of another command, like `publish`. When run implicitly as part of another command, they are provided with additional context so that the right artifacts are produced. When you `publish` with a runtime (for example, `dotnet publish -r linux-x64`), the implicit `restore` restores packages for the linux-x64 runtime. If you call `restore` explicitly, it does not restore runtime packages by default, because it doesn't have that context.

How to avoid restore during publish

Running `restore` as part of the `publish` operation may be undesirable for your scenario. To avoid `restore` during `publish` while creating self-contained applications, do the following:

- Set the `RuntimeIdentifiers` property to a semicolon-separated list of all the [RIDs](#) to be published.
- Set the `TargetLatestRuntimePatch` property to `true`.

No-restore argument with dotnet publish options

If you want to create both self-contained applications and [framework-dependent applications](#) with the same project file, and you want to use the `--no-restore` argument with `dotnet publish`, then choose one of the following:

1. Prefer the framework-dependent behavior. If the application is framework-dependent, this is the default behavior. If the application is self-contained, and can use an unpatched 2.1.0 local runtime, set the `TargetLatestRuntimePatch` to `false` in the project file.

2. Prefer the self-contained behavior. If the application is self-contained, this is the default behavior. If the application is framework-dependent, and requires the latest patch installed, set `TargetLatestRuntimePatch` to `true` in the project file.
3. Take explicit control of the runtime framework version by setting `RuntimeFrameworkVersion` to the specific patch version in the project file.

Single-file deployment and executable

9/20/2022 • 8 minutes to read • [Edit Online](#)

Bundling all application-dependent files into a single binary provides an application developer with the attractive option to deploy and distribute the application as a single file. Single-file deployment is available for both the [framework-dependent deployment model](#) and [self-contained applications](#).

This deployment model has been available since .NET Core 3.0 and has been enhanced in .NET 5. Previously in .NET Core 3.0, when a user runs your single file app, .NET Core host first extracts all files to a directory before running the application. .NET 5 improves this experience by directly running the code without the need to extract the files from the app.

The size of the single file in a self-contained application is large since it includes the runtime and the framework libraries. In .NET 6, you can [publish trimmed](#) to reduce the total size of trim-compatible applications. The single file deployment option can be combined with [ReadyToRun](#) and [Trim](#) publish options.

Single file deployment isn't compatible with Windows 7.

Sample project file

Here's a sample project file that specifies single file publishing:

```
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>net6.0</TargetFramework>
    <PublishSingleFile>true</PublishSingleFile>
    <SelfContained>true</SelfContained>
    <RuntimeIdentifier>win-x64</RuntimeIdentifier>
    <PublishReadyToRun>true</PublishReadyToRun>
  </PropertyGroup>

</Project>
```

These properties have the following functions:

- `PublishSingleFile`. Enables single file publishing. Also enables single file warnings during `dotnet build`.
- `SelfContained`. Determines whether the app is self-contained or framework-dependent.
- `RuntimeIdentifier`. Specifies the [OS and CPU type](#) you're targeting. Also sets `<SelfContained>true</SelfContained>` by default.
- `PublishReadyToRun`. Enables [ahead-of-time \(AOT\) compilation](#).

Single file apps are always OS and architecture specific. You need to publish for each configuration, such as Linux x64, Linux Arm64, Windows x64, and so forth.

Runtime configuration files, such as `*runtimeconfig.json` and `*deps.json`, are included in the single file. If an extra configuration file is needed, you can place it beside the single file.

Publish a single-file app

- [CLI](#)
- [Visual Studio](#)

- [Visual Studio for Mac](#)

Publish a single file application using the [dotnet publish](#) command.

1. Add `<PublishSingleFile>true</PublishSingleFile>` to your project file.

This change produces a single file app on self-contained publish. It also shows single file compatibility warnings during build.

```
<PropertyGroup>
  <PublishSingleFile>true</PublishSingleFile>
</PropertyGroup>
```

2. Publish the app for a specific runtime identifier using `dotnet publish -r <RID>`

The following example publishes the app for Windows as a self-contained single file application.

```
dotnet publish -r win-x64
```

The following example publishes the app for Linux as a framework dependent single file application.

```
dotnet publish -r linux-x64 --self-contained false
```

`<PublishSingleFile>` should be set in the project file to enable file analysis during build, but it's also possible to pass these options as `dotnet publish` arguments:

```
dotnet publish -r linux-x64 -p:PublishSingleFile=true --self-contained false
```

For more information, see [Publish .NET Core apps with .NET CLI](#).

Exclude files from being embedded

Certain files can be explicitly excluded from being embedded in the single file by setting the following metadata:

```
<ExcludeFromSingleFile>true</ExcludeFromSingleFile>
```

For example, to place some files in the publish directory but not bundle them in the file:

```
<ItemGroup>
  <Content Update="Plugin.dll">
    <CopyToPublishDirectory>PreserveNewest</CopyToPublishDirectory>
    <ExcludeFromSingleFile>true</ExcludeFromSingleFile>
  </Content>
</ItemGroup>
```

Include PDB files inside the bundle

The PDB file for an assembly can be embedded into the assembly itself (the `.dll`) using the setting below. Since the symbols are part of the assembly, they're part of the application as well:

```
<DebugType>embedded</DebugType>
```

For example, add the following property to the project file of an assembly to embed the PDB file to that assembly:

```
<PropertyGroup>
  <DebugType>embedded</DebugType>
</PropertyGroup>
```

Other considerations

Single file applications have all related PDB files alongside the application, not bundled by default. If you want to include PDBs inside the assembly for projects you build, set the `DebugType` to `embedded`. See [Include PDB files inside the bundle](#).

Managed C++ components aren't well suited for single file deployment. We recommend that you write applications in C# or another non-managed C++ language to be single file compatible.

Output differences from .NET 3.x

In .NET Core 3.x, publishing as a single file produced one file, consisting of the app itself, dependencies, and any other files in the folder during publish. When the app starts, the single file app was extracted to a folder and run from there.

Starting with .NET 5, only managed DLLs are bundled with the app into a single executable. When the app starts, the managed DLLs are extracted and loaded in memory, avoiding the extraction to a folder. On Windows, this approach means that the managed binaries are embedded in the single file bundle, but the native binaries of the core runtime itself are separate files.

To embed those files for extraction and get one output file, like in .NET Core 3.x, set the property

`IncludeNativeLibrariesForSelfExtract` to `true`. For more information about extraction, see [Including native libraries](#).

API incompatibility

Some APIs aren't compatible with single file deployment. Applications might require modification if they use these APIs. If you use a third-party framework or package, it's possible that they might use one of these APIs and need modification. The most common cause of problems is dependence on file paths for files or DLLs shipped with the application.

The table below has the relevant runtime library API details for single file use.

API	NOTE
<code>Assembly.CodeBase</code>	Throws PlatformNotSupportedException .
<code>Assembly.EscapedCodeBase</code>	Throws PlatformNotSupportedException .
<code>Assembly.GetFile</code>	Throws IOException .
<code>Assembly.GetFiles</code>	Throws IOException .
<code>Assembly.Location</code>	Returns an empty string.
<code>AssemblyName.CodeBase</code>	Returns <code>null</code> .
<code>AssemblyName.EscapedCodeBase</code>	Returns <code>null</code> .
<code>Module.FullyQualifiedName</code>	Returns a string with the value of <code><Unknown></code> or throws an exception.

API	NOTE
<code>Marshal.GetHINSTANCE</code>	Returns -1.
<code>Module.Name</code>	Returns a string with the value of <code><Unknown></code> .

We have some recommendations for fixing common scenarios:

- To access files next to the executable, use [ApplicationContext.BaseDirectory](#).
- To find the file name of the executable, use the first element of [Environment.GetCommandLineArgs\(\)](#), or starting with .NET 6, use the file name from [ProcessPath](#).
- To avoid shipping loose files entirely, consider using [embedded resources](#).

Attach a debugger

On Linux, the only debugger that can attach to self-contained single file processes or debug crash dumps is [SOS with LLDB](#).

On Windows and Mac, Visual Studio and VS Code can be used to debug crash dumps. Attaching to a running self-contained single file executable requires an extra file: *mscordbi.{dll,so}*.

Without this file, Visual Studio might produce the error: "Unable to attach to the process. A debug component is not installed." VS Code might produce the error: "Failed to attach to process: Unknown Error: 0x80131c3c."

To fix these errors, *mscordbi* needs to be copied next to the executable. *mscordbi* is [published](#) by default in the subdirectory with the application's runtime ID. So, for example, if you publish a self-contained single file executable using the `dotnet` CLI for Windows using the parameters `-r win-x64`, the executable would be placed in *bin/Debug/net5.0/win-x64/publish*. A copy of *mscordbi.dll* would be present in *bin/Debug/net5.0/win-x64*.

Include native libraries

Single file deployment doesn't bundle native libraries by default. On Linux, the runtime is prelinked into the bundle and only application native libraries are deployed to the same directory as the single file app. On Windows, only the hosting code is prelinked and both the runtime and application native libraries are deployed to the same directory as the single file app. This approach is to ensure a good debugging experience, which requires native files to be excluded from the single file.

Starting with .NET 6, the runtime is prelinked into the bundle on all platforms.

You can set a flag, `IncludeNativeLibrariesForSelfExtract`, to include native libraries in the single file bundle. These files are extracted to a directory in the client machine when the single file application is run.

Specifying `IncludeAllContentForSelfExtract` extracts all files, including the managed assemblies, before running the executable. This approach preserves the original .NET Core single file deployment behavior.

NOTE

If extraction is used, the files are extracted to disk before the app starts:

- If environment variable `DOTNET_BUNDLE_EXTRACT_BASE_DIR` is set to a path, the files are extracted to a directory under that path.
- Otherwise, if running on Linux or MacOS, the files are extracted to a directory under `$HOME/.net`.
- If running on Windows, the files are extracted to a directory under `%TEMP%/.net`.

To prevent tampering, these directories should not be writable by users or services with different privileges. Don't use `/tmp` or `/var/tmp` on most Linux and MacOS systems.

NOTE

In some Linux environments, such as under `systemd`, the default extraction does not work because `$HOME` is not defined. In such cases, we recommend that you set `$DOTNET_BUNDLE_EXTRACT_BASE_DIR` explicitly.

For `systemd`, a good alternative seems to be defining `DOTNET_BUNDLE_EXTRACT_BASE_DIR` in your service's unit file as `%h/.net`, which `systemd` expands correctly to `$HOME/.net` for the account running the service.

```
[Service]
Environment="DOTNET_BUNDLE_EXTRACT_BASE_DIR=%h/.net"
```

Compress assemblies in single-file apps

Starting with .NET 6, single file apps can be created with compression enabled on the embedded assemblies. Set the `EnableCompressionInSingleFile` property to `true`. The single file that's produced will have all of the embedded assemblies compressed, which can significantly reduce the size of the executable.

Compression comes with a performance cost. On application start, the assemblies must be decompressed into memory, which takes some time. We recommend that you measure both the size change and startup cost of enabling compression before using it. The impact can vary significantly between different applications.

See also

- [.NET Core application deployment](#)
- [Publish .NET apps with .NET CLI](#)
- [Publish .NET Core apps with Visual Studio](#)
- [`dotnet publish` command](#)

ReadyToRun Compilation

9/20/2022 • 4 minutes to read • [Edit Online](#)

.NET application startup time and latency can be improved by compiling your application assemblies as ReadyToRun (R2R) format. R2R is a form of ahead-of-time (AOT) compilation.

R2R binaries improve startup performance by reducing the amount of work the just-in-time (JIT) compiler needs to do as your application loads. The binaries contain similar native code compared to what the JIT would produce. However, R2R binaries are larger because they contain both intermediate language (IL) code, which is still needed for some scenarios, and the native version of the same code. R2R is only available when you publish an app that targets specific runtime environments (RID) such as Linux x64 or Windows x64.

To compile your project as ReadyToRun, the application must be published with the `PublishReadyToRun` property set to `true`.

There are two ways to publish your app as ReadyToRun:

1. Specify the `PublishReadyToRun` flag directly to the `dotnet publish` command. See [dotnet publish](#) for details.

```
dotnet publish -c Release -r win-x64 -p:PublishReadyToRun=true
```

2. Specify the property in the project.

- Add the `<PublishReadyToRun>` setting to your project.

```
<PropertyGroup>
  <PublishReadyToRun>true</PublishReadyToRun>
</PropertyGroup>
```

- Publish the application without any special parameters.

```
dotnet publish -c Release -r win-x64
```

Impact of using the ReadyToRun feature

Ahead-of-time compilation has complex performance impact on application performance, which can be difficult to predict. In general, the size of an assembly will grow to between two to three times larger. This increase in the physical size of the file may reduce the performance of loading the assembly from disk, and increase working set of the process. However, in return the number of methods compiled at run time is typically reduced substantially. The result is that most applications that have large amounts of code receive large performance benefits from enabling ReadyToRun. Applications that have small amounts of code will likely not experience a significant improvement from enabling ReadyToRun, as the .NET runtime libraries have already been precompiled with ReadyToRun.

The startup improvement discussed here applies not only to application startup, but also to the first use of any code in the application. For instance, ReadyToRun can be used to reduce the response latency of the first use of Web API in an ASP.NET application.

Interaction with tiered compilation

Ahead-of-time generated code is not as highly optimized as code produced by the JIT. To address this issue, tiered compilation will replace commonly used ReadyToRun methods with JIT-generated methods.

How is the set of precompiled assemblies chosen?

The SDK will precompile the assemblies that are distributed with the application. For self-contained applications, this set of assemblies will include the framework. C++/CLI binaries are not eligible for ReadyToRun compilation.

To exclude specific assemblies from ReadyToRun processing, use the `<PublishReadyToRunExclude>` list.

```
<ItemGroup>
  <PublishReadyToRunExclude Include="Contoso.Example.dll" />
</ItemGroup>
```

How is the set of methods to precompile chosen?

The compiler will attempt to pre-compile as many methods as it can. However, for various reasons, it's not expected that using the ReadyToRun feature will prevent the JIT from executing. Such reasons may include, but are not limited to:

- Use of generic types defined in separate assemblies.
- Interop with native code.
- Use of hardware intrinsics that the compiler cannot prove are safe to use on a target machine.
- Certain unusual IL patterns.
- Dynamic method creation via reflection or LINQ.

Symbol generation for use with profilers

When compiling an application with ReadyToRun, profilers may require symbols for examining the generated ReadyToRun files. To enable symbol generation, specify the `<PublishReadyToRunEmitSymbols>` property.

```
<PropertyGroup>
  <PublishReadyToRunEmitSymbols>true</PublishReadyToRunEmitSymbols>
</PropertyGroup>
```

These symbols will be placed in the publish directory and for Windows will have a file extension of .ni.pdb, and for Linux will have a file extension of .r2rmap. These files are not generally redistributed to end customers, but instead would typically be stored in a symbol server. In general these symbols are useful for debugging performance issues related to startup of applications, as [Tiered Compilation](#) will replace the ReadyToRun generated code with dynamically generated code. However, if attempting to profile an application that disables [Tiered Compilation](#) the symbols will be useful.

Composite ReadyToRun

Normal ReadyToRun compilation produces binaries that can be serviced and manipulated individually. Starting in .NET 6, support for Composite ReadyToRun compilation has been added. Composite ReadyToRun compiles a set of assemblies that must be distributed together. This has the advantage that the compiler is able to perform better optimizations and reduces the set of methods that cannot be compiled via the ReadyToRun process. However, as a tradeoff, compilation speed is significantly decreased, and the overall file size of the application is significantly increased. Due to these tradeoffs, use of Composite ReadyToRun is only recommended for applications that disable [Tiered Compilation](#) or applications running on Linux that are seeking the best startup time with [self-contained](#) deployment. To enable composite ReadyToRun compilation, specify the `<PublishReadyToRunComposite>` property.

```

<PropertyGroup>
  <PublishReadyToRunComposite>true</PublishReadyToRunComposite>
</PropertyGroup>

```

NOTE

In .NET 6, Composite ReadyToRun is only supported for [self-contained](#) deployment.

Cross platform/architecture restrictions

For some SDK platforms, the ReadyToRun compiler is capable of cross-compiling for other target platforms.

Supported compilation targets are described in the table below when targeting .NET 6 and later versions.

SDK PLATFORM	SUPPORTED TARGET PLATFORMS
Windows X64	Windows (X86, X64, Arm64), Linux (X64, Arm32, Arm64), macOS (X64, Arm64)
Windows X86	Windows (X86), Linux (Arm32)
Linux X64	Linux (X64, Arm32, Arm64), macOS (X64, Arm64)
Linux Arm32	Linux Arm32
Linux Arm64	Linux (X64, Arm32, Arm64), macOS (X64, Arm64)
macOS X64	Linux (X64, Arm32, Arm64), macOS (X64, Arm64)
macOS Arm64	Linux (X64, Arm32, Arm64), macOS (X64, Arm64)

Supported compilation targets are described in the table below when targeting .NET 5 and below.

SDK PLATFORM	SUPPORTED TARGET PLATFORMS
Windows X64	Windows X86, Windows X64, Windows Arm64
Windows X86	Windows X86, Windows Arm32
Linux X64	Linux X86, Linux X64, Linux Arm32, Linux Arm64
Linux Arm32	Linux Arm32
Linux Arm64	Linux Arm64
macOS X64	macOS X64

Trim self-contained deployments and executables

9/20/2022 • 3 minutes to read • [Edit Online](#)

The [framework-dependent deployment model](#) has been the most successful deployment model since the inception of .NET. In this scenario, the application developer bundles only the application and third-party assemblies with the expectation that the .NET runtime and runtime libraries will be available in the client machine. This deployment model continues to be the dominant one in the latest .NET release, however, there are some scenarios where the framework-dependent model is not the best choice. The alternative is to publish a [self-contained application](#), where the .NET runtime and runtime libraries are bundled together with the application and third-party assemblies.

The trim-self-contained deployment model is a specialized version of the self-contained deployment model that is optimized to reduce deployment size. Minimizing deployment size is a critical requirement for some client-side scenarios like Blazor applications. Depending on the complexity of the application, only a subset of the framework assemblies are referenced, and a subset of the code within each assembly is required to run the application. The unused parts of the libraries are unnecessary and can be trimmed from the packaged application.

However, there is a risk that the build-time analysis of the application can cause failures at run time, due to not being able to reliably analyze various problematic code patterns (largely centered on reflection use). To mitigate these problems, warnings are produced whenever the trimmer cannot fully analyze a code pattern. For information on what the trim warnings mean and how to resolve them, see [Introduction to trim warnings](#).

NOTE

- Trimming is fully supported in .NET 6 and later versions. In .NET Core 3.1 and .NET 5, trimming was an experimental feature.
- Trimming is *only* available to applications that are published self-contained.

Components that cause trimming problems

WARNING

Not all project types can be trimmed. For more information, see [Known trimming incompatibilities](#).

Any code that causes build time analysis challenges isn't suitable for trimming. Some common coding patterns that are problematic when used by an application originate from unbounded reflection usage and external dependencies that aren't visible at build time. An example of unbounded reflection is a legacy serializer, such as [XML serialization](#), and an example of invisible external dependencies is [built-in COM](#). To address trim warnings in your application, see [Introduction to trim warnings](#), and to make your library compatible with trimming, see [Prepare .NET libraries for trimming](#).

Enable trimming

- Add `<PublishTrimmed>true</PublishTrimmed>` to your project file.

This property will produce a trimmed app on self-contained publish. It also turns off trim-incompatible features and shows trim compatibility warnings during build.

```
<PropertyGroup>
  <PublishTrimmed>true</PublishTrimmed>
</PropertyGroup>
```

2. Then publish your app using either the [dotnet publish](#) command or Visual Studio.

Publish with the CLI

The following example publishes the app for Windows as a trimmed self-contained application.

```
dotnet publish -r win-x64
```

Trimming is only supported for self-contained apps.

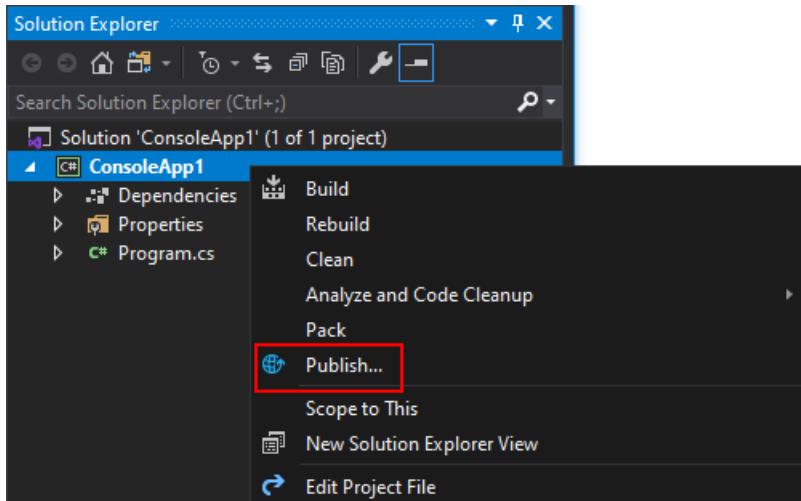
`<PublishTrimmed>` should be set in the project file so that trim-incompatible features are disabled during `dotnet build`. However, you can also set this option as an argument to `dotnet publish`:

```
dotnet publish -r win-x64 -p:PublishTrimmed=true
```

For more information, see [Publish .NET apps with .NET CLI](#).

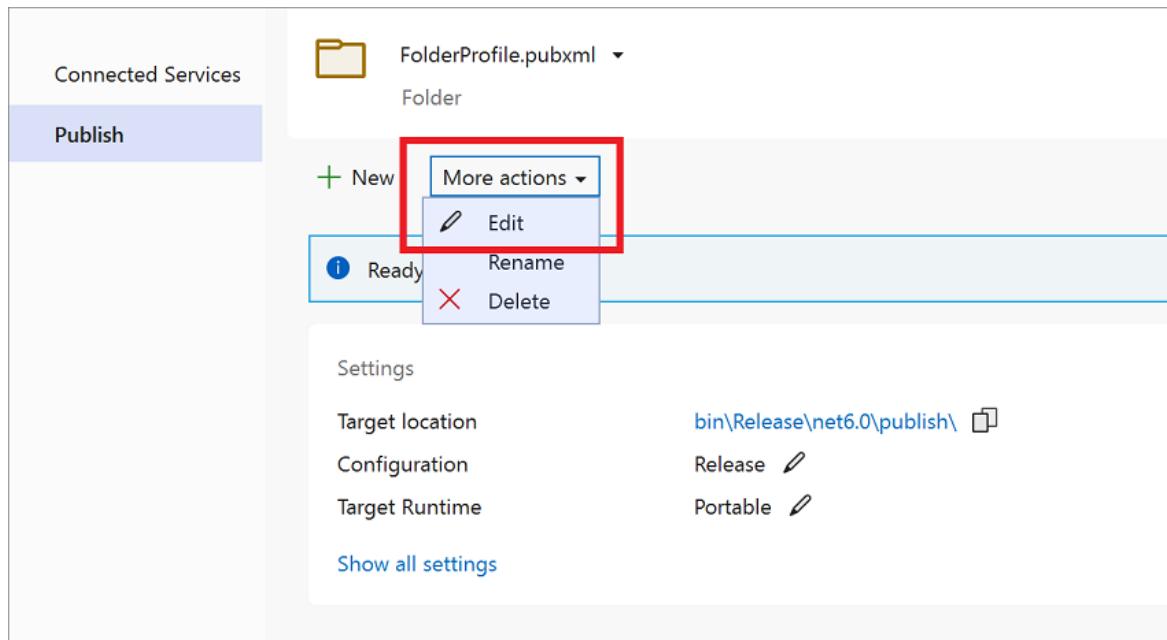
Publish with Visual Studio

1. In Solution Explorer, right-click on the project you want to publish and select **Publish**.



If you don't already have a publishing profile, follow the instructions to create one and choose the **Folder** target-type.

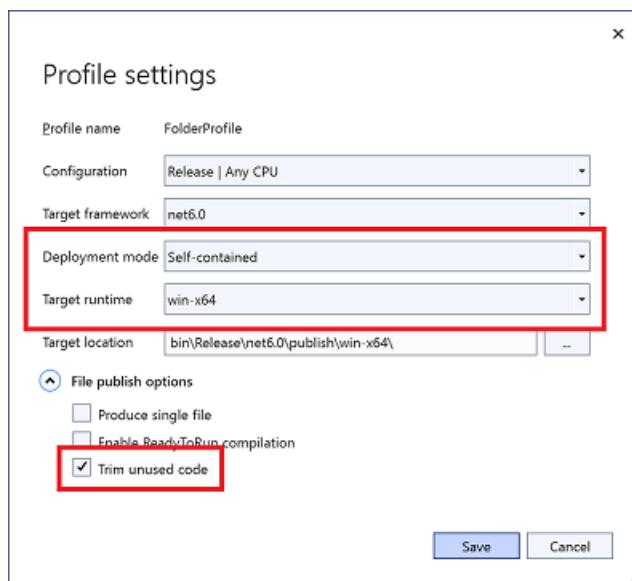
2. Choose **More actions > Edit**.



3. In the **Profile settings** dialog, set the following options:

- Set **Deployment mode** to **Self-contained**.
- Set **Target runtime** to the platform you want to publish to.
- Select **Trim unused code**.

Choose **Save** to save the settings and return to the **Publish** dialog.



4. Choose **Publish** to publish your app trimmed.

For more information, see [Publish .NET Core apps with Visual Studio](#).

Publish with Visual Studio for Mac

Visual Studio for Mac doesn't provide options to publish your app. You'll need to publish manually by following the instructions from the [Publishing with the CLI](#) section. For more information, see [Publish .NET apps with .NET CLI](#).

See also

- [.NET Core application deployment](#).
- [Publish .NET apps with .NET CLI](#).
- [Publish .NET Core apps with Visual Studio](#).

- [dotnet publish command.](#)

Introduction to trim warnings

9/20/2022 • 6 minutes to read • [Edit Online](#)

Conceptually, [trimming](#) is simple: when publishing the application, the .NET SDK analyzes the entire application and removes all unused code. However, it can be difficult to determine what is unused, or more precisely, what is used.

To prevent changes in behavior when trimming applications, the .NET SDK provides static analysis of trim compatibility through "trim warnings." Trim warnings are produced by the trimmer when it finds code that may not be compatible with trimming. Code that's not trim-compatible may produce behavioral changes, or even crashes, in an application after it has been trimmed. Ideally, all applications that use trimming should have no trim warnings. If there are any trim warnings, the app should be thoroughly tested after trimming to ensure that there are no behavior changes.

This article will help developers understand why some patterns produce trim warnings, and how these warnings can be addressed.

Examples of trim warnings

For most C# code, it's straightforward to determine what code is used and what code is unused—the trimmer can walk method calls, field and property references, and so on, and determine what code is accessed.

Unfortunately, some features, like reflection, present a significant problem. Consider the following code:

```
string s = Console.ReadLine();
Type type = Type.GetType(s);
foreach (var m in type.GetMethods())
{
    Console.WriteLine(m.Name);
}
```

In this example, [GetType\(\)](#) dynamically requests a type with an unknown name, and then prints the names of all of its methods. Because there's no way to know at publish time what type name is going to be used, there's no way for the trimmer to know which type to preserve in the output. It's likely that this code could have worked before trimming (as long as the input is something known to exist in the target framework), but would probably produce a null reference exception after trimming (due to `Type.GetType` returning null).

In this case, the trimmer issues a warning on the call to `Type.GetType`, indicating that it cannot determine which type is going to be used by the application.

Reacting to trim warnings

Trim warnings are meant to bring predictability to trimming. There are two large categories of warnings that you will likely see:

1. `RequiresUnreferencedCode`
2. `DynamicallyAccessedMembers`

RequiresUnreferencedCode

[RequiresUnreferencedCodeAttribute](#) is simple and broad: it's an attribute that means the member has been annotated incompatible with trimming, meaning that it might use reflection or some other mechanism to access code that may be trimmed away. This attribute is used when code is fundamentally not trim compatible, or the

trim dependency is too complex to explain to the trimmer. This would often be true for methods that use the C# `dynamic` keyword, accessing types from `LoadFrom(String)`, or other runtime code generation technologies. An example would be:

```
[RequiresUnreferencedCode("Use 'MethodFriendlyToTrimming' instead")]
void MethodWithAssemblyLoad() { ... }

void TestMethod()
{
    // IL2026: Using method 'MethodWithAssemblyLoad' which has 'RequiresUnreferencedCodeAttribute'
    // can break functionality when trimming application code. Use 'MethodFriendlyToTrimming' instead.
    MethodWithAssemblyLoad();
}
```

There aren't many workarounds for `RequiresUnreferencedCode`. The best fix is to avoid calling the method at all when trimming and use something else that's trim-compatible. If you're writing a library and it's not in your control whether or not to call the method, you can also add `RequiresUnreferencedCode` to your own method. This will annotate your method as not trim compatible. Adding `RequiresUnreferencedCode` will silence all trimming warnings in the given method, but will produce a warning whenever someone else calls it. For this reason, it is mostly useful to library authors to "bubble up" the warning to a public API.

If you can somehow determine that the call is safe, and all the code that's needed won't be trimmed away, you can also suppress the warning using `UnconditionalSuppressMessageAttribute`. For example:

```
[RequiresUnreferencedCode("Use 'MethodFriendlyToTrimming' instead")]
void MethodWithAssemblyLoad() { ... }

[UnconditionalSuppressMessage("AssemblyLoadTrimming", "IL2026:RequiresUnreferencedCode",
    Justification = "Everything referenced in the loaded assembly is manually preserved, so it's safe")]
void TestMethod()
{
    MethodWithAssemblyLoad(); // Warning suppressed
}
```

`UnconditionalSuppressMessage` is like `SuppressMessage` but it can be seen by `publish` and other post-build tools. `SuppressMessage` and `#pragma` directives are only present in source so they can't be used to silence warnings from the trimmer. Be very careful when suppressing trim warnings: it's possible that the call may be trim-compatible now, but as you change your code that may change, and you may forget to review all the suppressions.

DynamicallyAccessedMembers

`DynamicallyAccessedMembersAttribute` is usually about reflection. Unlike `RequiresUnreferencedCode`, reflection can sometimes be understood by the trimmer as long as it's annotated correctly. Let's take another look at the original example:

```
string s = Console.ReadLine();
Type type = Type.GetType(s);
foreach (var m in type.GetMethods())
{
    Console.WriteLine(m.Name);
}
```

In the example above, the real problem is `Console.ReadLine()`. Because *any* type could be read, the trimmer has no way to know if you need methods on `System.DateTime` or `System.Guid` or any other type. On the other hand, the following code would be fine:

```

Type type = typeof(System.DateTime);
foreach (var m in type.GetMethods())
{
    Console.WriteLine(m.Name);
}

```

Here the trimmer can see the exact type being referenced: `System.DateTime`. Now it can use flow analysis to determine that it needs to keep all public methods on `System.DateTime`. So where does

`DynamicallyAccessMembers` come in? When reflection is split across multiple methods. In the below code, we can see that the type `System.DateTime` flows to `Method3` where reflection is used to access `System.DateTime`'s methods,

```

void Method1()
{
    Method2<System.DateTime>();
}
void Method2<T>()
{
    Type t = typeof(T);
    Method3(t);
}
void Method3(Type type)
{
    var methods = type.GetMethods();
    ...
}

```

If you compile the previous code, now you see a warning:

Trim analysis warning IL2070: net6.Program.Method3(Type): 'this' argument does not satisfy 'DynamicallyAccessedMemberTypes.PublicMethods' in call to 'System.Type.GetMethods()'. The parameter 'type' of method 'net6.Program.Method3(Type)' does not have matching annotations. The source value must declare at least the same requirements as those declared on the target location it is assigned to.

For performance and stability, flow analysis isn't performed between methods, so an annotation is needed to pass information between methods, from the reflection call (`GetMethods`) to the source of the `Type`. In the previous example, the trimmer warning is saying that `GetMethods` requires the `Type` object instance it is called on to have the `PublicMethods` annotation, but the `type` variable doesn't have the same requirement. In other words, we need to pass the requirements from `GetMethods` up to the caller:

```

void Method1()
{
    Method2<System.DateTime>();
}
void Method2<T>()
{
    Type t = typeof(T);
    Method3(t);
}
void Method3(
    [DynamicallyAccessedMembers(DynamicallyAccessedMemberTypes.PublicMethods)] Type type)
{
    var methods = type.GetMethods();
    ...
}

```

After annotating the parameter `type`, the original warning disappears, but another appears:

IL2087: 'type' argument does not satisfy 'DynamicallyAccessedMemberTypes.PublicMethods' in call to 'C.Method3(Type)'. The generic parameter 'T' of 'C.Method2<T>()' does not have matching annotations.

We propagated annotations up to the parameter `type` of `Method3`, in `Method2` we have a similar issue. The trimmer is able to track the value `T` as it flows through the call to `typeof`, is assigned to the local variable `t`, and passed to `Method3`. At that point it sees that the parameter `type` requires `PublicMethods` but there are no requirements on `T`, and produces a new warning. To fix this, we must "annotate and propagate" by applying annotations all the way up the call chain until we reach a statically known type (like `System.DateTime` or `System.Tuple`), or another annotated value. In this case, we need to annotate the type parameter `T` of `Method2`.

```
void Method1()
{
    Method2<System.DateTime>();
}
void Method2<[DynamicallyAccessedMembers(DynamicallyAccessedMemberTypes.PublicMethods)] T>()
{
    Type t = typeof(T);
    Method3(t);
}
void Method3(
    [DynamicallyAccessedMembers(DynamicallyAccessedMemberTypes.PublicMethods)] Type type)
{
    var methods = type.GetMethods();
    ...
}
```

Now there are no warnings because the trimmer knows exactly which members may be accessed via runtime reflection (public methods) and on which types (`System.DateTime`), and it will preserve them. In general, this is the best way to deal with `DynamicallyAccessedMembers` warnings: add annotations so the trimmer knows what to preserve.

As with `RequiresUnreferencedCode` warnings, adding `RequiresUnreferencedCode` or `UnconditionalSuppressMessage` attributes also suppresses warnings but doesn't make the code compatible with trimming, while adding `DynamicallyAccessedMembers` does make it compatible.

Known trimming incompatibilities

9/20/2022 • 2 minutes to read • [Edit Online](#)

There are some patterns that are known to be incompatible with trimming. Some of these patterns may become compatible as tooling improves or as libraries make modifications to become trimming compatible.

Built-in COM marshalling

Alternative: [COM Wrappers](#)

Automatic [COM marshalling](#) has been built in to .NET since .NET Framework 1.0. It uses run-time code analysis to automatically convert between native COM objects and managed .NET objects. Unfortunately, trimming analysis cannot always predict what .NET code will need to be preserved for automatic COM marshalling. However, if [COM Wrappers](#) are used instead, trimming analysis can guarantee that all used code will be correctly preserved.

WPF

The Windows Presentation Foundation (WPF) framework makes substantial use of reflection and some features are heavily reliant on run-time code inspection. In .NET 6, it's not possible for trimming analysis to preserve all necessary code for WPF applications. Unfortunately, almost no WPF apps are runnable after trimming, so trimming support for WPF has been disabled in the .NET 6 SDK.

Windows Forms

The Windows Forms framework makes minimal use of reflection, but is heavily reliant on built-in COM marshalling. In .NET 6, it has not yet been converted to use ComWrappers. Unfortunately, almost no Windows Forms apps are runnable without built-in COM marshalling, so trimming support for Windows Forms apps has been disabled in the .NET 6 SDK.

Reflection-based serializers

Alternative: Reflection-free serializers, like source-generated [System.Text.Json](#).

Many uses of reflection can be made trimming-compatible, as described in [Introduction to trim warnings](#). However, serializers tend to have very complex uses of reflection. Many of these uses cannot be made analyzable at build time. Unfortunately, the best option is often to rewrite the system to use source generation instead.

Dynamic assembly loading and execution

Trimming and dynamic assembly loading is a common problem for systems that support plugins or extensions, usually through APIs like [LoadFrom\(String\)](#). Trimming relies on seeing all assemblies at build time, so it knows which code is used and cannot be trimmed away. Most plugin systems load third-party code dynamically, so it's not possible for the trimmer to identify what code is needed.

Trimming options

9/20/2022 • 6 minutes to read • [Edit Online](#)

The following MSBuild properties and items influence the behavior of [trimmed self-contained deployments](#). Some of the options mention `ILLink`, which is the name of the underlying tool that implements trimming. For more information about the underlying tool, see the [Trimmer documentation](#).

Trimming with `PublishTrimmed` was introduced in .NET Core 3.0. The other options are available only in .NET 5 and later versions.

Enable trimming

- `<PublishTrimmed>true</PublishTrimmed>`

Enable trimming during publish. This also turns off trim-incompatible features and enables [trim analysis](#) during build.

Place this setting in the project file to ensure that the setting applies during `dotnet build`, not just `dotnet publish`.

This setting enables trimming and will trim all assemblies by default. In .NET 6, only assemblies that opted-in to trimming via `[AssemblyMetadata("IsTrimable", "True")]` would be trimmed by default. You can return to the previous behavior by using `<TrimMode>partial</TrimMode>`.

This setting trims any assemblies that have been configured for trimming. With `Microsoft.NET.Sdk` in .NET 6, this includes any assemblies with `[AssemblyMetadata("IsTrimable", "True")]`, which is the case for the .NET runtime assemblies. In .NET 5, assemblies from the netcoreapp runtime pack are configured for trimming via `<IsTrimable>` MSBuild metadata. Other SDKs may define different defaults.

This setting also enables the trim-compatibility [Roslyn analyzer](#) and disables features that are incompatible with [trimming](#).

Trimming granularity

The default is to trim all assemblies in the app. This can be changed using the `TrimMode` property.

To only trim assemblies that have opted-in to trimming, set the property to `partial`:

```
<TrimMode>partial</TrimMode>
```

The default setting is `full`:

```
<TrimMode>full</TrimMode>
```

The following granularity settings control how aggressively unused IL is discarded. This can be set as a property affecting all trimmer input assemblies, or as metadata on an [individual assembly](#), which overrides the property setting.

- `<TrimMode>link</TrimMode>`

Enable member-level trimming, which removes unused members from types. This is the default in .NET

6+.

- <TrimMode>copyused</TrimMode>

Enable assembly-level trimming, which keeps an entire assembly if any part of it is used (in a statically understood way).

Assemblies with <IsTrimmable>true</IsTrimmable> metadata but no explicit TrimMode will use the global TrimMode. The default TrimMode for Microsoft.NET.Sdk is link in .NET 6+, and copyused in previous versions.

Trim additional assemblies

In .NET 6+, PublishTrimmed trims assemblies with the following assembly-level attribute:

```
[AssemblyMetadata("IsTrimmable", "True")]
```

The framework libraries have this attribute. In .NET 6+, you can also opt in to trimming for a library without this attribute, specifying the assembly by name (without the .dll extension).

In .NET 7, <TrimMode>full</TrimMode> is the default, but if you change the trim mode to partial, you can opt-in individual assemblies to trimming.

```
<ItemGroup>
  <TrimmableAssembly Include="MyAssembly" />
</ItemGroup>
```

This is equivalent to setting [AssemblyMetadata("IsTrimmable", "True")] when building the assembly.

Trimming settings for individual assemblies

When publishing a trimmed app, the SDK computes an ItemGroup called ManagedAssemblyToLink that represents the set of files to be processed for trimming. ManagedAssemblyToLink may have metadata that controls the trimming behavior per assembly. To set this metadata, create a target that runs before the built-in PrepareForILLink target. The following example shows how to enable trimming of MyAssembly.

```
<Target Name="ConfigureTrimming"
      BeforeTargets="PrepareForILLink">
  <ItemGroup>
    <ManagedAssemblyToLink Condition="'%(Filename)' == 'MyAssembly'">
      <IsTrimmable>true</IsTrimmable>
    </ManagedAssemblyToLink>
  </ItemGroup>
</Target>
```

You can also use this to override the trimming behavior specified by the library author, by setting <IsTrimmable>false</IsTrimmable> for an assembly with [AssemblyMetadata("IsTrimmable", "True")].

Do not add or remove items to/from ManagedAssemblyToLink, because the SDK computes this set during publish and expects it not to change. The supported metadata is:

- <IsTrimmable>true</IsTrimmable>

Control whether the given assembly is trimmed.

- <TrimMode>copyused</TrimMode> Or <TrimMode>link</TrimMode>

Control the trimming granularity of this assembly. This takes precedence over the global TrimMode.

Setting `TrimMode` on an assembly implies `<IsTrimmable>true</IsTrimmable>`.

- `<TrimmerSingleWarn>True</TrimmerSingleWarn>` or `<TrimmerSingleWarn>False</TrimmerSingleWarn>`

Control whether to show [single warnings](#) for this assembly.

Root assemblies

If an assembly is not trimmed it is considered "rooted", which means that it and all of its statically understood dependencies will be kept. Additional assemblies may be "rooted" by name (without the `.dll` extension):

```
<ItemGroup>
  <TrimmerRootAssembly Include="MyAssembly" />
</ItemGroup>
```

Root descriptors

Another way to specify roots for analysis is using an XML file that uses the trimmer [descriptor format](#). This lets you root specific members instead of a whole assembly.

```
<ItemGroup>
  <TrimmerRootDescriptor Include="MyRoots.xml" />
</ItemGroup>
```

For example, `MyRoots.xml` might root a specific method that is dynamically accessed by the application:

```
<linker>
  <assembly fullname="MyAssembly">
    <type fullname="MyAssembly.MyClass">
      <method name="DynamicallyAccessedMethod" />
    </type>
  </assembly>
</linker>
```

Analysis warnings

- `<SuppressTrimAnalysisWarnings>false</SuppressTrimAnalysisWarnings>`

Enable trim analysis warnings.

Trimming removes IL that is not statically reachable. Apps that use reflection or other patterns that create dynamic dependencies may be broken by trimming. To warn about such patterns, set

`<SuppressTrimAnalysisWarnings>` to `false`. This will include warnings about the entire app, including your own code, library code, and framework code.

Roslyn analyzer

Setting `PublishTrimmed` in .NET 6+ also enables a Roslyn analyzer that shows a *limited* set of analysis warnings. You can also enable or disable the analyzer independently of `PublishTrimmed`.

- `<EnableTrimAnalyzer>true</EnableTrimAnalyzer>`

Enable a Roslyn analyzer for a subset of trim analysis warnings.

Suppress warnings

You can suppress individual [warning codes](#) using the usual MSBuild properties respected by the toolchain, including `NoWarn`, `WarningsAsErrors`, `WarningsNotAsErrors`, and `TreatWarningsAsErrors`. There is an additional option that controls the ILLink warn-as-error behavior independently:

- `<ILLinkTreatWarningsAsErrors>false</ILLinkTreatWarningsAsErrors>`

Don't treat ILLink warnings as errors. This may be useful to avoid turning trim analysis warnings into errors when treating compiler warnings as errors globally.

Show detailed warnings

In .NET 6+, trim analysis produces at most one warning for each assembly that comes from a `PackageReference`, indicating that the assembly's internals are not compatible with trimming. You can also show individual warnings for all assemblies:

- `<TrimmerSingleWarn>false</TrimmerSingleWarn>`

Show all detailed warnings, instead of collapsing them to a single warning per assembly.

Remove symbols

Symbols are usually trimmed to match the trimmed assemblies. You can also remove all symbols:

- `<TrimmerRemoveSymbols>true</TrimmerRemoveSymbols>`

Remove symbols from the trimmed application, including embedded PDBs and separate PDB files. This applies to both the application code and any dependencies that come with symbols.

The SDK also makes it possible to disable debugger support using the property `DebuggerSupport`. When debugger support is disabled, trimming will remove symbols automatically (`TrimmerRemoveSymbols` will default to true).

Trimming framework library features

Several feature areas of the framework libraries come with trimmer directives that make it possible to remove the code for disabled features.

- `<AutoreleasePoolSupport>false</AutoreleasePoolSupport>` (default)

Remove code that creates autorelease pools on supported platforms. See [AutoreleasePool for managed threads](#). This is the default for the .NET SDK.

- `<DebuggerSupport>false</DebuggerSupport>`

Remove code that enables better debugging experiences. This setting also [removes symbols](#).

- `<EnableUnsafeBinaryFormatterSerialization>false</EnableUnsafeBinaryFormatterSerialization>`

Remove BinaryFormatter serialization support. For more information, see [BinaryFormatter serialization methods are obsolete](#).

- `<EnableUnsafeUTF7Encoding>false</EnableUnsafeUTF7Encoding>`

Remove insecure UTF-7 encoding code. For more information, see [UTF-7 code paths are obsolete](#).

- `<EventSourceSupport>false</EventSourceSupport>`

Remove EventSource related code or logic.

- `<HttpActivityPropagationSupport>false</HttpActivityPropagationSupport>`

Remove code related to diagnostics support for System.Net.Http.

- `<InvariantGlobalization>true</InvariantGlobalization>`
Remove globalization-specific code and data. For more information, see [Invariant mode](#).
- `<MetadataUpdaterSupport>false</MetadataUpdaterSupport>`
Remove metadata update-specific logic related to hot reload.
- `<UseNativeHttpHandler>true</UseNativeHttpHandler>`
Use the default platform implementation of `HttpMessageHandler` for Android/iOS and remove the managed implementation.
- `<UseSystemResourceKeys>true</UseSystemResourceKeys>`
Strip exception messages for `System.*` assemblies. When an exception is thrown from a `System.*` assembly, the message will be a simplified resource ID instead of the full message.

These properties cause the related code to be trimmed and also disable features via the `runtimeconfig` file. For more information about these properties, including the corresponding `runtimeconfig` options, see [feature switches](#). Some SDKs may have default values for these properties.

Framework features disabled when trimming

The following features are incompatible with trimming because they require code that is not statically referenced. These are disabled by default in trimmed apps.

WARNING

Enable these features at your own risk. They are likely to break trimmed apps without extra work to preserve the dynamically referenced code.

- `<BuiltInComInteropSupport>`
Built-in COM support is disabled.
- `<CustomResourceTypesSupport>`
Use of custom resource types is not supported. `ResourceManager` code paths that use reflection for custom resource types is trimmed.
- `<EnableCppCLIHostActivation>`
`C++/CLI` host activation is disabled.
- `<EnableUnsafeBinaryFormatterInDesigntimeLicenseContextSerialization>`
`DesigntimeLicenseContextSerializer` use of `BinaryFormatter` serialization is disabled.
- `<StartupHookSupport>`
Running code before `Main` with `DOTNET_STARTUP_HOOKS` is not supported. For more information, see [host startup hook](#).

Prepare .NET libraries for trimming

9/20/2022 • 12 minutes to read • [Edit Online](#)

The .NET SDK makes it possible to reduce the size of self-contained apps by [trimming](#), which removes unused code from the app and its dependencies. Not all code is compatible with trimming, so .NET 6 provides trim analysis warnings to detect patterns that may break trimmed apps. To resolve warnings originating from the app code, see [resolving trim warnings](#). This article describes how to prepare libraries for trimming with the aid of these warnings, including recommendations for fixing some common cases.

Enable library trim warnings

TIP

Ensure you're using the .NET 6 SDK or later for these steps. They will not work correctly in previous versions.

There are two ways to find trim warnings in your library:

1. Enable project-specific trimming using the `IsTrimmable` property.
2. Add your library as a reference to a sample app, and trim the sample app.

Consider doing both. Project-specific trimming is convenient and shows trim warnings for one project, but relies on the references being marked trim-compatible in order to see all warnings. Trimming a sample app is more work, but will always show all warnings.

Enable project-specific trimming

TIP

To use the latest version of the analyzer with the most coverage, consider using the [.NET 7 preview SDK](#). Note this will only update the tooling used to build your app, this does not require you to target the .NET 7 runtime.

Set `<IsTrimmable>true</IsTrimmable>` in a `<PropertyGroup>` tag in your library project file. This will mark your assembly as "trimmable" and enable trim warnings for that project. Being "trimmable" means your library is considered compatible with trimming and should have no trim warnings when building the library. When used in a trimmed application, the assembly will have its unused members trimmed in the final output.

If you want to see trim warnings, but don't want to mark your library as trim-compatible, you can add `<EnableTrimAnalyzer>true</EnableTrimAnalyzer>` instead.

Show all warnings with sample application

To show all analysis warnings for your library, including warnings about dependencies, you need the trimmer to analyze the implementation of your library and the implementations of dependencies your library uses. When building and publishing a library, the implementations of the dependencies are not available, and the reference assemblies that are available do not have enough information for the trimmer to determine if they are compatible with trimming. Because of this, you'll need to create and publish a self-contained sample application which produces an executable that includes your library and the dependencies it relies on. This executable includes all the information the trimmer requires to warn you about all trim incompatibilities in your library code, as well as the code that your library references from its dependencies.

To create your sample app, first create a separate application project with `dotnet new` and modify the project file

to look like the following. No changes to the source code are necessary. You'll need to add the following to your project file:

- Set the PublishTrimmed property to `true` with `<PublishTrimmed>true</PublishTrimmed>` in a `<PropertyGroup>` tag.
- A reference to your library with `<PublishTrimmed>true</PublishTrimmed>` in an `<ItemGroup>` tag.
- Specify your library as a trimmer root assembly with `<TrimmerRootAssembly Include="YourLibraryName" />` in an `<ItemGroup>` tag.
 - This ensures that every part of the library is analyzed. It tells the trimmer that this assembly is a "root" which means the trimmer will analyze the assembly as if everything will be used, and traverses all possible code paths that originate from that assembly. This is necessary in case the library has `[AssemblyMetadata("IsTrimmable", "True")]`, which would otherwise let trimming remove the unused library without analyzing it.
- Set the TrimmerDefaultAction property to `trim` with `<TrimmerDefaultAction>link</TrimmerDefaultAction>` in a `<PropertyGroup>` tag.
 - This ensures that the trimmer only analyzes the parts of the library's dependencies that are used. It tells the trimmer that any code that is not part of a "root" can be trimmed if it is unused. Without this option, you would see warnings originating from *any* part of a dependency that doesn't set `[AssemblyMetadata("IsTrimmable", "True")]`, including parts that are unused by your library.

```
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>net6.0</TargetFramework>
    <PublishTrimmed>true</PublishTrimmed>
    <!-- Prevent warnings from unused code in dependencies -->
    <TrimmerDefaultAction>link</TrimmerDefaultAction>
  </PropertyGroup>

  <ItemGroup>
    <ProjectReference Include="path/to/MyLibrary.csproj" />
    <!-- Analyze the whole library, even if attributed with "IsTrimmable" -->
    <TrimmerRootAssembly Include="MyLibrary" />
  </ItemGroup>

</Project>
```

Once your project file is updated, run `dotnet publish` with the [runtime identifier \(RID\)](#) you want to target.

```
dotnet publish -c Release -r <RID>
```

You can also follow the same pattern for multiple libraries. To see trim analysis warnings for more than one library at a time, add them all to the same project as `ProjectReference` and `TrimmerRootAssembly` items. This will warn about dependencies if *any* of the root libraries use a trim-unfriendly API in a dependency. To see warnings that have to do with only a particular library, reference that library only.

NOTE

The analysis results depend on the implementation details of your dependencies. If you update to a new version of a dependency, this may introduce analysis warnings if the new version added non-understood reflection patterns, even if there were no API changes. In other words, introducing trim analysis warnings to a library is a breaking change when the library is used with `PublishTrimmed`.

Resolve trim warnings

The above steps will produce warnings about code that may cause problems when used in a trimmed app. Here are a few examples of the most common kinds of warnings you may encounter, with recommendations for fixing them.

RequiresUnreferencedCode

```
using System.Diagnostics.CodeAnalysis;

public class MyLibrary
{
    public static void Method()
    {
        // warning IL2026 : MyLibrary.Method: Using method 'MyLibrary.DynamicBehavior' which has
        // 'RequiresUnreferencedCodeAttribute' can break functionality
        // when trimming application code.
        DynamicBehavior();
    }

    [RequiresUnreferencedCode("DynamicBehavior is incompatible with trimming.")]
    static void DynamicBehavior()
    {
    }
}
```

This means the library calls a method that has explicitly been annotated as incompatible with trimming, using [RequiresUnreferencedCodeAttribute](#). To get rid of the warning, consider whether `Method` needs to call `DynamicBehavior` to do its job. If so, annotate the caller `Method` with `RequiresUnreferencedCode` as well; this will "bubble up" the warning so that callers of `Method` get a warning instead:

```
// Warn for calls to Method, but not for Method's call to DynamicBehavior.
[RequiresUnreferencedCode("Calls DynamicBehavior.")]
public static void Method()
{
    DynamicBehavior(); // OK. Doesn't warn now.
}
```

Once you have "bubbled up" the attribute all the way to public APIs (so that these warnings are produced only for public methods, if at all), you are done. Apps that call your library will now get warnings if they call those public APIs, but these will no longer produce warnings like

`IL2104: Assembly 'MyLibrary' produced trim warnings.`

DynamicallyAccessedMembers

```
using System.Diagnostics.CodeAnalysis;

public class MyLibrary
{
    static void UseMethods(Type type)
    {
        // warning IL2070: MyLibrary.UseMethods(Type): 'this' argument does not satisfy
        // 'DynamicallyAccessedMemberTypes.PublicMethods' in call to 'System.Type.GetMethods()'.
        // The parameter 't' of method 'MyLibrary.UseMethods(Type)' does not have matching annotations.
        foreach (var method in type.GetMethods())
        {
            // ...
        }
    }
}
```

Here, `UseMethods` is calling a reflection method that has a `DynamicallyAccessedMembersAttribute` requirement. The requirement states that the type's public methods are available. In this case, you can satisfy the requirement by adding the same requirement to the parameter of `UseMethods`.

```
static void UseMethods(
    // State the requirement in the UseMethods parameter.
    [DynamicallyAccessedMembers(DynamicallyAccessedMemberTypes.PublicMethods)]
    Type type)
{
    // ...
}
```

Now any calls to `UseMethods` will produce warnings if they pass in values that don't satisfy the `PublicMethods` requirement. Like with `RequiresUnreferencedCode`, once you have bubbled up such warnings to public APIs, you are done.

Here is another example where an unknown `Type` flows into the annotated method parameter, this time from a field:

```
static Type type;

static void UseMethodsHelper()
{
    // warning IL2077: MyLibrary.UseMethodsHelper(Type): 'type' argument does not satisfy
    // 'DynamicallyAccessedMemberTypes.PublicMethods' in call to 'MyLibrary.UseMethods(Type)'.
    // The field 'System.Type MyLibrary::type' does not have matching annotations.
    UseMethods(type);
}
```

Similarly, here the problem is that the field `type` is passed into a parameter with these requirements. You can fix it by adding `DynamicallyAccessedMembers` to the field. This will warn about code that assigns incompatible values to the field instead. Sometimes this process will continue until a public API is annotated, and other times it will end when a concrete type flows into a location with these requirements. For example:

```
[DynamicallyAccessedMembers(DynamicallyAccessedMemberTypes.PublicMethods)]
static Type type;

static void InitializeTypeField()
{
    MyLibrary.type = typeof(System.Tuple);
}
```

In this case the trim analysis will simply keep public methods of `System.Tuple`, and will not produce further warnings.

Recommendations

In general, try to avoid reflection if possible. When using reflection, limit it in scope so that it is reachable only from a small part of the library.

- Avoid using non-understood patterns in places like static constructors that will result in the warning propagating to all members of the class.
- Avoid annotating virtual methods or interface methods, which will require all overrides to have matching annotations.
- In some cases, you will be able to mechanically propagate warnings through your code without issues. Sometimes this will result in much of your public API being annotated with `RequiresUnreferencedCode`, which

is the right thing to do if the library indeed behaves in ways that can't be understood statically by the trim analysis.

- In other cases, you might discover that your code uses patterns that can't be expressed in terms of the `DynamicallyAccessedMembers` attributes, even if it only uses reflection to operate on statically known types. In these cases, you may need to reorganize some of your code to make it follow an analyzable pattern.
- Sometimes the existing design of an API will render it mostly trim-incompatible, and you may need to find other ways to accomplish what it is doing. A common example is reflection-based serializers. In these cases, consider adopting other technology like source generators to produce code that is more easily statically analyzed.

Resolve warnings for non-analyzable patterns

It's better to resolve warnings by expressing the intent of your code using `RequiresUnreferencedCode` and `DynamicallyAccessedMembers` when possible. However, in some cases you may be interested in enabling trimming of a library that uses patterns that can't be expressed with those attributes, or without refactoring existing code. This section describes some advanced ways to resolve trim analysis warnings.

WARNING

These techniques might break your code if used incorrectly.

UnconditionalSuppressMessage

If the intent of your code can't be expressed with the annotations, but you know that the warning doesn't represent a real issue at run time, you can suppress the warnings using

[UnconditionalSuppressMessageAttribute](#). This is similar to `SuppressMessageAttribute`, but it's persisted in IL and respected during trim analysis.

WARNING

When suppressing warnings, you are responsible for guaranteeing the trim compatibility of your code based on invariants that you know to be true by inspection. Be careful with these annotations, because if they are incorrect, or if invariants of your code change, they might end up hiding real issues.

For example:

```

class TypeCollection
{
    Type[] types;

    // Ensure that only types with preserved constructors are stored in the array
    [DynamicallyAccessedMembers(DynamicallyAccessedMemberTypes.PublicParameterlessConstructor)]
    public Type this[int i]
    {
        // warning IL2063: TypeCollection.Item.get: Value returned from method 'TypeCollection.Item.get'
        // can not be statically determined and may not meet 'DynamicallyAccessedMembersAttribute'
        requirements.
        get => types[i];
        set => types[i] = value;
    }
}

class TypeCreator
{
    TypeCollection types;

    public void CreateType(int i)
    {
        types[i] = typeof(TypeWithConstructor);
        Activator.CreateInstance(types[i]); // No warning!
    }
}

class TypeWithConstructor
{
}

```

Here, the indexer property has been annotated so that the returned `Type` meets the requirements of `CreateInstance`. This already ensures that the `TypeWithConstructor` constructor is kept, and that the call to `CreateInstance` doesn't warn. Furthermore, the indexer setter annotation ensures that any types stored in the `Type[]` have a constructor. However, the analysis isn't able to see this, and still produces a warning for the getter, because it doesn't know that the returned type has its constructor preserved.

If you are sure that the requirements are met, you can silence this warning by adding `UnconditionalSuppressMessage` to the getter:

```

[DynamicallyAccessedMembers(DynamicallyAccessedMemberTypes.PublicParameterlessConstructor)]
public Type this[int i]
{
    [UnconditionalSuppressMessage("ReflectionAnalysis", "IL2063",
        Justification = "The list only contains types stored through the annotated setter.")]
    get => types[i];
    set => types[i] = value;
}

```

It is important to underline that it is only valid to suppress a warning if there are annotations or code that ensure the reflected-on members are visible targets of reflection. It is not sufficient that the member was simply a target of a call, field or property access. It may appear to be the case sometimes but such code is bound to break eventually as more trimming optimizations are added. Properties, fields, and methods that are not visible targets of reflection could be inlined, have their names removed, get moved to different types, or otherwise optimized in ways that will break reflecting on them. When suppressing a warning, it's only permissible to reflect on targets that were visible targets of reflection to the trimming analyzer elsewhere.

```
[UnconditionalSuppressMessage("ReflectionAnalysis", "IL2063",
    // Invalid justification and suppression: property being non-reflectively
    // used by the app doesn't guarantee that the property will be available
    // for reflection. Properties that are not visible targets of reflection
    // are already optimized away with Native AOT trimming and may be
    // optimized away for non-native deployment in the future as well.
    Justification = "*INVALID* Only need to serialize properties that are used by the app. *INVALID*")]
public string Serialize(object o)
{
    StringBuilder sb = new StringBuilder();
    foreach (var property in o.GetType().GetProperties())
    {
        AppendProperty(sb, property, o);
    }
    return sb.ToString();
}
```

DynamicDependency

This attribute can be used to indicate that a member has a dynamic dependency on other members. This results in the referenced members being kept whenever the member with the attribute is kept, but doesn't silence warnings on its own. Unlike the other attributes which teach the trim analysis about the reflection behavior of your code, `DynamicDependency` only keeps additional members. This can be used together with `UnconditionalSuppressMessageAttribute` to fix some analysis warnings.

WARNING

Use `DynamicDependencyAttribute` only as a last resort when the other approaches aren't viable. It is preferable to express the reflection behavior of your code using `RequiresUnreferencedCodeAttribute` or `DynamicallyAccessedMembersAttribute`.

```
[DynamicDependency("Helper", "MyType", "MyAssembly")]
static void RunHelper()
{
    var helper = Assembly.Load("MyAssembly").GetType("MyType").GetMethod("Helper");
    helper.Invoke(null, null);
}
```

Without `DynamicDependency`, trimming might remove `Helper` from `MyAssembly` or remove `MyAssembly` completely if it's not referenced elsewhere, producing a warning that indicates a possible failure at run time. The attribute ensures that `Helper` is kept.

The attribute specifies the members to keep via a `string` or via `DynamicallyAccessedMemberTypes`. The type and assembly are either implicit in the attribute context, or explicitly specified in the attribute (by `Type`, or by `string`s for the type and assembly name).

The type and member strings use a variation of the C# documentation comment ID string [format](#), without the member prefix. The member string should not include the name of the declaring type, and may omit parameters to keep all members of the specified name. Some examples of the format follow:

```
[DynamicDependency("Method")]
[DynamicDependency("Method(System.Boolean,System.String)")]
[DynamicDependency("MethodOnDifferentType()", typeof(ContainingType))]
[DynamicDependency("MemberName")]
[DynamicDependency("MemberOnUnreferencedAssembly", "ContainingType", "UnreferencedAssembly")]
[DynamicDependency("MemberName", "Namespace.ContainingType.NestedType", "Assembly")]
// generics
[DynamicDependency("GenericMethodName``1")]
[DynamicDependency("GenericMethod``2(``0,``1)")]
[DynamicDependency("MethodWithGenericParameterTypes(System.Collections.Generic.List{System.String})")]
[DynamicDependency("MethodOnGenericType(`0)", "GenericType`1", "UnreferencedAssembly")]
[DynamicDependency("MethodOnGenericType(`0)", typeof(GenericType<>))]
```

This attribute is designed to be used in cases where a method contains reflection patterns that can not be analyzed even with the help of [DynamicallyAccessedMembersAttribute](#).

IL2001: Descriptor file tried to preserve fields on type that has no fields

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

An XML descriptor file is trying to preserve fields on a type with no fields.

Rule description

[Descriptor files](#) are used to direct the IL trimmer to always keep certain members in an assembly, regardless of whether the trimmer can find references to them. However, trying to preserve members that cannot be found will trigger a warning.

Example

```
<linker>
  <assembly fullname="test">
    <type fullname="TestType" preserve="fields" />
  </assembly>
</linker>
```

```
// IL2001: Type 'TestType' has no fields to preserve
class TestType
{
    void OnlyMethod() {}
}
```

IL2002: Descriptor file tried to preserve methods on type that has no methods

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

An XML descriptor file is trying to preserve methods on a type with no methods.

Rule description

[Descriptor files](#) are used to direct the IL trimmer to always keep certain members in an assembly, regardless of whether the trimmer can find references to them. However, trying to preserve members that cannot be found will trigger a warning.

Example

```
<linker>
  <assembly fullname="test">
    <type fullname="TestType" preserve="methods" />
  </assembly>
</linker>
```

```
// IL2002: Type 'TestType' has no methods to preserve
struct TestType
{
    public int Number;
}
```

IL2003: Could not resolve dependency assembly specified in a 'PreserveDependency' attribute

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

The assembly specified in a `PreserveDependencyAttribute` could not be resolved.

Rule description

Trimmer keeps a cache with the assemblies that it has seen. If the assembly specified in the `PreserveDependencyAttribute` is not found in this cache, the trimmer does not have a way to find the member to preserve.

Example

```
// IL2003: Could not resolve dependency assembly 'NonExistentAssembly' specified in a 'PreserveDependency' attribute
[PreserveDependency("MyMethod", "MyType", "NonExistentAssembly")]
void TestMethod()
{}
```

IL2004: Could not resolve dependency type specified in a 'PreserveDependency' attribute

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

The type specified in a `PreserveDependencyAttribute` could not be resolved.

Rule description

Trimmer keeps a cache with the assemblies that it has seen. If the type specified in the `PreserveDependencyAttribute` is not found inside an assembly in this cache, the trimmer does not have a way to find the member to preserve.

Example

```
// IL2004: Could not resolve dependency type 'NonExistentType' specified in a 'PreserveDependency' attribute
[PreserveDependency("MyMethod", "NonExistentType", "MyAssembly")]
void TestMethod()
{
}
```

IL2005: Could not resolve dependency member specified in a 'PreserveDependency' attribute

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

The member of a type specified in a `PreserveDependencyAttribute` could not be resolved.

Example

```
// IL2005: Could not resolve dependency member 'NonExistentMethod' declared on type 'MyType' specified in a
// 'PreserveDependency' attribute
[PreserveDependency("NonExistentMethod", "MyType", "MyAssembly")]
void TestMethod()
{
}
```

IL2007: Could not resolve assembly specified in descriptor file

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

An assembly specified in a descriptor file could not be resolved.

Rule description

[Descriptor files](#) are used to instruct the trimmer to always keep certain items in an assembly, regardless of whether the trimmer could find any references to them.

The assembly specified in the descriptor file by its full name could not be found in any of the assemblies seen by the trimmer.

Example

```
<!-- IL2007: Could not resolve assembly 'NonExistentAssembly' -->
<linker>
  <assembly fullname="NonExistentAssembly" />
</linker>
```

IL2008: Could not resolve type specified in descriptor file

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

A type specified in a descriptor file could not be resolved.

Rule description

[Descriptor files](#) are used to instruct the trimmer to always keep certain items in an assembly, regardless of whether the trimmer could find any references to them.

A type specified in a descriptor file could not be found in the assembly matching the `fullname` argument that was passed to the parent of the `type` element.

Example

```
<!-- IL2008: Could not resolve type 'NonExistentType' -->
<linker>
  <assembly fullname="MyAssembly">
    <type fullname="NonExistentType" />
  </assembly>
</linker>
```

IL2009: Could not resolve method specified in descriptor file

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

A method specified on a type in a descriptor file could not be resolved.

Rule description

[Descriptor files](#) are used to instruct the trimmer to always keep certain items in an assembly, regardless of whether the trimmer could find any references to them.

A method specified in a descriptor file could not be found in the type matching the `fullname` argument that was passed to the parent of the `method` element.

Example

```
<!-- IL2009: Could not find method 'NonExistentMethod' on type 'MyType' -->
<linker>
  <assembly fullname="MyAssembly">
    <type fullname="MyType">
      <method name="NonExistentMethod" />
    </type>
  </assembly>
</linker>
```

IL2010: Invalid value on a method substitution

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

The value used in a substitution file for replacing a method's body does not represent a value of a built-in type or match the return type of the method.

Rule description

[Substitution files](#) are used to instruct the trimmer to replace specific method bodies with a `throw` statement or to return constant statements.

The value passed to the `value` argument of a `method` element could not be converted by the trimmer to a type matching the return type of the specified method.

Example

```
<!-- IL2010: Invalid value for 'MyType.MyMethodReturningInt()' stub -->
<linker>
  <assembly fullname="MyAssembly">
    <type fullname="MyType">
      <method name="MyMethodReturningInt" body="stub" value="NonNumber" />
    </type>
  </assembly>
</linker>
```

IL2011: Unknown body modification action

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

The action value passed to the `body` argument of a `method` element in a substitution file is invalid.

Rule description

[Substitution files](#) are used to instruct the trimmer to replace specific method bodies with a `throw` statement or to return constant statements.

The value passed to the `body` argument of a `method` element was invalid. The only supported options for this argument are `remove` and `stub`.

Example

```
<!-- IL2011: Unknown body modification 'nonaction' for 'MyType.MyMethod()' -->
<linker>
  <assembly fullname="MyAssembly">
    <type fullname="MyType">
      <method name="MyMethod" body="nonaction" value="NonNumber" />
    </type>
  </assembly>
</linker>
```

IL2012: Could not find field on type in substitution file

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

A field specified for substitution in a substitution file could not be found.

Rule description

[Substitution files](#) are used to instruct the trimmer to replace specific method bodies with either a throw or return constant statements.

A field specified in a substitution file could not be found in the type matching the `fullname` argument that was passed to the parent of the `field` element.

Example

```
<!-- IL2012: Could not find field 'NonExistentField' on type 'MyType' -->
<linker>
  <assembly fullname="MyAssembly">
    <type fullname="MyType">
      <field name="NonExistentField" />
    </type>
  </assembly>
</linker>
```

IL2013: Substituted fields must be static or constant

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

A field specified for substitution in a substitution file is non-static or constant.

Rule description

[Substitution files](#) are used to instruct the trimmer to replace specific method bodies with either a throw or return constant statements.

Trimmer cannot substitute non-static or constant fields.

Example

```
<!-- IL2013: Substituted field 'MyType.InstanceField' needs to be static field -->
<linker>
  <assembly fullname="MyAssembly">
    <type fullname="MyType">
      <field name="InstanceField" value="5" />
    </type>
  </assembly>
</linker>
```

IL2014: Missing value for field substitution

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

A field was specified for substitution in a substitution file but no value to be substituted for was given.

Rule description

[Substitution files](#) are used to instruct the trimmer to replace specific method bodies with either a throw or return constant statements.

A `field` element specified in the substitution file does not specify the required `value` argument.

Example

```
<!-- IL2014: Missing 'value' attribute for field 'MyType.MyField' -->
<linker>
  <assembly fullname="MyAssembly">
    <type fullname="MyType">
      <field name="MyField" />
    </type>
  </assembly>
</linker>
```

IL2015: Invalid value for field substitution

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

The value used in a substitution file for replacing a field's value does not represent a value of a built-in type or does not match the type of the field.

Rule description

[Substitution files](#) are used to instruct the trimmer to replace specific method bodies with either a throw or return constant statements.

The value passed to the `value` argument of a `field` element could not be converted by the trimmer to a type matching the return type of the specified field.

Example

```
<!-- IL2015: Invalid value 'NonNumber' for 'MyType.IntField' -->
<linker>
  <assembly fullname="MyAssembly">
    <type fullname="MyType">
      <field name="IntField" value="NonNumber" />
    </type>
  </assembly>
</linker>
```

IL2016: Could not find event on type

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

Could not find event on type.

Rule description

An event specified in an [xml file for the trimmer](#) file could not be found in the type matching the `fullname` argument that was passed to the parent of the `event` element.

Example

```
<!-- IL2016: Could not find event 'NonExistentEvent' on type 'MyType' -->
<linker>
  <assembly fullname="MyAssembly">
    <type fullname="MyType">
      <event name="NonExistentEvent" />
    </type>
  </assembly>
</linker>
```

IL2017: Could not find property on type

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

Could not find property on type.

Rule description

A property specified in an [xml file for the trimmer](#) file could not be found in the type matching the `fullname` argument that was passed to the parent of the `property` element.

Example

```
<!-- IL2017: Could not find property 'NonExistentProperty' on type 'MyType' -->
<linker>
  <assembly fullname="MyAssembly">
    <type fullname="MyType">
      <property name="NonExistentProperty" />
    </type>
  </assembly>
</linker>
```

IL2018: Could not find the get accessor of property on type in descriptor file

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

A `get` accessor specified in a descriptor file could not be found.

Rule description

[Descriptor files](#) are used to instruct the trimmer to always keep certain items in an assembly, regardless of whether the trimmer could find any references to them.

A `get` accessor specified in a descriptor file could not be found in the property matching the `signature` argument that was passed to the `property` element.

Example

```
<!-- IL2018: Could not find the get accessor of property 'SetOnlyProperty' on type 'MyType' -->
<linker>
  <assembly fullname="MyAssembly">
    <type fullname="MyType">
      <property signature="System.Boolean SetOnlyProperty" accessors="get" />
    </type>
  </assembly>
</linker>
```

IL2019: Could not find the set accessor of property on type in descriptor file

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

A `set` accessor specified in a descriptor file could not be found.

Rule description

[Descriptor files](#) are used to instruct the trimmer to always keep certain items in an assembly, regardless of whether the trimmer could find any references to them.

A `set` accessor specified in a descriptor file could not be found in the property matching the `signature` argument that was passed to the `property` element.

Example

```
<!-- IL2019: Could not find the set accessor of property 'GetOnlyProperty' on type 'MyType' -->
<linker>
  <assembly fullname="MyAssembly">
    <type fullname="MyType">
      <property signature="System.Boolean GetOnlyProperty" accessors="set" />
    </type>
  </assembly>
</linker>
```

IL2022: Could not find matching constructor for custom attribute specified in custom attribute annotations file

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

The constructor of a custom attribute specified in a custom attribute annotations file could not be found.

Rule description

[Custom attribute annotation files](#) are used to instruct the trimmer to behave as if the specified item has a given attribute. Attribute annotations can only be used to add attributes that have an effect on the trimmer behavior; all other attributes are ignored. Attributes added via attribute annotations only influence the trimmer behavior and they are never added to the output assembly.

A value passed to an `argument` child of an `attribute` element could not be converted by the trimmer to a type matching the attribute's constructor argument type.

Example

```
<!-- IL2022: Could not find matching constructor for custom attribute 'attribute-type' arguments -->
<linker>
  <assembly fullname="MyAssembly">
    <type fullname="MyType">
      <attribute fullname="AttributeWithNoParametersAttribute">
        <argument>ExtraArgumentValue</argument>
      </attribute>
    </type>
  </assembly>
</linker>
```

IL2023: There is more than one `return` child element specified for a method in a custom attribute annotations file

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

A method has more than one `return` element specified. There can only be one `return` element when putting an attribute on the return parameter of a method.

Rule description

[Custom attribute annotation files](#) are used to instruct the trimmer to behave as if the specified item has a given attribute. Attribute annotations can only be used to add attributes that have effect on the trimmer behavior. All other attributes are ignored. Attributes added via attribute annotations only influence the trimmer behavior, and they are never added to the output assembly.

A `method` element has more than one `return` element specified. Trimmer only allows one attribute annotation on the return type of a given method.

Example

```
<!-- IL2023: There is more than one 'return' child element specified for method 'method' -->
<linker>
  <assembly fullname="MyAssembly">
    <type fullname="MyType">
      <method name="MyMethod">
        <return>
          <attribute fullname="FirstAttribute"/>
        </return>
        <return>
          <attribute fullname="SecondAttribute"/>
        </return>
      </method>
    </type>
  </assembly>
</linker>
```

IL2024: There is more than one value specified for the same method parameter in a custom attribute annotations file

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

A method parameter has more than one value element specified. There can only be one value specified for each method parameter.

Rule description

[Custom attribute annotation files](#) are used to instruct the trimmer to behave as if the specified item has a given attribute. Attribute annotations can only be used to add attributes which have effect on the trimmer behavior, all other attributes will be ignored. Attributes added via attribute annotations only influence the trimmer behavior and they are never added to the output assembly.

There is more than one `parameter` element with the same `name` value in a given `method`. All attributes on a `parameter` should be put in a single element.

Example

```
<!-- IL2024: More than one value specified for parameter 'parameter' of method 'method' -->
<linker>
  <assembly fullname="MyAssembly">
    <type fullname="MyType">
      <method name="MyMethod">
        <parameter name="methodParameter">
          <attribute fullname="FirstAttribute"/>
        </parameter>
        <parameter name="methodParameter">
          <attribute fullname="SecondAttribute"/>
        </parameter>
      </method>
    </type>
  </assembly>
</linker>
```

IL2025: Duplicate preserve of a member in a descriptor file

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

A member on a type is marked for preservation more than once in a descriptor file.

Rule description

[Descriptor files](#) are used to instruct the trimmer to always keep certain items in an assembly, regardless of whether the trimmer could find any references to them.

Members in this file should only appear once.

Example

```
<!-- IL2025: Duplicate preserve of 'method' -->
<linker>
  <assembly fullname="MyAssembly">
    <type fullname="MyType">
      <method name="MyMethod"/>
      <method name="MyMethod"/>
    </type>
  </assembly>
</linker>
```

IL2026: Members attributed with RequiresUnreferencedCode may break when trimming

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

Calling (or accessing via reflection) a member annotated with [RequiresUnreferencedCodeAttribute](#).

For example:

```
[RequiresUnreferencedCode("Use 'MethodFriendlyToTrimming' instead", Url="http://help/unreferencedcode")]
void MethodWithUnreferencedCodeUsage()
{
}

void TestMethod()
{
    // IL2026: Using method 'MethodWithUnreferencedCodeUsage' which has 'RequiresUnreferencedCodeAttribute'
    // can break functionality when trimming application code. Use 'MethodFriendlyToTrimming' instead.
    http://help/unreferencedcode
    MethodWithUnreferencedCodeUsage();
}
```

Rule description

[RequiresUnreferencedCodeAttribute](#) indicates that the member references code that may be removed by the trimmer.

Common examples include:

- [Load\(String\)](#) is marked as `RequiresUnreferencedCode` because the Assembly being loaded may access members that have been trimmed away. The trimmer removes all members from the framework except the ones directly used by the application, so it is likely that loading new Assemblies at run time will try to access missing members.
- [XmlSerializer](#) is marked as `RequiresUnreferencedCode` because `XmlSerializer` uses complex reflection to scan input types. The reflection cannot be tracked by the trimmer, so members transitively used by the input types may be trimmed away.

IL2027: Known trimmer attribute used more than once on a single member

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

Trimmer found multiple instances of the same trimmer-supported attribute on a single member.

IL2028: Known trimmer attribute does not have the required number of parameters

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

Trimmer found an instance of a known attribute that lacks the required constructor parameters or has more than the accepted parameters. This can happen if a custom assembly defines a custom attribute whose full name conflicts with the trimmer-known attributes, since the trimmer recognizes custom attributes by matching namespace and type name.

IL2029: Attribute element in custom attribute annotations file does not have required argument `fullname` or it is empty

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

An attribute element in a custom attribute annotations file does not have required argument `fullname` or its value is an empty string.

Rule description

Custom attribute annotation files are used to instruct the trimmer to behave as if the specified item has a given attribute. Attribute annotations can only be used to add attributes that have effect on the trimmer behavior. All other attributes are ignored. Attributes added via attribute annotations only influence the trimmer behavior, and they're never added to the output assembly.

All `attribute` elements must have the required `fullname` argument and its value cannot be an empty string.

Example

```
<!-- IL2029: 'attribute' element does not contain required attribute 'fullname' or it's empty -->
<linker>
  <assembly fullname="MyAssembly">
    <attribute/>
  </assembly>
</linker>
```

IL2030: Could not resolve an assembly specified in a custom attribute annotations file

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

Could not resolve assembly from the `assembly` argument of an attribute element in a custom attribute annotations file.

Rule description

[Custom attribute annotation files](#) are used to instruct the trimmer to behave as if the specified item has a given attribute. Attribute annotations can only be used to add attributes which have effect on the trimmer behavior, all other attributes will be ignored. Attributes added via attribute annotations only influence the trimmer behavior and they are never added to the output assembly.

The value of the `assembly` argument in an `attribute` element does not match any of the assemblies seen by the trimmer.

Example

```
<!-- IL2030: Could not resolve assembly 'NonExistentAssembly' for attribute 'MyAttribute' -->
<linker>
  <assembly fullname="MyAssembly">
    <attribute fullname="MyAttribute" assembly="NonExistentAssembly"/>
  </assembly>
</linker>
```

IL2031: Could not resolve custom attribute specified in a custom attribute annotations file

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

Could not resolve custom attribute from the type name specified in the `fullname` argument of an attribute element in a custom attribute annotations file.

Rule description

[Custom attribute annotation files](#) are used to instruct the trimmer to behave as if the specified item has a given attribute. Attribute annotations can only be used to add attributes which have effect on the trimmer behavior, all other attributes will be ignored. Attributes added via attribute annotations only influence the trimmer behavior and they are never added to the output assembly.

An attribute specified in a custom attribute annotations file could not be found in the assembly matching the `fullname` argument that was passed to the parent of the `attribute` element.

Example

```
<!-- IL2031: Attribute type 'NonExistentTypeAttribute' could not be found -->
<linker>
  <assembly fullname="MyAssembly">
    <attribute fullname="NonExistentTypeAttribute"/>
  </assembly>
</linker>
```

IL2032: Unrecognized value passed to the parameter 'parameter' of 'System.Activator.CreateInstance' method

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

The value passed to the assembly or type name of the `CreateInstance` method cannot be statically analyzed. The trimmer cannot guarantee the availability of the target type.

Example

```
void TestMethod(string assemblyName, string typeName)
{
    // IL2032 Trim analysis: Unrecognized value passed to the parameter 'typeName' of method
    // 'System.Activator.CreateInstance(string, string)'. It's not possible to guarantee the availability of the
    // target type.
    Activator.CreateInstance("MyAssembly", typeName);

    // IL2032 Trim analysis: Unrecognized value passed to the parameter 'assemblyName' of method
    // 'System.Activator.CreateInstance(string, string)'. It's not possible to guarantee the availability of the
    // target type.
    Activator.CreateInstance(assemblyName, "MyType");
}
```

IL2033: 'PreserveDependencyAttribute' is deprecated

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

`PreserveDependencyAttribute` was an internal attribute used by the trimmer and is not supported. Use `DynamicDependencyAttribute` instead.

Example

```
// IL2033: 'PreserveDependencyAttribute' is deprecated. Use 'DynamicDependencyAttribute' instead.  
[PreserveDependency("OtherMethod")]  
public void TestMethod()  
{  
}
```

IL2034: 'DynamicDependencyAttribute' could not be analyzed

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

Application contains an invalid use of [DynamicDependencyAttribute](#). Ensure that you are using one of the officially supported constructors.

IL2035: Unresolved assembly in 'DynamicDependencyAttribute'

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

The value passed to the `assemblyName` parameter of a [DynamicDependencyAttribute](#) could not be resolved.

Example

```
// IL2035: Unresolved assembly 'NonExistentAssembly' in 'DynamicDependencyAttribute'  
[DynamicDependency("Method", "Type", "NonExistentAssembly")]  
public void TestMethod()  
{  
}
```

IL2036: Unresolved type in 'DynamicDependencyAttribute'

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

The value passed to the `typeName` parameter of a [DynamicDependencyAttribute](#) could not be resolved.

Example

```
// IL2036: Unresolved type 'NonExistentType' in 'DynamicDependencyAttribute'  
[DynamicDependency("Method", "NonExistentType", "MyAssembly")]  
public void TestMethod()  
{  
}
```

IL2037: Unresolved member in 'DynamicDependencyAttribute'

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

The value passed to the member signature parameter of a [DynamicDependencyAttribute](#) could not resolve to any member. Ensure that the value passed refers to an existing member and that it uses the correct [ID string format](#).

Example

```
// IL2037: Unresolved type 'NonExistentType' in 'DynamicDependencyAttribute'  
[DynamicDependency("Method", "NonExistentType", "MyAssembly")]  
public void TestMethod()  
{  
}
```

IL2038: Missing `name` argument on a resource element in a substitution file

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

A `resource` element in a substitution file does not specify the required `name` argument.

Rule description

[Substitution files](#) are used to instruct the trimmer to replace specific method bodies with either a throw or return constant statements.

All `resource` elements in a substitution file must have the required `name` argument specifying the resource to remove.

Example

```
<!-- IL2038: Missing 'name' attribute for resource. -->
<linker>
  <assembly fullname="MyAssembly">
    <resource />
  </assembly>
</linker>
```

IL2039: Invalid `action` value on resource element in a substitution file

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

The value passed to the `action` argument of a `resource` element in a substitution file is not valid.

Rule description

[Substitution files](#) are used to instruct the trimmer to replace specific method bodies with either a throw or return constant statements.

The value passed to the `action` argument of a `resource` element was invalid. The only supported value for this argument is `remove`.

Example

```
<!-- IL2039: Invalid value 'NonExistentAction' for attribute 'action' for resource 'MyResource'. -->
<linker>
    <assembly fullname="MyAssembly">
        <resource name="MyResource" action="NonExistentAction"/>
    </assembly>
</linker>
```

IL2040: Could not find embedded resource specified in a substitution file

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

No embedded resource with name matching the value used in the `name` argument could be found in the specified assembly.

Rule description

[Substitution files](#) are used to instruct the trimmer to replace specific method bodies with either a throw or return constant statements.

The resource name in a substitution file could not be found in the specified assembly. The name of the resource to remove must match the name of an embedded resource in the assembly.

Example

```
<!-- IL2040: Could not find embedded resource 'NonExistentResource' to remove in assembly 'MyAssembly'. -->
<linker>
  <assembly fullname="MyAssembly">
    <resource name="NonExistentResource" action="remove"/>
  </assembly>
</linker>
```

IL2041: 'DynamicallyAccessedMembersAttribute' is not allowed on methods

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

[DynamicallyAccessedMembersAttribute](#) was put directly on a method. This is only allowed for instance methods on [Type](#). This attribute should usually be placed on the return value of the method or one of the parameters.

Example

```
// IL2041: The 'DynamicallyAccessedMembersAttribute' is not allowed on methods. It is allowed on method
return value or method parameters though.
[DynamicallyAccessedMembers(DynamicallyAccessedMemberType.PublicMethods)]

[return: DynamicallyAccessedMembers(DynamicallyAccessedMemberType.PublicMethods)]
public Type GetInterestingType()
{
}
```

IL2042: Could not find a unique backing field to propagate the 'DynamicallyAccessedMembersAttribute' annotation on a property

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

The trimmer could not determine the backing field of a property annotated with [DynamicallyAccessedMembersAttribute](#).

Example

```
// IL2042: Could not find a unique backing field for property 'MyProperty' to propagate
'DynamicallyAccessedMembersAttribute'
[DynamicallyAccessedMembers(DynamicallyAccessedMemberType.PublicMethods)]
public Type MyProperty
{
    get { return GetTheValue(); }
    set { }
}

// To fix this annotate the accessors manually:
public Type MyProperty
{
    [return: DynamicallyAccessedMembers(DynamicallyAccessedMemberType.PublicMethods)]
    get { return GetTheValue(); }

    [param: DynamicallyAccessedMembers(DynamicallyAccessedMemberType.PublicMethods)]
    set { }
}
```

IL2043: 'DynamicallyAccessedMembersAttribute' on property conflicts with the same attribute on its accessor method

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

While propagating [DynamicallyAccessedMembersAttribute](#) from the annotated property to its accessor method, the trimmer found that the accessor already has such an attribute. Only the existing attribute will be used.

Example

```
// IL2043: 'DynamicallyAccessedMembersAttribute' on property 'MyProperty' conflicts with the same attribute
// on its accessor 'get_MyProperty'.
[DynamicallyAccessedMembers(DynamicallyAccessedMemberType.PublicMethods)]
public Type MyProperty
{
    [return: DynamicallyAccessedMembers(DynamicallyAccessedMemberType.PublicFields)]
    get { return GetTheValue(); }
}
```

IL2044: Could not find any type in a namespace specified in a descriptor file

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

The descriptor file specified a namespace that has no types in it.

Rule description

[Descriptor files](#) are used to instruct the trimmer to always keep certain items in an assembly, regardless of whether the trimmer could find any references to them.

A namespace specified in the descriptor file could not be found in the assembly matching the `fullname` argument that was passed to the parent of the `namespace` element.

Example

```
<!-- IL2044: Could not find any type in namespace 'NonExistentNamespace' -->
<linker>
  <assembly fullname="MyAssembly">
    <namespace fullname="NonExistentNamespace" />
  </assembly>
</linker>
```

IL2045: Custom attribute is referenced in code but the trimmer was instructed to remove all of its instances

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

The trimmer was instructed to remove all instances of a custom attribute but kept its type as part of its analysis. This will likely result in breaking the code where the custom attribute's type is being used.

Example

```
<linker>
  <assembly fullname="MyAssembly">
    <type fullname="MyAttribute">
      <attribute internal="RemoveAttributeInstances"/>
    </type>
  </assembly>
</linker>
```

```
// This attribute instance will be removed
[MyAttribute]
class MyType
{
}

public void TestMethod()
{
    // IL2045 for 'MyAttribute' reference
    typeof(MyType).GetCustomAttributes(typeof(MyAttribute), false);
}
```

IL2046: All interface implementations and method overrides must have annotations matching the interface or overridden virtual method 'RequiresUnreferencedCodeAttribute' annotations

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

There is a mismatch in the `RequiresUnreferencedCodeAttribute` annotations between an interface and its implementation or a virtual method and its override.

Example

A base member has the attribute but the derived member does not have the attribute.

```
public class Base
{
    [RequiresUnreferencedCode("Message")]
    public virtual void TestMethod() {}
}

public class Derived : Base
{
    // IL2046: Base member 'Base.TestMethod' with 'RequiresUnreferencedCodeAttribute' has a derived member
    // 'Derived.TestMethod()' without 'RequiresUnreferencedCodeAttribute'. For all interfaces and overrides the
    // implementation attribute must match the definition attribute.
    public override void TestMethod() {}
}
```

A derived member has the attribute but the overridden base member does not have the attribute.

```
public class Base
{
    public virtual void TestMethod() {}
}

public class Derived : Base
{
    // IL2046: Member 'Derived.TestMethod()' with 'RequiresUnreferencedCodeAttribute' overrides base member
    // 'Base.TestMethod()' without 'RequiresUnreferencedCodeAttribute'. For all interfaces and overrides the
    // implementation attribute must match the definition attribute.
    [RequiresUnreferencedCode("Message")]
    public override void TestMethod() {}
}
```

An interface member has the attribute but its implementation does not have the attribute.

```
interface IRUC
{
    [RequiresUnreferencedCode("Message")]
    void TestMethod();
}

class Implementation : IRUC
{
    // IL2046: Interface member 'IRUC.TestMethod()' with 'RequiresUnreferencedCodeAttribute' has an
    // implementation member 'Implementation.TestMethod()' without 'RequiresUnreferencedCodeAttribute'. For all
    // interfaces and overrides the implementation attribute must match the definition attribute.
    public void TestMethod () { }
}
```

An implementation member has the attribute but the interface that it implements does not have the attribute.

```
interface IRUC
{
    void TestMethod();
}

class Implementation : IRUC
{
    [RequiresUnreferencedCode("Message")]
    // IL2046: Member 'Implementation.TestMethod()' with 'RequiresUnreferencedCodeAttribute' implements
    // interface member 'IRUC.TestMethod()' without 'RequiresUnreferencedCodeAttribute'. For all interfaces and
    // overrides the implementation attribute must match the definition attribute.
    public void TestMethod () { }
}
```

IL2048: Internal trimmer attribute 'RemoveAttributeInstances' is being used on a member

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

Internal trimmer attribute `RemoveAttributeInstances` is being used on a member but it can only be used on a type.

Example

```
<!-- IL2048: Internal attribute 'RemoveAttributeInstances' can only be used on a type, but is being used on  
'MyMethod' -->  
<linker>  
  <assembly fullname="MyAssembly">  
    <type fullname="MyType">  
      <method name="MyMethod">  
        <attribute internal="RemoveAttributeInstances" />  
      </method>  
    </type>  
  </assembly>  
</linker>
```

IL2049: Unrecognized internal attribute

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

An internal attribute name specified in a custom attribute annotations file is not supported by the trimmer.

Example

```
<!-- IL2049: Unrecognized internal attribute 'InvalidInternalAttributeName' -->
<linker>
  <assembly fullname="MyAssembly">
    <type fullname="MyType">
      <method name="MyMethod">
        <attribute internal="InvalidInternalAttributeName" />
      </method>
    </type>
  </assembly>
</linker>
```

IL2050: Correctness of COM interop cannot be guaranteed

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

Trimmer found a p/invoke method that declares a parameter with COM marshalling. Correctness of COM interop cannot be guaranteed after trimming.

Example

```
// IL2050: M1(): P/invoke method 'M2(C)' declares a parameter with COM marshalling. Correctness of COM
interop cannot be guaranteed after trimming. Interfaces and interface members might be removed.
static void M1 ()
{
    M2 (null);
}

[DllImport ("Foo")]
static extern void M2 (C autoLayout);

[StructLayout (LayoutKind.Auto)]
public class C
{}
```

IL2051: Property element does not have required argument `name` in custom attribute annotations file

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

A property element in a custom attribute annotations file does not specify the required argument `name`.

Example

```
<!-- IL2051: Property element does not contain attribute 'name' -->
<linker>
  <assembly fullname="MyAssembly">
    <type fullname="MyType">
      <attribute fullname="MyAttribute">
        <property>UnspecifiedPropertyName</property>
      </attribute>
    </type>
  </assembly>
</linker>
```

IL2052: Could not find property specified in custom attribute annotations file

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

Could not find a property matching the value of the `name` argument specified in a `property` element in a custom attribute annotations file.

Example

```
<!-- IL2052: Property 'NonExistentPropertyName' could not be found -->
<linker>
  <assembly fullname="MyAssembly">
    <type fullname="MyType">
      <attribute fullname="MyAttribute">
        <property name="NonExistentPropertyName">SomeValue</property>
      </attribute>
    </type>
  </assembly>
</linker>
```

IL2053: Invalid value used in property element in custom attribute annotations file

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

Value used in a `property` element in a custom attribute annotations file does not match the type of the attribute's property.

Example

```
<!-- IL2053: Invalid value 'StringValue' for property 'IntProperty' -->
<linker>
  <assembly fullname="MyAssembly">
    <type fullname="MyType">
      <attribute fullname="MyAttribute">
        <property name="IntProperty">StringValue</property>
      </attribute>
    </type>
  </assembly>
</linker>
```

IL2054: Invalid argument value in custom attribute annotations file

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

Value used in an `argument` element in a custom attribute annotations file does not match the type of the attribute's constructor arguments.

Example

```
<!-- IL2054: Invalid argument value 'NonExistentEnumValue' for parameter of type 'MyEnumType' of attribute
'AttributeWithEnumParameterAttribute' -->
<linker>
  <assembly fullname="MyAssembly">
    <type fullname="MyType">
      <attribute fullname="AttributeWithEnumParameterAttribute">
        <argument>NonExistentEnumValue</argument>
      </attribute>
    </type>
  </assembly>
</linker>
```

IL2055: Call to 'System.Type.MakeGenericType' cannot be statically analyzed by the trimmer

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

A call to `Type.MakeGenericType(Type[])` cannot be statically analyzed by the trimmer.

Rule description

This can either be that the type on which `MakeGenericType(Type[])` is called cannot be statically determined, or that the type parameters to be used for generic arguments cannot be statically determined. If the open generic type has `DynamicallyAccessedMembersAttribute` annotations on any of its generic parameters, the trimmer currently cannot validate that the requirements are fulfilled by the calling method.

Example

```
class Lazy<[DynamicallyAccessedMembers(DynamicallyAccessedMemberType.PublicParameterlessConstructor)] T>
{
    // ...
}

void TestMethod(Type unknownType)
{
    // IL2055 Trim analysis: Call to `System.Type.MakeGenericType(Type[])` can not be statically analyzed.
    It's not possible to guarantee the availability of requirements of the generic type.
    typeof(Lazy<>).MakeGenericType(new Type[] { typeof(TestType) });

    // IL2055 Trim analysis: Call to `System.Type.MakeGenericType(Type[])` can not be statically analyzed.
    It's not possible to guarantee the availability of requirements of the generic type.
    unknownType.MakeGenericType(new Type[] { typeof(TestType) });
}
```

IL2056: A

'System.Diagnostics.CodeAnalysis.DynamicallyAccessedMembersAttribute' annotation on a property conflicts with the same attribute on its backing field

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

Property annotated with [DynamicallyAccessedMembersAttribute](#) also has that attribute on its backing field.

Rule description

While propagating [DynamicallyAccessedMembersAttribute](#) from a property to its backing field, the trimmer found its backing field to be already annotated. Only the existing attribute will be used.

The trimmer will only propagate annotations to compiler generated backing fields, making this warning only possible when the backing field is explicitly annotated with [CompilerGeneratedAttribute](#).

Example

```
[DynamicallyAccessedMembers(DynamicallyAccessedMemberTypes.PublicConstructors)]
[CompilerGenerated]
Type backingField;

[DynamicallyAccessedMembers(DynamicallyAccessedMemberTypes.PublicConstructors)]
Type PropertyWithAnnotatedBackingField
{
    get { return backingField; }
    set { backingField = value; }
}
```

IL2057: Unrecognized value passed to the `typeName` parameter of `'System.Type.GetType(String)'`

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

An unrecognized value was passed to the `typeName` parameter of `Type.GetType(String)`.

Rule description

If the type name passed to the `typeName` parameter of `GetType(String)` is statically known, the trimmer can make sure it is preserved and that the application code will continue to work after trimming. If the type is unknown and the trimmer cannot see the type being used anywhere else, the trimmer might end up removing it from the application, potentially breaking it.

Example

```
void TestMethod()
{
    string typeName = ReadName();

    // IL2057 Trim analysis: Unrecognized value passed to the parameter 'typeName' of method
    'System.Type.GetType(String typeName)'
    Type.GetType(typeName);
}
```

IL2058: Parameters passed to 'Assembly.CreateInstance' cannot be statically analyzed

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

A call to `CreateInstance` was found in the analyzed code.

Rule description

Trimmer does not analyze assembly instances and thus does not know on which assembly `CreateInstance` was called.

Example

```
void TestMethod()
{
    // IL2058 Trim analysis: Parameters passed to method 'Assembly.CreateInstance(string)' cannot be
    // analyzed. Consider using methods 'System.Type.GetType' and `System.Activator.CreateInstance` instead.
    AssemblyLoadContext.Default.Assemblies.First(a => a.Name == "MyAssembly").CreateInstance("MyType");

    // This can be replaced by
    Activator.CreateInstance(Type.GetType("MyType, MyAssembly"));
}
```

How to fix

Trimmer has support for `Type.GetType(String)`. The result can be passed to `CreateInstance` to create an instance of the type.

IL2059: Unrecognized value passed to the `type` parameter of 'System.Runtime.CompilerServices.RuntimeHelpers.RunClassConstructo

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

An unrecognized value was passed to the `type` parameter of `RuntimeHelpers.RunClassConstructor(RuntimeTypeHandle)`.

Rule description

If the type passed to `RunClassConstructor(RuntimeTypeHandle)` is not statically known, the trimmer cannot guarantee the availability of the target static constructor.

Example

```
void TestMethod(Type type)
{
    // IL2059 Trim analysis: Unrecognized value passed to the parameter 'type' of method
    'System.Runtime.CompilerServices.RuntimeHelpers.RunClassConstructor(RuntimeTypeHandle type)'.
    // It's not possible to guarantee the availability of the target static constructor.
    RuntimeHelpers.RunClassConstructor(type.TypeHandle);
}
```

IL2060: Call to 'System.Reflection.MethodInfo.MakeGenericMethod' cannot be statically analyzed by the trimmer

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

A call to [MethodInfo.MakeGenericMethod\(Type\[\]\)](#) cannot be statically analyzed by the trimmer.

Rule description

This can either be that the method on which the [MakeGenericMethod\(Type\[\]\)](#) is called cannot be statically determined, or that the type parameters to be used for the generic arguments cannot be statically determined. If the open generic method has [DynamicallyAccessedMembersAttribute](#) annotations on any of its generic parameters, the trimmer currently cannot validate that the requirements are fulfilled by the calling method.

Example

```
class Test
{
    public static void
TestGenericMethod<[DynamicallyAccessedMembers(DynamicallyAccessedMemberTypes.PublicProperties)] T>()
    {
    }

    void TestMethod(Type unknownType)
    {
        // IL2060 Trim analysis: Call to 'System.Reflection.MethodInfo.MakeGenericMethod' can not be statically
        // analyzed. It's not possible to guarantee the availability of requirements of the generic method
        typeof(Test).GetMethod("TestGenericMethod").MakeGenericMethod(new Type[] { typeof(TestType) });

        // IL2060 Trim analysis: Call to 'System.Reflection.MethodInfo.MakeGenericMethod' can not be statically
        // analyzed. It's not possible to guarantee the availability of requirements of the generic method
        unknownMethod.MakeGenericMethod(new Type[] { typeof(TestType) });
    }
}
```

IL2061: Assembly name passed to method 'Assembly.CreateInstance' references an assembly that could not be resolved

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

A call to `CreateInstance` had an assembly that could not be resolved.

Example

```
void TestMethod()
{
    // IL2061 Trim analysis: The assembly name 'NonExistentAssembly' passed to method
    // 'System.Activator.CreateInstance(string, string)' references assembly which is not available.
    Activator.CreateInstance("NonExistentAssembly", "MyType");
}
```

IL2062: Value passed to a method parameter annotated with

'DynamicallyAccessedMembersAttribute' cannot be statically determined and may not meet the attribute's requirements

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

The parameter 'parameter' of method 'method' has a [DynamicallyAccessedMembersAttribute](#) annotation, but the value passed to it can't be statically analyzed. Trimmer cannot make sure that the requirements declared by the attribute are met by the argument value.

Example

```
void NeedsPublicConstructors([DynamicallyAccessedMembers(DynamicallyAccessedMemberTypes.PublicConstructors)]
Type type)
{
    // ...
}

void TestMethod(Type[] types)
{
    // IL2062: Value passed to parameter 'type' of method 'NeedsPublicConstructors' can not be statically
    // determined and may not meet 'DynamicallyAccessedMembersAttribute' requirements.
    NeedsPublicConstructors(types[1]);
}
```

IL2063: Value returned from a method whose return type is annotated with 'DynamicallyAccessedMembersAttribute' cannot be statically determined and may not meet the attribute's requirements

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

The return value of method 'method' has a [DynamicallyAccessedMembersAttribute](#) annotation, but the value returned from the method cannot be statically analyzed. Trimmer cannot make sure that the requirements declared by the attribute are met by the returned value.

Example

```
[return: DynamicallyAccessedMembers(DynamicallyAccessedMemberTypes.PublicConstructors)]
Type TestMethod(Type[] types)
{
    // IL2063 Trim analysis: Value returned from method 'TestMethod' can not be statically determined and
    // may not meet 'DynamicallyAccessedMembersAttribute' requirements.
    return types[1];
}
```

IL2064: Value assigned to a field annotated with 'DynamicallyAccessedMembersAttribute' cannot be statically determined and may not meet the attribute's requirements.

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

The field 'field' has a [DynamicallyAccessedMembersAttribute](#) annotation, but the value assigned to it can not be statically analyzed. Trimmer cannot make sure that the requirements declared by the attribute are met by the assigned value.

Example

```
[DynamicallyAccessedMembers(DynamicallyAccessedMemberTypes.PublicConstructors)]
Type _typeField;

void TestMethod(Type[] types)
{
    // IL2064 Trim analysis: Value assigned to field '_typeField' can not be statically determined and may
    // not meet 'DynamicallyAccessedMembersAttribute' requirements.
    _typeField = _types[1];
}
```

IL2065: Value passed to the implicit `this` parameter of a method annotated with 'DynamicallyAccessedMembersAttribute' cannot be statically determined and may not meet the attribute's requirements.

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

The method 'method' has a `DynamicallyAccessedMembersAttribute` annotation (which applies to the implicit `this` parameter), but the value used for the `this` parameter cannot be statically analyzed. Trimmer cannot make sure that the requirements declared by the attribute are met by the `this` value.

Example

```
void TestMethod(Type[] types)
{
    // IL2065 Trim analysis: Value passed to implicit 'this' parameter of method 'Type.GetMethods()' can not
    be statically determined and may not meet 'DynamicallyAccessedMembersAttribute' requirements.
    _types[1].GetMethods(); // Type.GetMethods has
    [DynamicallyAccessedMembers(DynamicallyAccessedMemberTypes.PublicMethods)] attribute
}
```

IL2066: Type passed to generic parameter 'parameter' of 'type' (or 'method') cannot be statically determined and may not meet 'DynamicallyAccessedMembersAttribute' requirements

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

The generic parameter 'parameter' of 'type' (or 'method') is annotated with [DynamicallyAccessedMembersAttribute](#), but the value used for it cannot be statically analyzed. Trimmer cannot make sure that the requirements declared on the attribute are met by the value.

Example

```
static void MethodWithUnresolvedGenericArgument<[DynamicallyAccessedMembers
(DynamicallyAccessedMemberTypes.PublicMethods)] T>()
{
}

// IL2066: TestMethod(): Type passed to generic parameter 'T' of 'TypeWithUnresolvedGenericArgument<T>' can
// not be statically determined and may not meet 'DynamicallyAccessedMembersAttribute' requirements.
// IL2066: TestMethod(): Type passed to generic parameter 'T' of 'MethodWithUnresolvedGenericArgument<T>()' can
// not be statically determined and may not meet 'DynamicallyAccessedMembersAttribute' requirements.
static void TestMethod()
{
    var _ = new TypeWithUnresolvedGenericArgument<Dependencies.UnresolvedType>();
    MethodWithUnresolvedGenericArgument<Dependencies.UnresolvedType>();
}
```

IL2067: 'target parameter' argument does not satisfy 'DynamicallyAccessedMembersAttribute' in call to 'target method'. The parameter 'source parameter' of method 'source method' does not have matching annotations. The source value must declare at least the same requirements as those declared on the target location it is assigned to

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

The target location declares some requirements on the type value via its [DynamicallyAccessedMembersAttribute](#). Those requirements must be met by those declared on the source value also via the [DynamicallyAccessedMembersAttribute](#). The source value can declare more requirements than the target if necessary.

Example

```
void NeedsPublicConstructors([DynamicallyAccessedMembers(DynamicallyAccessedMemberTypes.PublicConstructors)]
Type type)
{
    // ...
}

void TestMethod(Type type)
{
    // IL2067 Trim analysis: 'type' argument does not satisfy 'DynamicallyAccessedMembersAttribute' in call
    // to 'NeedsPublicConstructor'. The parameter 'type' of method 'TestMethod' does not have matching annotations.
    // The source value must declare at least the same requirements as those declared on the target location it is
    // assigned to.
    NeedsPublicConstructors(type);
}
```

Fixing

See [Fixing Warnings](#) for guidance.

IL2068: 'target method' method return value does not satisfy 'DynamicallyAccessedMembersAttribute' requirements. The parameter 'source parameter' of method 'source method' does not have matching annotations. The source value must declare at least the same requirements as those declared on the target location it is assigned to

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

The target location declares some requirements on the type value via its [DynamicallyAccessedMembersAttribute](#). Those requirements must be met by those declared on the source value also via the [DynamicallyAccessedMembersAttribute](#). The source value can declare more requirements than the target, if necessary.

Example

```
[return: DynamicallyAccessedMembers(DynamicallyAccessedMemberTypes.PublicConstructors)]
Type TestMethod(Type type)
{
    // IL2068 Trim analysis: 'TestMethod' method return value does not satisfy
    // 'DynamicallyAccessedMembersAttribute' requirements. The parameter 'type' of method 'TestMethod' does not
    // have matching annotations. The source value must declare at least the same requirements as those declared on
    // the target location it is assigned to.
    return type;
}
```

Fixing

See [Fixing Warnings](#) for guidance.

IL2069: Value stored in field 'target field' does not satisfy 'DynamicallyAccessedMembersAttribute' requirements. The parameter 'source parameter' of method 'source method' does not have matching annotations. The source value must declare at least the same requirements as those declared on the target location it is assigned to

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

The target location declares some requirements on the type value via its [DynamicallyAccessedMembersAttribute](#). Those requirements must be met by those declared on the source value also via the [DynamicallyAccessedMembersAttribute](#). The source value can declare more requirements than the target, if necessary.

Example

```
[DynamicallyAccessedMembers(DynamicallyAccessedMemberTypes.PublicConstructors)]
Type _typeField;

void TestMethod(Type type)
{
    // IL2069 Trim analysis: value stored in field '_typeField' does not satisfy
    // 'DynamicallyAccessedMembersAttribute' requirements. The parameter 'type' of method 'TestMethod' does not
    // have matching annotations. The source value must declare at least the same requirements as those declared on
    // the target location it is assigned to.
    _typeField = type;
}
```

Fixing

See [Fixing Warnings](#) for guidance.

IL2070: 'this' argument does not satisfy 'DynamicallyAccessedMembersAttribute' in call to 'target method'. The parameter 'source parameter' of method 'source method' does not have matching annotations. The source value must declare at least the same requirements as those declared on the target location it is assigned to

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

The target location declares some requirements on the type value via its [DynamicallyAccessedMembersAttribute](#). Those requirements must be met by those declared on the source value also via the [DynamicallyAccessedMembersAttribute](#). The source value can declare more requirements than the target, if necessary.

Example

```
void GenericWithAnnotation<[DynamicallyAccessedMembers(DynamicallyAccessedMemberTypes.Interfaces)] T>()
{
}

void TestMethod(Type type)
{
    // IL2070 Trim analysis: 'this' argument does not satisfy 'DynamicallyAccessedMemberTypes.Interfaces' in
    // call to 'System.Reflection.MethodInfo.MakeGenericMethod(Type[])'. The parameter 'type' of method
    // 'TestMethod' does not have matching annotations. The source value must declare at least the same
    // requirements as those declared on the target location it is assigned to.
    typeof(AnnotatedGenerics).GetMethod(nameof(GenericWithAnnotation)).MakeGenericMethod(type);
}
```

Fixing

See [Fixing Warnings](#) for guidance.

IL2072: 'target parameter' argument does not satisfy 'DynamicallyAccessedMembersAttribute' in call to 'target method'. The return value of method 'source method' does not have matching annotations. The source value must declare at least the same requirements as those declared on the target location it is assigned to

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

The target location declares some requirements on the type value via its [DynamicallyAccessedMembersAttribute](#). Those requirements must be declared by the source value also via the [DynamicallyAccessedMembersAttribute](#). The source value can declare more requirements than the target, if necessary.

Example

```
Type GetCustomType() { return typeof(CustomType); }

void NeedsPublicConstructors([DynamicallyAccessedMembers(DynamicallyAccessedMemberTypes.PublicConstructors)]
    Type type)
{
    // ...
}

void TestMethod()
{
    // IL2072 Trim analysis: 'type' argument does not satisfy 'DynamicallyAccessedMembersAttribute' in call
    // to 'NeedsPublicConstructors'. The return value of method 'GetCustomType' does not have matching annotations.
    // The source value must declare at least the same requirements as those declared on the target location it is
    // assigned to.
    NeedsPublicConstructors(GetCustomType());
}
```

Fixing

See [Fixing Warnings](#) for guidance.

IL2073: 'target method' method return value does not satisfy 'DynamicallyAccessedMembersAttribute' requirements. The return value of method 'source method' does not have matching annotations. The source value must declare at least the same requirements as those declared on the target location it is assigned to

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

The target location declares some requirements on the type value via its [DynamicallyAccessedMembersAttribute](#). Those requirements must be declared by the source value also via the [DynamicallyAccessedMembersAttribute](#). The source value can declare more requirements than the target, if necessary.

Example

```
Type GetCustomType() { return typeof(CustomType); }

[return: DynamicallyAccessedMembers(DynamicallyAccessedMemberTypes.PublicConstructors)]
Type TestMethod()
{
    // IL2073 Trim analysis: 'TestMethod' method return value does not satisfy
    'DynamicallyAccessedMembersAttribute' requirements. The return value of method 'GetCustomType' does not have
    matching annotations. The source value must declare at least the same requirements as those declared on the
    target location it is assigned to.
    return GetCustomType();
}
```

Fixing

See [Fixing Warnings](#) for guidance.

IL2074: Value stored in field 'target field' does not satisfy 'DynamicallyAccessedMembersAttribute' requirements. The return value of method 'source method' does not have matching annotations. The source value must declare at least the same requirements as those declared on the target location it is assigned to

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

The target location declares some requirements on the type value via its [DynamicallyAccessedMembersAttribute](#). Those requirements must be declared by the source value also via the [DynamicallyAccessedMembersAttribute](#). The source value can declare more requirements than the target, if necessary.

Example

```
Type GetCustomType() { return typeof(CustomType); }

[DynamicallyAccessedMembers(DynamicallyAccessedMemberTypes.PublicConstructors)]
Type _typeField;

void TestMethod()
{
    // IL2074 Trim analysis: value stored in field '_typeField_' does not satisfy
    // 'DynamicallyAccessedMembersAttribute' requirements. The return value of method 'GetCustomType' does not have
    // matching annotations. The source value must declare at least the same requirements as those declared on the
    // target location it is assigned to.
    _typeField = GetCustomType();
}
```

Fixing

See [Fixing Warnings](#) for guidance.

IL2075: 'this' argument does not satisfy 'DynamicallyAccessedMembersAttribute' in call to 'target method'. The return value of method 'source method' does not have matching annotations. The source value must declare at least the same requirements as those declared on the target location it is assigned to

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

The target location declares some requirements on the type value via its [DynamicallyAccessedMembersAttribute](#). Those requirements must be declared by the source value also via the [DynamicallyAccessedMembersAttribute](#). The source value can declare more requirements than the target, if necessary.

Example

```
Type GetCustomType() { return typeof(CustomType); }

void TestMethod()
{
    // IL2075 Trim analysis: 'this' argument does not satisfy 'DynamicallyAccessedMembersAttribute' in call
    // to 'GetMethods'. The return value of method 'GetCustomType' does not have matching annotations. The source
    // value must declare at least the same requirements as those declared on the target location it is assigned
    // to.
    GetCustomType().GetMethods(); // Type.GetMethods is annotated with
    DynamicallyAccessedMemberTypes.PublicMethods
}
```

Fixing

See [Fixing Warnings](#) for guidance.

IL2077: 'target parameter' argument does not satisfy 'DynamicallyAccessedMembersAttribute' in call to 'target method'. The field 'source field' does not have matching annotations. The source value must declare at least the same requirements as those declared on the target location it is assigned to

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

The target location declares some requirements on the type value via its [DynamicallyAccessedMembersAttribute](#). Those requirements must be declared by the source value also via the [DynamicallyAccessedMembersAttribute](#). The source value can declare more requirements than the target, if necessary.

Example

```
void NeedsPublicConstructors([DynamicallyAccessedMembers(DynamicallyAccessedMemberTypes.PublicConstructors)]
Type type)
{
    // ...
}

Type _typeField;

void TestMethod()
{
    // IL2077 Trim analysis: 'type' argument does not satisfy 'DynamicallyAccessedMembersAttribute' in call
    // to 'NeedsPublicConstructors'. The field '_typeField' does not have matching annotations. The source value
    // must declare at least the same requirements as those declared on the target location it is assigned to.
    NeedsPublicConstructors(_typeField);
}
```

Fixing

See [Fixing Warnings](#) for guidance.

IL2078: 'target method' method return value does not satisfy 'DynamicallyAccessedMembersAttribute' requirements. The field 'source field' does not have matching annotations. The source value must declare at least the same requirements as those declared on the target location it is assigned to

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

The target location declares some requirements on the type value via its [DynamicallyAccessedMembersAttribute](#). Those requirements must be declared by the source value also via the [DynamicallyAccessedMembersAttribute](#). The source value can declare more requirements than the target, if necessary.

Example

```
Type _typeField;

[return: DynamicallyAccessedMembers(DynamicallyAccessedMemberTypes.PublicConstructors)]
Type TestMethod()
{
    // IL2078 Trim analysis: 'TestMethod' method return value does not satisfy
    // 'DynamicallyAccessedMembersAttribute' requirements. The field '_typeField' does not have matching
    // annotations. The source value must declare at least the same requirements as those declared on the target
    // location it is assigned to.
    return _typeField;
}
```

Fixing

See [Fixing Warnings](#) for guidance.

IL2079: Value stored in field 'target field' does not satisfy 'DynamicallyAccessedMembersAttribute' requirements. The field 'source field' does not have matching annotations. The source value must declare at least the same requirements as those declared on the target location it is assigned to

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

The target location declares some requirements on the type value via its [DynamicallyAccessedMembersAttribute](#). Those requirements must be declared by the source value also via the [DynamicallyAccessedMembersAttribute](#). The source value can declare more requirements than the target, if necessary.

Example

```
Type _typeField;

[DynamicallyAccessedMembers(DynamicallyAccessedMemberTypes.PublicConstructors)]
Type _typeFieldWithRequirements;

void TestMethod()
{
    // IL2079 Trim analysis: value stored in field '_typeFieldWithRequirements' does not satisfy
    // 'DynamicallyAccessedMembersAttribute' requirements. The field '_typeField' does not have matching
    // annotations. The source value must declare at least the same requirements as those declared on the target
    // location it is assigned to.
    _typeFieldWithRequirements = _typeField;
}
```

Fixing

See [Fixing Warnings](#) for guidance.

IL2080: 'this' argument does not satisfy 'DynamicallyAccessedMembersAttribute' in call to 'target method'. The field 'source field' does not have matching annotations. The source value must declare at least the same requirements as those declared on the target location it is assigned to

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

The target location declares some requirements on the type value via its [DynamicallyAccessedMembersAttribute](#). Those requirements must be declared by the source value also via the [DynamicallyAccessedMembersAttribute](#). The source value can declare more requirements than the target, if necessary.

Example

```
[DynamicallyAccessedMembers(DynamicallyAccessedMemberTypes.PublicConstructors)]
Type _typeFieldWithRequirements;

void TestMethod()
{
    // IL2080 Trim analysis: 'this' argument does not satisfy 'DynamicallyAccessedMemberTypes' in call to
    // 'GetMethod'. The field '_typeFieldWithRequirements' does not have matching annotations. The source value
    // must declare at least the same requirements as those declared on the target location it is assigned to.
    _typeFieldWithRequirements.GetMethod("Foo");
}
```

Fixing

See [Fixing Warnings](#) for guidance.

IL2082: 'target parameter' argument does not satisfy 'DynamicallyAccessedMembersAttribute' in call to 'target method'. The implicit 'this' argument of method 'source method' does not have matching annotations. The source value must declare at least the same requirements as those declared on the target location it is assigned to

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

The target location declares some requirements on the type value via its [DynamicallyAccessedMembersAttribute](#). Those requirements must be declared by the source value also via the [DynamicallyAccessedMembersAttribute](#). The source value can declare more requirements than the target, if necessary.

Example

```
void NeedsPublicConstructors([DynamicallyAccessedMembers(DynamicallyAccessedMemberTypes.PublicConstructors)]
Type type)
{
    // ...
}

// This can only happen within methods of System.Type type (or derived types). Assume the below method is
// declared on System.Type
void TestMethod()
{
    // IL2082 Trim analysis: 'type' argument does not satisfy 'DynamicallyAccessedMembersAttribute' in call
    // to 'NeedsPublicConstructors'. The implicit 'this' argument of method 'TestMethod' does not have matching
    // annotations. The source value must declare at least the same requirements as those declared on the target
    // location it is assigned to.
    NeedsPublicConstructors(this);
}
```

Fixing

See [Fixing Warnings](#) for guidance.

IL2083: 'target method' method return value does not satisfy 'DynamicallyAccessedMembersAttribute' requirements. The implicit 'this' argument of method 'source method' does not have matching annotations. The source value must declare at least the same requirements as those declared on the target location it is assigned to

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

The target location declares some requirements on the type value via its [DynamicallyAccessedMembersAttribute](#). Those requirements must be met by those declared on the source value also via the [DynamicallyAccessedMembersAttribute](#). The source value can declare more requirements than the source if necessary.

Example

```
// This can only happen within methods of System.Type type (or derived types). Assume the below method is
declared on System.Type
[DynamicallyAccessedMembers (DynamicallyAccessedMemberTypes.PublicMethods)]
[return: DynamicallyAccessedMembers(DynamicallyAccessedMemberTypes.PublicConstructors)]
Type TestMethod()
{
    // IL2083 Trim analysis: 'TestMethod' method return value does not satisfy
    'DynamicallyAccessedMembersAttribute' requirements. The implicit 'this' argument of method 'TestMethod' does
    not have matching annotations. The source value must declare at least the same requirements as those
    declared on the target location it is assigned to.
    return this;
}
```

Fixing

See [Fixing Warnings](#) for guidance.

IL2084: Value stored in field 'target field' does not satisfy 'DynamicallyAccessedMembersAttribute' requirements. The implicit 'this' argument of method 'source method' does not have matching annotations. The source value must declare at least the same requirements as those declared on the target location it is assigned to

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

The target location declares some requirements on the type value via its [DynamicallyAccessedMembersAttribute](#). Those requirements must be met by those declared on the source value also via the [DynamicallyAccessedMembersAttribute](#). The source value can declare more requirements than the source if necessary.

Example

```
[DynamicallyAccessedMembers(DynamicallyAccessedMemberTypes.PublicConstructors)]
Type _typeFieldWithRequirements;

// This can only happen within methods of System.Type type (or derived types). Assume the below method is
declared on System.Type
void TestMethod()
{
    // IL2084 Trim analysis: value stored in field '_typeFieldWithRequirements' does not satisfy
    'DynamicallyAccessedMembersAttribute' requirements. The implicit 'this' argument of method 'TestMethod' does
    not have matching annotations. The source value must declare at least the same requirements as those
    declared on the target location it is assigned to.
    _typeFieldWithRequirements = this;
}
```

Fixing

See [Fixing Warnings](#) for guidance.

IL2085: 'this' argument does not satisfy 'DynamicallyAccessedMembersAttribute' in call to 'target method'. The implicit 'this' argument of method 'source method' does not have matching annotations. The source value must declare at least the same requirements as those declared on the target location it is assigned to

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

The target location declares some requirements on the type value via its [DynamicallyAccessedMembersAttribute](#). Those requirements must be met by those declared on the source value also via the [DynamicallyAccessedMembersAttribute](#). The source value can declare more requirements than the source if necessary.

Example

```
// This can only happen within methods of System.Type type (or derived types). Assume the below method is
// declared on System.Type
void TestMethod()
{
    // IL2085 Trim analysis: 'this' argument does not satisfy 'DynamicallyAccessedMembersAttribute' in call
    // to 'GetMethods'. The implicit 'this' argument of method 'TestMethod' does not have matching annotations. The
    // source value must declare at least the same requirements as those declared on the target location it is
    // assigned to.
    this.GetMethods(); // Type.GetMethods is annotated with DynamicallyAccessedMemberTypes.PublicMethods
}
```

Fixing

See [Fixing Warnings](#) for guidance.

IL2087: 'target parameter' argument does not satisfy 'DynamicallyAccessedMembersAttribute' in call to 'target method'. The generic parameter 'source generic parameter' of 'source method or type' does not have matching annotations. The source value must declare at least the same requirements as those declared on the target location it is assigned to

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

The target location declares some requirements on the type value via its [DynamicallyAccessedMembersAttribute](#). Those requirements must be met by those declared on the source value also via the [DynamicallyAccessedMembersAttribute](#). The source value can declare more requirements than the source if necessary.

Example

```
void NeedsPublicConstructors([DynamicallyAccessedMembers(DynamicallyAccessedMemberTypes.PublicConstructors)]
Type type)
{
    // ...
}

void TestMethod<TSource>()
{
    // IL2087 Trim analysis: 'type' argument does not satisfy 'DynamicallyAccessedMembersAttribute' in call
    // to 'NeedsPublicConstructor'. The generic parameter 'TSource' of 'TestMethod' does not have matching
    // annotations. The source value must declare at least the same requirements as those declared on the target
    // location it is assigned to.
    NeedsPublicConstructors(typeof(TSource));
}
```

Fixing

See [Fixing Warnings](#) for guidance.

IL2088: 'target method' method return value does not satisfy 'DynamicallyAccessedMembersAttribute' requirements. The generic parameter 'source generic parameter' of 'source method or type' does not have matching annotations. The source value must declare at least the same requirements as those declared on the target location it is assigned to

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

The target location declares some requirements on the type value via its [DynamicallyAccessedMembersAttribute](#). Those requirements must be met by those declared on the source value also via the [DynamicallyAccessedMembersAttribute](#). The source value can declare more requirements than the source if necessary.

Example

```
[DynamicallyAccessedMembers (DynamicallyAccessedMemberTypes.PublicMethods)]
[return: DynamicallyAccessedMembers(DynamicallyAccessedMemberTypes.PublicConstructors)]
Type TestMethod<TSource>()
{
    // IL2088 Trim analysis: 'TestMethod' method return value does not satisfy
    'DynamicallyAccessedMembersAttribute' requirements. The generic parameter 'TSource' of 'TestMethod' does not
    have matching annotations. The source value must declare at least the same requirements as those declared on
    the target location it is assigned to.
    return typeof(TSource);
}
```

Fixing

See [Fixing Warnings](#) for guidance.

IL2089: Value stored in field 'target field' does not satisfy 'DynamicallyAccessedMembersAttribute' requirements. The generic parameter 'source generic parameter' of 'source method or type' does not have matching annotations. The source value must declare at least the same requirements as those declared on the target location it is assigned to

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

The target location declares some requirements on the type value via its [DynamicallyAccessedMembersAttribute](#). Those requirements must be met by those declared on the source value also via the [DynamicallyAccessedMembersAttribute](#). The source value can declare more requirements than the source if necessary.

Example

```
[DynamicallyAccessedMembers(DynamicallyAccessedMemberTypes.PublicConstructors)]
Type _typeFieldWithRequirements;

void TestMethod<TSource>()
{
    // IL2089 Trim analysis: value stored in field '_typeFieldWithRequirements' does not satisfy
    // 'DynamicallyAccessedMembersAttribute' requirements. The generic parameter 'TSource' of 'TestMethod'
    // does not have matching annotations. The source value must declare at least the same requirements as those declared on
    // the target location it is assigned to.
    _typeFieldWithRequirements = typeof(TSource);
}
```

Fixing

See [Fixing Warnings](#) for guidance.

IL2090: 'this' argument does not satisfy 'DynamicallyAccessedMembersAttribute' in call to 'target method'. The generic parameter 'source generic parameter' of 'source method or type' does not have matching annotations. The source value must declare at least the same requirements as those declared on the target location it is assigned to

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

The target location declares some requirements on the type value via its [DynamicallyAccessedMembersAttribute](#). Those requirements must be met by those declared on the source value also via the [DynamicallyAccessedMembersAttribute](#). The source value can declare more requirements than the source if necessary.

Example

```
void TestMethod<TSource>()
{
    // IL2090 Trim analysis: 'this' argument does not satisfy 'DynamicallyAccessedMembersAttribute' in call
    // to 'GetMethod'. The generic parameter 'TSource' of 'TestMethod' does not have matching annotations. The
    // source value must declare at least the same requirements as those declared on the target location it is
    // assigned to.
    typeof(TSource).GetMethod(); // Type.GetMethods is annotated with
    DynamicallyAccessedMemberTypes.PublicMethods
}
```

Fixing

See [Fixing Warnings](#) for guidance.

IL2091: 'target generic parameter' generic argument does not satisfy

'DynamicallyAccessedMembersAttribute' in 'target method or type'. The generic parameter 'source target parameter' of 'source method or type' does not have matching annotations. The source value must declare at least the same requirements as those declared on the target location it is assigned to

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

The target location declares some requirements on the type value via its [DynamicallyAccessedMembersAttribute](#). Those requirements must be met by those declared on the source value also via the [DynamicallyAccessedMembersAttribute](#). The source value can declare more requirements than the source if necessary.

Example

```
void NeedsPublicConstructors<[DynamicallyAccessedMembers(DynamicallyAccessedMemberTypes.PublicConstructors)]>(TTarget>()
{
    // ...
}

void TestMethod<TSource>()
{
    // IL2091 Trim analysis: 'TTarget' generic argument does not satisfy
    // 'DynamicallyAccessedMembersAttribute' in 'NeedsPublicConstructors'. The generic parameter 'TSource' of
    // 'TestMethod' does not have matching annotations. The source value must declare at least the same
    // requirements as those declared on the target location it is assigned to.
    NeedsPublicConstructors<TSource>();
}
```

Fixing

See [Fixing Warnings](#) for guidance.

IL2092: The 'DynamicallyAccessedMemberTypes' value used in a 'DynamicallyAccessedMembersAttribute' annotation on a method's parameter does not match the 'DynamicallyAccessedMemberTypes' value of the overridden parameter annotation. All overridden members must have the same attribute's usage

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

All overrides of a virtual method, including the base method, must have the same [DynamicallyAccessedMembersAttribute](#) usage on all their components (return value, parameters, and generic parameters).

Example

```
public class Base
{
    public virtual void TestMethod([DynamicallyAccessedMembers(DynamicallyAccessedMemberTypes.PublicMethods)]
Type type) {}

public class Derived : Base
{
    // IL2092: 'DynamicallyAccessedMemberTypes' in 'DynamicallyAccessedMembersAttribute' on the parameter
    // 'type' of method 'Derived.TestMethod' don't match overridden parameter 'type' of method 'Base.TestMethod'.
    All overridden members must have the same 'DynamicallyAccessedMembersAttribute' usage.
    public override void TestMethod([DynamicallyAccessedMembers(DynamicallyAccessedMemberTypes.PublicFields)]
Type type) {}
}
```

IL2093: The 'DynamicallyAccessedMemberTypes' value used in a 'DynamicallyAccessedMembersAttribute' annotation on a method's return type does not match the 'DynamicallyAccessedMemberTypes' value of the overridden return type annotation. All overridden members must have the same attribute's usage

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

All overrides of a virtual method including the base method must have the same [DynamicallyAccessedMembersAttribute](#) usage on all its components (return value, parameters and generic parameters).

Example

```
public class Base
{
    [return: DynamicallyAccessedMembers(DynamicallyAccessedMemberTypes.PublicMethods)]
    public virtual Type TestMethod() {}

public class Derived : Base
{
    // IL2093: 'DynamicallyAccessedMemberTypes' in 'DynamicallyAccessedMembersAttribute' on the return value
    // of method 'Derived.TestMethod' don't match overridden return value of method 'Base.TestMethod'. All
    // overridden members must have the same 'DynamicallyAccessedMembersAttribute' usage.
    [return: DynamicallyAccessedMembers(DynamicallyAccessedMemberTypes.PublicFields)]
    public override Type TestMethod() {}
}
```

IL2094: The 'DynamicallyAccessedMemberTypes' value used in a 'DynamicallyAccessedMembersAttribute' annotation on a method's implicit `this` parameter does not match the 'DynamicallyAccessedMemberTypes' value of the overridden `this` parameter annotation. All overridden members must have the same attribute's usage

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

All overrides of a virtual method including the base method must have the same `DynamicallyAccessedMembersAttribute` usage on all its components (return value, parameters and generic parameters).

Example

```
// This only works on methods in System.Type and derived classes - this is just an example
public class Type
{
    [DynamicallyAccessedMembers(DynamicallyAccessedMemberTypes.PublicMethods)]
    public virtual void TestMethod() {}

    public class DerivedType : Type
    {
        // IL2094: 'DynamicallyAccessedMemberTypes' in 'DynamicallyAccessedMembersAttribute' on the implicit 'this' parameter of method 'DerivedType.TestMethod' don't match overridden implicit 'this' parameter of method 'Type.TestMethod'. All overridden members must have the same 'DynamicallyAccessedMembersAttribute' usage.
        [DynamicallyAccessedMembers(DynamicallyAccessedMemberTypes.PublicFields)]
        public override void TestMethod() {}
    }
}
```

IL2095: The 'DynamicallyAccessedMemberTypes' value used in a 'DynamicallyAccessedMembersAttribute' annotation on a method's generic parameter does not match the 'DynamicallyAccessedMemberTypes' value of the overridden generic parameter annotation. All overridden members must have the same attribute's usage

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

All overrides of a virtual method including the base method must have the same [DynamicallyAccessedMembersAttribute](#) usage on all its components (return value, parameters and generic parameters).

Example

```
public class Base
{
    public virtual void TestMethod<[DynamicallyAccessedMembers(DynamicallyAccessedMemberTypes.PublicMethods)]
T>() {}
}

public class Derived : Base
{
    // IL2095: 'DynamicallyAccessedMemberTypes' in 'DynamicallyAccessedMembersAttribute' on the generic
    // parameter 'T' of method 'Derived.TestMethod<T>' don't match overridden generic parameter 'T' of method
    // 'Base.TestMethod<T>'. All overridden members must have the same 'DynamicallyAccessedMembersAttribute' usage.
    public override void TestMethod<[DynamicallyAccessedMembers(DynamicallyAccessedMemberTypes.PublicFields)]
T>() {}
}
```

IL2096: Call to
'Type.GetType(System.String, System.Boolean, System.Boolean)'
can perform case insensitive lookup of the type. Currently,
Trimmer cannot guarantee presence of all the matching types"

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

Specifying a case-insensitive search on an overload of `GetType(String, Boolean, Boolean)` is not supported by Trimmer. Specify `false` to perform a case-sensitive search or use an overload that does not use an `ignoreCase` boolean.

Example

```
void TestMethod()
{
    // IL2096 Trim analysis: Call to 'System.Type.GetType(String,Boolean,Boolean)' can perform case
    // insensitive lookup of the type, currently ILLink can not guarantee presence of all the matching types
    Type.GetType ("typeName", false, true);
}
```

IL2097: Field annotated with 'DynamicallyAccessedMembersAttribute' is not of type 'System.Type', 'System.String', or derived

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

[DynamicallyAccessedMembersAttribute](#) is only applicable to items of type [Type](#), [String](#), or derived. On all other types the attribute will be ignored. Using the attribute on any other type is likely incorrect and unintentional.

Example

```
// IL2097: Field '_valueField' has 'DynamicallyAccessedMembersAttribute', but that attribute can only be applied to fields of type 'System.Type' or 'System.String'  
[DynamicallyAccessedMembers(DynamicallyAccessedMemberTypes.PublicMethods)]  
object _valueField;  
}
```

IL2098: Method's parameter annotated with 'DynamicallyAccessedMembersAttribute' is not of type 'System.Type', 'System.String', or derived

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

[DynamicallyAccessedMembersAttribute](#) is only applicable to items of type [Type](#), [String](#), or derived. On all other types the attribute will be ignored. Using the attribute on any other type is likely incorrect and unintentional.

Example

```
// IL2098: Parameter 'param' of method 'TestMethod' has 'DynamicallyAccessedMembersAttribute', but that
// attribute can only be applied to parameters of type 'System.Type' or 'System.String'
void TestMethod([DynamicallyAccessedMembers(DynamicallyAccessedMemberTypes.PublicMethods)] object param)
{
    // param.GetType()....
}
```

IL2099: Property annotated with 'DynamicallyAccessedMembersAttribute' is not of type 'System.Type;; 'System.String', or derived

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

[DynamicallyAccessedMembersAttribute](#) is only applicable to items of type [Type](#), [String](#), or derived. On all other types the attribute will be ignored. Using the attribute on any other type is likely incorrect and unintentional.

Example

```
// IL2099: Parameter 'param' of method 'TestMethod' has 'DynamicallyAccessedMembersAttribute', but that
attribute can only be applied to properties of type 'System.Type' or 'System.String'
[DynamicallyAccessedMembers(DynamicallyAccessedMemberTypes.PublicMethods)]
object TestProperty { get; set; }
```

IL2100: Trimmer XML contains unsupported wildcard on argument `fullname` of an assembly element

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

A wildcard cannot be the value of the `fullname` argument for an assembly element in a Trimmer XML. Use a specific assembly name instead.

Example

```
<!-- IL2100: XML contains unsupported wildcard for assembly "fullname" attribute -->
<linker>
    <assembly fullname="*>
        <type fullname="MyType" />
    </assembly>
</linker>
```

IL2101: Assembly's embedded XML references a different assembly in `fullname` argument of an assembly element

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

Embedded attribute or substitution XML may only contain elements that apply to the containing assembly. Attempting to modify another assembly will not have any effect.

Example

```
<!-- IL2101: Embedded XML in assembly 'ContainingAssembly' contains assembly "fullname" attribute for  
another assembly 'OtherAssembly' -->  
<linker>  
  <assembly fullname="OtherAssembly">  
    <type fullname="MyType" />  
  </assembly>  
</linker>
```

IL2102: Invalid 'Reflection.AssemblyMetadataAttribute' attribute found in assembly. Value must be `True`

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

`AssemblyMetadataAttribute` may be used at the assembly level to turn on trimming for the assembly. The attribute contains an unsupported value. The only supported value is `True`.

Example

```
<!-- IL2102: Embedded XML in assembly 'ContainingAssembly' contains assembly "fullname" attribute for  
another assembly 'OtherAssembly' -->  
<linker>  
  <assembly fullname="OtherAssembly">  
    <type fullname="MyType" />  
  </assembly>  
</linker>
```

IL2103: Value passed to the 'propertyAccessor' parameter of method

'System.Linq.Expressions.Expression.Property(Expression, MethodInfo)' cannot be statically determined as a property accessor

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

The value passed to the `propertyAccessor` parameter of `Property(Expression, MethodInfo)` was not recognized as a property accessor method. Trimmer can't guarantee the presence of the property.

Example

```
void TestMethod(MethodInfo methodInfo)
{
    // IL2103: Value passed to the 'propertyAccessor' parameter of method
    // 'System.Linq.Expressions.Expression.Property(Expression, MethodInfo)' cannot be statically determined as a
    // property accessor.
    Expression.Property(null, methodInfo);
}
```

IL2104: Assembly produced trim warnings

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

The assembly 'assembly' produced trim analysis warnings in the context of the app. This means the assembly has not been fully annotated for trimming. Consider contacting the library author to request they add trim annotations to the library. To see detailed warnings for this assembly, turn off grouped warnings by passing either `--singlewarn-` `ILLink` command line option to show detailed warnings for all assemblies, or `--singlewarn- "assembly"` to show detailed warnings for that assembly. For more information on annotating libraries for trimming, see [Prepare .NET libraries for trimming](#).

IL2105: Type 'type' was not found in the caller assembly nor in the base library. Type name strings used for dynamically accessing a type should be assembly qualified

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

Type name strings representing dynamically accessed types must be assembly qualified. Otherwise, linker will first search for the type name in the caller's assembly and then in *System.Private.CoreLib*. If the linker fails to resolve the type name, `null` is returned.

Example

```
void TestInvalidTypeName()
{
    RequirePublicMethodOnAType("Foo.Unqualified.TypeName");
}

void RequirePublicMethodOnAType(
    [DynamicallyAccessedMembers(DynamicallyAccessedMemberTypes.PublicMethods)]
    string typeName)
{}
```

IL2106: Return type of method 'method' has 'DynamicallyAccessedMembersAttribute', but that attribute can only be applied to properties of type 'System.Type' or 'System.String'

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

[DynamicallyAccessedMembersAttribute](#) is only applicable to items of type [Type](#) or [String](#) (or derived). On all other types, the attribute will be ignored. Using the attribute on any other type is likely incorrect and unintentional.

Example

```
// IL2106: Return type of method 'TestMethod' has 'DynamicallyAccessedMembersAttribute', but that attribute
// can only be applied to properties of type 'System.Type' or 'System.String'
[DynamicallyAccessedMembers(DynamicallyAccessedMemberTypes.PublicMethods)] object TestMethod()
{
    return typeof(TestType);
}
```

IL2107: Methods 'method1' and 'method2' are both associated with state machine type 'type'. This is currently unsupported and may lead to incorrectly reported warnings

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

The trimmer cannot correctly handle if the same compiler-generated state machine type is associated (via the state-machine attributes) with two different methods. The trimmer derives warning suppressions from the method that produced the state machine and does not support reprocessing the same method or type more than once.

Example

```
class <compiler_generated_state_machine>_type {
    void MoveNext()
    {
        // This should normally produce IL2026
        CallSomethingWhichRequiresUnreferencedCode ();
    }
}

[RequiresUnreferencedCode ("")]
[IteratorStateMachine(typeof(<compiler_generated_state_machine>_type))]
IEnumerable<int> UserDefinedMethod()
{
    // Uses the state machine type
    // The IL2026 from the state machine should be suppressed in this case
}

// IL2107: Methods 'UserDefinedMethod' and 'SecondUserDefinedMethod' are both associated with state machine
// type '<compiler_generated_state_machine>_type'.
[IteratorStateMachine(typeof(<compiler_generated_state_machine>_type))]
IEnumerable<int> SecondUserDefinedMethod()
{
    // Uses the state machine type
    // The IL2026 from the state should be reported in this case
}
```

IL2108: Invalid scope 'scope' used in 'UnconditionalSuppressMessageAttribute' on module 'module' with target 'target'

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

The only scopes supported on global unconditional suppressions are `module`, `type`, and `member`. If the scope and target arguments are null or missing on a global suppression, it's assumed that the suppression is put on the module. Global unconditional suppressions using invalid scopes are ignored.

Example

```
// IL2108: Invalid scope 'method' used in 'UnconditionalSuppressMessageAttribute' on module 'Warning' with
// target 'MyTarget'.
[module: UnconditionalSuppressMessage ("Test suppression with invalid scope", "IL2026", Scope = "method",
Target = "MyTarget")]

class Warning
{
    static void Main(string[] args)
    {
        Foo();
    }

    [RequiresUnreferencedCode("Warn when Foo() is called")]
    static void Foo()
    {
    }
}
```

IL2109: Type derives from base type that has 'RequiresUnreferencedCodeAttribute'

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

A type is referenced in code, and this type derives from a base type with [RequiresUnreferencedCodeAttribute](#), which can break functionality of a trimmed application. Types that derive from a base class with [RequiresUnreferencedCodeAttribute](#) need to explicitly use the [RequiresUnreferencedCodeAttribute](#) or suppress this warning.

Example

```
[RequiresUnreferencedCode("Using any of the members inside this class is trim unsafe",
Url="http://help/unreferencedcode")]
public class UnsafeClass {
    public UnsafeClass () {}
    public static void UnsafeMethod();
}

// IL2109: Type 'Derived' derives from 'UnsafeClass' which has 'RequiresUnreferencedCodeAttribute'. Using
any of the members inside this class is trim unsafe. http://help/unreferencedcode
class Derived : UnsafeClass {}
```

IL2110: Field with 'DynamicallyAccessedMembersAttribute' is accessed via reflection. Trimmer cannot guarantee availability of the requirements of the field

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

The trimmer can't guarantee that all requirements of the `DynamicallyAccessedMembersAttribute` are fulfilled if the field is accessed via reflection.

Example

```
[DynamicallyAccessedMembers(DynamicallyAccessedMemberTypes.PublicMethods)]
Type _field;

void TestMethod()
{
    // IL2110: Field '_field' with 'DynamicallyAccessedMembersAttribute' is accessed via reflection. Trimmer
    can't guarantee availability of the requirements of the field.
    typeof(Test).GetField("_field");
}
```

IL2111: Method with parameters or return value with 'DynamicallyAccessedMembersAttribute' is accessed via reflection. Trimmer cannot guarantee availability of the requirements of the method

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

The trimmer can't guarantee that all requirements of the `DynamicallyAccessedMembersAttribute` are fulfilled if the method is accessed via reflection.

Example

```
void MethodWithRequirements([DynamicallyAccessedMembers(DynamicallyAccessedMemberTypes.PublicMethods)] Type
type)
{
}

void TestMethod()
{
    // IL2111: Method 'MethodWithRequirements' with parameters or return value with
    `DynamicallyAccessedMembersAttribute` is accessed via reflection. Trimmer can't guarantee availability of
    the requirements of the method.
    typeof(Test).GetMethod("MethodWithRequirements");
}
```

IL2112: 'DynamicallyAccessedMembersAttribute' on 'type' or one of its base types references 'member', which requires unreferenced code

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

A type is annotated with [DynamicallyAccessedMembersAttribute](#) indicating that the type may dynamically access some members declared on the type or its derived types. This instructs the trimmer to keep the specified members, but one of them is annotated with [RequiresUnreferencedCodeAttribute](#), which can break functionality when trimming. The [DynamicallyAccessedMembersAttribute](#) annotation may be directly on the type, or implied by an annotation on one of its base or interface types. This warning originates from the member with [RequiresUnreferencedCodeAttribute](#).

Example

```
[DynamicallyAccessedMembers(DynamicallyAccessedMemberTypes.PublicMethods)]
public class AnnotatedType {
    // Trim analysis warning IL2112: AnnotatedType.Method(): 'DynamicallyAccessedMembersAttribute' on
    // 'AnnotatedType' or one of its
    // base types references 'AnnotatedType.Method()' which requires unreferenced code. Using this member is
    // trim unsafe.
    [RequiresUnreferencedCode("Using this member is trim unsafe")]
    public static void Method() { }
}
```

IL2113: 'DynamicallyAccessedMembersAttribute' on 'type' or one of its base types references 'member', which requires unreferenced code

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

A type is annotated with [RequiresUnreferencedCodeAttribute](#) indicating that the type may dynamically access some members declared on one of its derived types. This instructs the trimmer to keep the specified members, but a member of one of the base or interface types is annotated with [RequiresUnreferencedCodeAttribute](#), which can break functionality when trimming. The [RequiresUnreferencedCodeAttribute](#) annotation may be directly on the type, or implied by an annotation on one of its base or interface types. This warning originates from the type which has [RequiresUnreferencedCodeAttribute](#) requirements.

Example

```
public class BaseType {
    [RequiresUnreferencedCode("Using this member is trim unsafe")]
    public static void Method() { }
}

// Trim analysis warning IL2113: AnnotatedType: 'DynamicallyAccessedMembersAttribute' on 'AnnotatedType' or
// one of its
// base types references 'BaseType.Method()' which requires unreferenced code. Using this member is trim
// unsafe.
[DynamicallyAccessedMembers(DynamicallyAccessedMemberTypes.PublicMethods)]
public class AnnotatedType : BaseType {
}
```

IL2114: 'DynamicallyAccessedMembersAttribute' on 'type' or one of its base types references 'member' which has 'DynamicallyAccessedMembersAttribute' requirements

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

A type is annotated with [RequiresUnreferencedCodeAttribute](#) indicating that the type may dynamically access some members declared on the type or its derived types. This instructs the trimmer to keep the specified members, but one of them is annotated with [RequiresUnreferencedCodeAttribute](#) which can not be statically verified. The [RequiresUnreferencedCodeAttribute](#) annotation may be directly on the type, or implied by an annotation on one of its base or interface types. This warning originates from the member with [RequiresUnreferencedCodeAttribute](#) requirements.

Example

```
[DynamicallyAccessedMembers(DynamicallyAccessedMemberTypes.PublicFields)]
public class AnnotatedType {
    // Trim analysis warning IL2114: System.Type AnnotatedType::Field: 'DynamicallyAccessedMembersAttribute'
    // on 'AnnotatedType' or one of its
    // base types references 'System.Type AnnotatedType::Field' which has
    'DynamicallyAccessedMembersAttribute' requirements .
    [DynamicallyAccessedMembers(DynamicallyAccessedMemberTypes.PublicProperties)]
    public static Type Field;
}
```

IL2115: 'DynamicallyAccessedMembersAttribute' on 'type' or one of its base types references 'member' which has 'DynamicallyAccessedMembersAttribute' requirements

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

A type is annotated with [DynamicallyAccessedMembersAttribute](#) indicating that the type may dynamically access some members declared on one of the derived types. This instructs the trimmer to keep the specified members, but a member of one of the base or interface types is annotated with [DynamicallyAccessedMembersAttribute](#) which cannot be statically verified. The [DynamicallyAccessedMembersAttribute](#) annotation may be directly on the type, or implied by an annotation on one of its base or interface types. This warning originates from the type which has [DynamicallyAccessedMembersAttribute](#) requirements.

Example

```
public class BaseType {
    [DynamicallyAccessedMembers(DynamicallyAccessedMemberTypes.PublicProperties)]
    public static Type Field;
}

// Trim analysis warning IL2115: AnnotatedType: 'DynamicallyAccessedMembersAttribute' on 'AnnotatedType' or
// one of its
// base types references 'System.Type BaseType::Field' which has 'DynamicallyAccessedMembersAttribute'
// requirements .
[DynamicallyAccessedMembers(DynamicallyAccessedMemberTypes.PublicFields)]
public class AnnotatedType : BaseType {
```

IL2116: 'RequiresUnreferencedCodeAttribute' cannot be placed directly on a static constructor. Consider placing it on the type declaration instead

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

`RequiresUnreferencedCodeAttribute` is not allowed on static constructors since these are not callable by the user.

Placing the attribute directly on a static constructor will have no effect. Annotate the method's containing type instead.

Example

```
public class MyClass {  
    // Trim analysis warning IL2116: 'RequiresUnreferencedCodeAttribute' cannot be placed directly on static  
    // constructor 'MyClass..cctor()', consider placing 'RequiresUnreferencedCodeAttribute' on the type declaration  
    // instead.  
    [RequiresUnreferencedCode("Dangerous")]  
    static MyClass () { }  
}
```

Native AOT Deployment

9/20/2022 • 3 minutes to read • [Edit Online](#)

Publishing your app as *native AOT* produces an app that is [self-contained](#) and that has been ahead-of-time (AOT) compiled to native code. Native AOT apps start up very quickly and use less memory. Users of the application can run it on a machine that doesn't have the .NET runtime installed.

The benefit of native AOT is most significant for workloads with a high number of deployed instances, such as cloud infrastructure and hyper-scale services. It is currently not supported with ASP.NET Core, but only console apps.

The native AOT deployment model uses an ahead of time compiler to compile IL to native code at the time of publish. Native AOT apps don't use a Just-In-Time (JIT) compiler when the application runs. Native AOT apps can run in restricted environments where a JIT is not allowed. Native AOT applications target a specific runtime environment, such as Linux x64 or Windows x64, just like publishing a [self-contained app](#).

There are some limitations in the .NET native AOT deployment model, with the main one being that run-time code generation is not possible. For more information, see [Limitations of Native AOT deployment](#). The support in the .NET 7 release is targeted towards console-type applications.

WARNING

Native AOT is supported in .NET 7 but only a limited number of libraries are fully compatible with native AOT in .NET 7.

Prerequisites

The following prerequisites need to be installed before publishing .NET projects with native AOT.

On Windows, install [Visual Studio 2022](#), including Desktop development with C++ workload.

On Linux, install clang and developer packages for libraries that .NET runtime depends on.

- Ubuntu (18.04+)

```
sudo apt-get install clang zlib1g-dev
```

- Alpine (3.15+)

```
sudo apk add clang gcc lld musl-dev build-base zlib-dev
```

Publish Native AOT - CLI

1. Add `<PublishAot>true</PublishAot>` to your project file.

This will enable native AOT compilation during publish. It will also enable dynamic code usage analysis during build and editing. Prefer placing this setting in the project file to passing it on the command line since it controls behaviors outside publish.

```
<PropertyGroup>
  <PublishAot>true</PublishAot>
</PropertyGroup>
```

2. Publish the app for a specific runtime identifier using `dotnet publish -r <RID>`.

The following example publishes the app for Windows as a native AOT application on a machine with the required prerequisites installed.

```
dotnet publish -r win-x64 -c Release
```

The following example publishes the app for Linux as a native AOT application. A native AOT binary produced on Linux machine is only going to work on same or newer Linux version. For example, native AOT binary produced on Ubuntu 20.04 is going to run on Ubuntu 20.04 and later, but it is not going to run on Ubuntu 18.04.

```
dotnet publish -r linux-arm64 -c Release
```

The app will be available in the publish directory and will contain all the code needed to run in it, including a stripped-down version of the coreclr runtime.

Check out the [native AOT samples](#) available in the dotnet/samples repository on GitHub. The samples include [Linux](#) and [Windows](#) Dockerfiles that demonstrate how to automate installation of prerequisites and publishing .NET projects with native AOT using containers.

Native Debug Information

The native AOT publishing follows platform conventions for native toolchains. The default behavior of native toolchains on Windows is to produce debug information in a separate `.pdb` file. The default behavior of native toolchains on Linux is to include the debug information in the native binary which makes the native binary significantly larger.

Set the `StripSymbols` property to `true` to produce the debug information in a separate `.dbg` file and exclude it from the native binary on Linux. This property has no effect on Windows.

```
<PropertyGroup>
  <StripSymbols>true</StripSymbols>
</PropertyGroup>
```

Limitations of Native AOT deployment

Native AOT applications come with a few fundamental limitations and compatibility issues. The key limitations include:

- No dynamic loading (for example, `Assembly.LoadFile()`)
- No runtime code generation (for example, `System.Reflection.Emit`)
- No C++/CLI
- No built-in COM (only applies to Windows)
- Requires trimming, which has [limitations](#)
- Implies compilation into a single file, which has known [incompatibilities](#)
- Apps include required runtime libraries (just like [self-contained apps](#), increasing their size, as compared to framework-dependent apps)

The publish process will analyze the entire project and its dependencies and produce warnings whenever the limitations could potentially be hit by the published application at runtime.

The first release of native AOT in .NET 7 has additional limitations. These include:

- Should be targeted for console type applications (not ASP.NET Core).
- Not all the runtime libraries are fully annotated to be native AOT compatible (that is, some warnings in the runtime libraries are not actionable by end developers).
- Limited diagnostic support (debugging and profiling).

Platform/architecture restrictions

The following table shows supported compilation targets when targeting .NET 7.

PLATFORM	SUPPORTED ARCHITECTURE
Windows	x64, Arm64
Linux	x64, Arm64

IL3050: Avoid calling members annotated with 'RequiresDynamicCodeAttribute' when publishing as native AOT

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

When publishing an app as native AOT (by setting the `PublishAot` property in a project to `true`), calling members annotated with the `RequiresDynamicCodeAttribute` attribute may result in exceptions at run time. Members annotated with this attribute may require ability to dynamically create new code at run time, and native AOT publishing model doesn't provide a way to generate native code at run time.

Rule description

`RequiresDynamicCodeAttribute` indicates that the member references code that may require code generation at runtime.

Example

```
// AOT analysis warning IL3050: Program.<Main>$({String[]}): Using member
'System.Type.MakeGenericType(Type[])'
// which has 'RequiresDynamicCodeAttribute' can break functionality when AOT compiling. The native code for
// this instantiation might not be available at runtime.
typeof(Generic<>).MakeGenericType(typeof(SomeStruct));

class Generic<T> { }

struct SomeStruct { }
```

How to fix violations

Members annotated with the `RequiresDynamicCodeAttribute` attribute have a message that provides useful information to users who are publishing as native AOT. Consider adapting existing code to the attribute's message or removing the violating call.

IL3051: 'RequiresDynamicCodeAttribute' attribute must be consistently applied on virtual and interface methods

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

There is a mismatch in the `RequiresDynamicCodeAttribute` annotations between an interface and its implementation or a virtual method and its override.

Example

A base member has the attribute but the derived member does not have the attribute.

```
public class Base
{
    [RequiresDynamicCode("Message")]
    public virtual void TestMethod() {}

public class Derived : Base
{
    // IL3051: Base member 'Base.TestMethod' with 'RequiresDynamicCodeAttribute' has a derived member
    // 'Derived.TestMethod()' without 'RequiresDynamicCodeAttribute'. For all interfaces and overrides the
    // implementation attribute must match the definition attribute.
    public override void TestMethod() {}
}
```

A derived member has the attribute but the overridden base member does not have the attribute.

```
public class Base
{
    public virtual void TestMethod() {}

public class Derived : Base
{
    // IL3051: Member 'Derived.TestMethod()' with 'RequiresDynamicCodeAttribute' overrides base member
    // 'Base.TestMethod()' without 'RequiresDynamicCodeAttribute'. For all interfaces and overrides the
    // implementation attribute must match the definition attribute.
    [RequiresDynamicCode("Message")]
    public override void TestMethod() {}
}
```

An interface member has the attribute but its implementation does not have the attribute.

```
interface IRDC
{
    [RequiresDynamicCode("Message")]
    void TestMethod();
}

class Implementation : IRDC
{
    // IL3051: Interface member 'IRDC.TestMethod()' with 'RequiresDynamicCodeAttribute' has an implementation
    // member 'Implementation.TestMethod()' without 'RequiresDynamicCodeAttribute'. For all interfaces and
    // overrides the implementation attribute must match the definition attribute.
    public void TestMethod () { }
}
```

An implementation member has the attribute but the interface that it implements does not have the attribute.

```
interface IRDC
{
    void TestMethod();
}

class Implementation : IRDC
{
    [RequiresDynamicCode("Message")]
    // IL3051: Member 'Implementation.TestMethod()' with 'RequiresDynamicCodeAttribute' implements interface
    // member 'IRDC.TestMethod()' without 'RequiresDynamicCodeAttribute'. For all interfaces and overrides the
    // implementation attribute must match the definition attribute.
    public void TestMethod () { }
}
```

IL3052: COM interop is not supported with full ahead of time compilation

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

Built-in COM is not supported with native AOT compilation. Use [COM wrappers](#) instead.

When the unsupported code path is reached at run time, an exception will be thrown.

Example

```
using System.Runtime.InteropServices;

// AOT analysis warning IL3052: CorRuntimeHost.CorRuntimeHost(): COM interop is not supported
// with full ahead of time compilation.
new CorRuntimeHost();

[Guid("CB2F6723-AB3A-11D2-9C40-00C04FA30A3E")]
[ComImport]
[ClassInterface(ClassInterfaceType.None)]
public class CorRuntimeHost
{
}
```

IL3053: Assembly produced AOT warnings

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

The assembly produced one or more AOT analysis warnings. The warnings have been collapsed into a single warning message because they refer to code that likely comes from a third party and is not directly actionable. Using the library with native AOT might be problematic.

To see the detailed warnings, specify `<TrimmerSingleWarn>false</TrimmerSingleWarn>` in your project file.

IL3054: Generic expansion to a method or type was aborted due to generic recursion

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

Methods on generic types and generic methods that are instantiated over different types are supported by different native code bodies specialized for the given type parameter.

It is possible to form a cycle between generic instantiations in a way that the number of native code bodies required to support the application becomes unbounded. Because native AOT deployments require generating all native method bodies at the time of publishing the application, this would require compiling an infinite number of methods.

When the AOT compilation process detects such unbounded growth, it cuts off the growth by generating a throwing method. If the application goes beyond the cutoff point at run time, an exception is thrown.

Even though it's unlikely the throwing method body will be reached at run time, it's advisable to remove the generic recursion by restructuring the code. Generic recursion negatively affects compilation speed and the size of the output executable.

Example

The following program will work correctly for input "2" but throws an exception for input "100".

```
// AOT analysis warning IL3054:  
// Program.<<Main>$>g__CauseGenericRecursion|0_0<Struct`1<Struct`1<Struct`1<Struct`1<Int32>>>>(Int32):  
// Generic expansion to 'Program'.  
<<Main>$>g__CauseGenericRecursion|0_0<Struct`1<Struct`1<Struct`1<Struct`1<Struct`1<Int32>>>>(Int32)'  
// was aborted due to generic recursion. An exception will be thrown at runtime if this codepath  
// is ever reached. Generic recursion also negatively affects compilation speed and the size of  
// the compilation output. It is advisable to remove the source of the generic recursion  
// by restructuring the program around the source of recursion. The source of generic recursion  
// might include: 'Program.<<Main>$>g__CauseGenericRecursion|0_0<T>(Int32)  
  
using System;  
  
int number = int.Parse(Console.ReadLine());  
Console.WriteLine(CauseGenericRecursion<int>(number));  
  
static int CauseGenericRecursion<T>(int i) => 1 + CauseGenericRecursion<Struct<T>>(i - 1);  
  
struct Struct<T> { }
```

The behavior of the program at run time for input "100":

```
Unhandled Exception: System.TypeLoadException: Could not load type 'Program' from assembly 'repro, Version=7.0.0.0, Culture=neutral, PublicKeyToken=null'.
  at Internal.Runtime.CompilerHelpers.ThrowHelpers.ThrowTypeLoadExceptionWithArgument(ExceptionStringID, String, String, String) + 0x42
  at Program.<<Main>$>g__CauseGenericRecursion|0_0[T](Int32) + 0x29
  at Program.<<Main>$>g__CauseGenericRecursion|0_0[T](Int32) + 0x1f
  at Program.<<Main>$>g__CauseGenericRecursion|0_0[T](Int32) + 0x1f
  at Program.<<Main>$>g__CauseGenericRecursion|0_0[T](Int32) + 0x1f
  at Program.<<Main>$>g__CauseGenericRecursion|0_0[T](Int32) + 0x1f
  at Program.<<Main>$>g__CauseGenericRecursion|0_0[T](Int32) + 0x3a
```

IL3055: P/Invoke method declares a parameter with an abstract delegate

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

P/Invoke marshalling code needs to be generated ahead of time. If marshalling code for a delegate wasn't pregenerated, P/Invoke marshalling will fail with an exception at run time.

Marshalling code is generated for delegate types that either:

- Are used in signatures of P/Invoke methods.
- Appear as fields of types passed to native code via P/Invoke.
- Are decorated with [UnmanagedFunctionPointerAttribute](#).

If a concrete type cannot be inferred from the P/Invoke signature, marshalling code might not be available at run time and the P/Invoke will throw an exception.

Replace [Delegate](#) or [MulticastDelegate](#) in the P/Invoke signature with a concrete delegate type.

Example

```
using System;
using System.Runtime.InteropServices;

PinvokeMethod(() => { });

// AOT analysis warning IL3055: Program.<Main>$<Main>$(String[]): P/Invoke method
// 'Program.<Main>$>g__PinvokeMethod|0_1(Delegate)' declares a parameter with an abstract delegate.
// Correctness of interop for abstract delegates cannot be guaranteed after native compilation:
// the marshalling code for the delegate might not be available. Use a non-abstract delegate type
// or ensure any delegate instance passed as parameter is marked with `UnmanagedFunctionPointerAttribute`.
[DllImport("library")]
static extern void PinvokeMethod(Delegate del);
```

IL3056: `RequiresDynamicCodeAttribute` cannot be placed directly on a static constructor

9/20/2022 • 2 minutes to read • [Edit Online](#)

Cause

`RequiresDynamicCodeAttribute` is not allowed on static constructors since these are not callable by the user. Placing the attribute directly on a static constructor will have no effect. Annotate the method's containing type instead.

Example

```
public class MyClass {  
    // Trim analysis warning IL2116: 'RequiresDynamicCodeAttribute' cannot be placed directly on static  
    // constructor 'MyClass..cctor()', consider placing 'RequiresDynamicCodeAttribute' on the type declaration  
    // instead.  
    [RequiresDynamicCode("Dangerous")]  
    static MyClass () { }  
}
```

Runtime package store

9/20/2022 • 5 minutes to read • [Edit Online](#)

Starting with .NET Core 2.0, it's possible to package and deploy apps against a known set of packages that exist in the target environment. The benefits are faster deployments, lower disk space usage, and improved startup performance in some cases.

This feature is implemented as a *runtime package store*, which is a directory on disk where packages are stored (typically at `/usr/local/share/dotnet/store` on macOS/Linux and `C:/Program Files/dotnet/store` on Windows). Under this directory, there are subdirectories for architectures and **target frameworks**. The file layout is similar to the way that [NuGet assets are laid out on disk](#):

```
\dotnet
  \store
    \x64
      \netcoreapp2.0
        \microsoft.applicationinsights
        \microsoft.aspnetcore
        ...
    \x86
      \netcoreapp2.0
        \microsoft.applicationinsights
        \microsoft.aspnetcore
        ...
```

A *target manifest* file lists the packages in the runtime package store. Developers can target this manifest when publishing their app. The target manifest is typically provided by the owner of the targeted production environment.

Preparing a runtime environment

The administrator of a runtime environment can optimize apps for faster deployments and lower disk space use by building a runtime package store and the corresponding target manifest.

The first step is to create a *package store manifest* that lists the packages that compose the runtime package store. This file format is compatible with the project file format (*csproj*).

```
<Project Sdk="Microsoft.NET.Sdk">
  <ItemGroup>
    <PackageReference Include="NUGET_PACKAGE" Version="VERSION" />
    <!-- Include additional packages here -->
  </ItemGroup>
</Project>
```

Example

The following example package store manifest (*packages.csproj*) is used to add `Newtonsoft.Json` and `Moq` to a runtime package store:

```
<Project Sdk="Microsoft.NET.Sdk">
<ItemGroup>
<PackageReference Include="Newtonsoft.Json" Version="10.0.3" />
<PackageReference Include="Moq" Version="4.7.63" />
</ItemGroup>
</Project>
```

Provision the runtime package store by executing `dotnet store` with the package store manifest, runtime, and framework:

```
dotnet store --manifest <PATH_TO_MANIFEST_FILE> --runtime <RUNTIME_IDENTIFIER> --framework <FRAMEWORK>
```

Example

```
dotnet store --manifest packages.csproj --runtime win10-x64 --framework netcoreapp2.0 --framework-version 2.0.0
```

You can pass multiple target package store manifest paths to a single `dotnet store` command by repeating the option and path in the command.

By default, the output of the command is a package store under the `.dotnet/store` subdirectory of the user's profile. You can specify a different location using the `--output <OUTPUT_DIRECTORY>` option. The root directory of the store contains a target manifest `artifact.xml` file. This file can be made available for download and be used by app authors who want to target this store when publishing.

Example

The following `artifact.xml` file is produced after running the previous example. Note that `Castle.Core` is a dependency of `Moq`, so it's included automatically and appears in the `artifacts.xml` manifest file.

```
<StoreArtifacts>
<Package Id="Newtonsoft.Json" Version="10.0.3" />
<Package Id="Castle.Core" Version="4.1.0" />
<Package Id="Moq" Version="4.7.63" />
</StoreArtifacts>
```

Publishing an app against a target manifest

If you have a target manifest file on disk, you specify the path to the file when publishing your app with the `dotnet publish` command:

```
dotnet publish --manifest <PATH_TO_MANIFEST_FILE>
```

Example

```
dotnet publish --manifest manifest.xml
```

You deploy the resulting published app to an environment that has the packages described in the target manifest. Failing to do so results in the app failing to start.

Specify multiple target manifests when publishing an app by repeating the option and path (for example, `--manifest manifest1.xml --manifest manifest2.xml`). When you do so, the app is trimmed for the union of packages specified in the target manifest files provided to the command.

If you deploy an application with a manifest dependency that's present in the deployment (the assembly is present in the `bin` folder), the runtime package store *isn't used* on the host for that assembly. The `bin` folder assembly is used regardless of its presence in the runtime package store on the host.

The version of the dependency indicated in the manifest must match the version of the dependency in the runtime package store. If you have a version mismatch between the dependency in the target manifest and the version that exists in the runtime package store and the app doesn't include the required version of the package in its deployment, the app fails to start. The exception includes the name of the target manifest that called for the runtime package store assembly, which helps you troubleshoot the mismatch.

When the deployment is *trimmed* on publish, only the specific versions of the manifest packages you indicate are withheld from the published output. The packages at the versions indicated must be present on the host for the app to start.

Specifying target manifests in the project file

An alternative to specifying target manifests with the `dotnet publish` command is to specify them in the project file as a semicolon-separated list of paths under a `<TargetManifestFiles>` tag.

```
<PropertyGroup>
  <TargetManifestFiles>manifest1.xml;manifest2.xml</TargetManifestFiles>
</PropertyGroup>
```

Specify the target manifests in the project file only when the target environment for the app is well-known, such as for .NET Core projects. This isn't the case for open-source projects. The users of an open-source project typically deploy it to different production environments. These production environments generally have different sets of packages pre-installed. You can't make assumptions about the target manifest in such environments, so you should use the `--manifest` option of `dotnet publish`.

ASP.NET Core implicit store (.NET Core 2.0 only)

The ASP.NET Core implicit store applies only to ASP.NET Core 2.0. We strongly recommend applications use ASP.NET Core 2.1 and later, which does **not** use the implicit store. ASP.NET Core 2.1 and later use the shared framework.

For .NET Core 2.0, the runtime package store feature is used implicitly by an ASP.NET Core app when the app is deployed as a [framework-dependent deployment](#) app. The targets in `Microsoft.NET.Sdk.Web` include manifests referencing the implicit package store on the target system. Additionally, any framework-dependent app that depends on the `Microsoft.AspNetCore.All` package results in a published app that contains only the app and its assets and not the packages listed in the `Microsoft.AspNetCore.All` metapackage. It's assumed that those packages are present on the target system.

The runtime package store is installed on the host when the .NET SDK is installed. Other installers may provide the runtime package store, including Zip/tarball installations of the .NET SDK, `apt-get`, Red Hat Yum, the .NET Core Windows Server Hosting bundle, and manual runtime package store installations.

When deploying a [framework-dependent deployment](#) app, make sure that the target environment has the .NET SDK installed. If the app is deployed to an environment that doesn't include ASP.NET Core, you can opt out of the implicit store by specifying `<PublishWithAspNetCoreTargetManifest>` set to `false` in the project file as in the following example:

```
<PropertyGroup>
  <PublishWithAspNetCoreTargetManifest>false</PublishWithAspNetCoreTargetManifest>
</PropertyGroup>
```

NOTE

For [self-contained deployment](#) apps, it's assumed that the target system doesn't necessarily contain the required manifest packages. Therefore, `<PublishWithAspNetCoreTargetManifest>` cannot be set to `true` for an self-contained app.

See also

- [dotnet-publish](#)
- [dotnet-store](#)

.NET RID Catalog

9/20/2022 • 5 minutes to read • [Edit Online](#)

RID is short for *Runtime Identifier*. RID values are used to identify target platforms where the application runs. They're used by .NET packages to represent platform-specific assets in NuGet packages. The following values are examples of RIDs: `linux-x64`, `ubuntu.14.04-x64`, `win7-x64`, or `osx.10.12-x64`. For the packages with native dependencies, the RID designates on which platforms the package can be restored.

A single RID can be set in the `<RuntimeIdentifier>` element of your project file. Multiple RIDs can be defined as a semicolon-delimited list in the project file's `<RuntimeIdentifiers>` element. They're also used via the `--runtime` option with the following .NET CLI commands:

- `dotnet build`
- `dotnet clean`
- `dotnet pack`
- `dotnet publish`
- `dotnet restore`
- `dotnet run`
- `dotnet store`

RIDs that represent concrete operating systems usually follow this pattern:

`[os].[version]-[architecture]-[additional qualifiers]` where:

- `[os]` is the operating/platform system moniker. For example, `ubuntu`.
- `[version]` is the operating system version in the form of a dot-separated (`.`) version number. For example, `15.10`.

The version **shouldn't** be a marketing version, as marketing versions often represent multiple discrete versions of the operating system with varying platform API surface area.

- `[architecture]` is the processor architecture. For example: `x86`, `x64`, `arm`, or `arm64`.
- `[additional qualifiers]` further differentiate different platforms. For example: `aot`.

RID graph

The RID graph or runtime fallback graph is a list of RIDs that are compatible with each other. The RIDs are defined in the [Microsoft.NETCore.Platforms](#) package. You can see the list of supported RIDs and the RID graph in the `runtime.json` file, which is located in the `dotnet/runtime` repository. In this file, you can see that all RIDs, except for the base one, contain an `#import` statement. These statements indicate compatible RIDs.

When NuGet restores packages, it tries to find an exact match for the specified runtime. If an exact match is not found, NuGet walks back the graph until it finds the closest compatible system according to the RID graph.

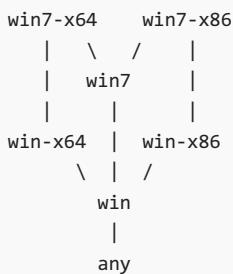
The following example is the actual entry for the `osx.10.12-x64` RID:

```
"osx.10.12-x64": {
    "#import": [ "osx.10.12", "osx.10.11-x64" ]
}
```

The above RID specifies that `osx.10.12-x64` imports `osx.10.11-x64`. So, when NuGet restores packages, it tries

to find an exact match for `osx.10.12-x64` in the package. If NuGet can't find the specific runtime, it can restore packages that specify `osx.10.11-x64` runtimes, for example.

The following example shows a slightly bigger RID graph also defined in the `runtime.json` file:



All RIDs eventually map back to the root `any` RID.

There are some considerations about RIDs that you have to keep in mind when working with them:

- Don't try to parse RIDs to retrieve component parts.
- Use RIDs that are already defined for the platform.
- The RIDs need to be specific, so don't assume anything from the actual RID value.
- Don't build RIDs programmatically unless absolutely necessary.

Some apps need to compute RIDs programmatically. If so, the computed RIDs must match the catalog exactly, including in casing. RIDs with different casing would cause problems when the OS is case sensitive, for example, Linux, because the value is often used when constructing things like output paths. For example, consider a custom publishing wizard in Visual Studio that relies on information from the solution configuration manager and project properties. If the solution configuration passes an invalid value, for example, `ARM64` instead of `arm64`, it could result in an invalid RID, such as `win-ARM64`.

Using RIDs

To be able to use RIDs, you have to know which RIDs exist. New values are added regularly to the platform. For the latest and complete version, see the `runtime.json` file in the `dotnet/runtime` repository.

RIDs that aren't tied to a specific version or OS distribution are the preferred choice, especially when dealing with multiple Linux distros since most distribution RIDs are mapped to the not-distribution-specific RIDs.

The following list shows a small subset of the most common RIDs used for each OS.

Windows RIDs

Only common values are listed. For the latest and complete version, see the `runtime.json` file in the `dotnet/runtime` repository.

- Windows, not version-specific
 - `win-x64`
 - `win-x86`
 - `win-arm`
 - `win-arm64`
- Windows 7 / Windows Server 2008 R2
 - `win7-x64`
 - `win7-x86`

- Windows 8.1 / Windows Server 2012 R2
 - `win81-x64`
 - `win81-x86`
 - `win81-arm`
- Windows 11 / Windows Server 2022 / Windows 10 / Windows Server 2016
 - `win10-x64`
 - `win10-x86`
 - `win10-arm`
 - `win10-arm64`

There are no `win11` RIDs; use `win10` RIDs for Windows 11. For more information, see [.NET dependencies and requirements](#).

Linux RIDs

Only common values are listed. For the latest and complete version, see the `runtime.json` file in the `dotnet/runtime` repository. Devices running a distribution not listed below may work with one of the not-distribution-specific RIDs. For example, Raspberry Pi devices running a Linux distribution not listed can be targeted with `linux-arm`.

- Linux, not distribution-specific
 - `linux-x64` (Most desktop distributions like CentOS, Debian, Fedora, Ubuntu, and derivatives)
 - `linux-musl-x64` (Lightweight distributions using `musl` like Alpine Linux)
 - `linux-arm` (Linux distributions running on Arm like Raspbian on Raspberry Pi Model 2+)
 - `linux-arm64` (Linux distributions running on 64-bit Arm like Ubuntu Server 64-bit on Raspberry Pi Model 3+)
- Red Hat Enterprise Linux
 - `rhel-x64` (Superseded by `linux-x64` for RHEL above version 6)
 - `rhel.6-x64`
- Tizen
 - `tizen`
 - `tizen.4.0.0`
 - `tizen.5.0.0`

For more information, see [.NET dependencies and requirements](#).

macOS RIDs

macOS RIDs use the older "OSX" branding. Only common values are listed. For the latest and complete version, see the `runtime.json` file in the `dotnet/runtime` repository.

- macOS, not version-specific
 - `osx-x64` (Minimum OS version is macOS 10.12 Sierra)
- macOS 10.10 Yosemite
 - `osx.10.10-x64`
- macOS 10.11 El Capitan
 - `osx.10.11-x64`
- macOS 10.12 Sierra
 - `osx.10.12-x64`
- macOS 10.13 High Sierra

- `osx.10.13-x64`
- macOS 10.14 Mojave
 - `osx.10.14-x64`
- macOS 10.15 Catalina
 - `osx.10.15-x64`
- macOS 11.0 Big Sur
 - `osx.11.0-x64`
 - `osx.11.0-arm64`
- macOS 12 Monterey
 - `osx.12-x64`
 - `osx.12-arm64`

For more information, see [.NET dependencies and requirements](#).

iOS RIDs

Only common values are listed. For the latest and complete version, see the [runtime.json](#) file in the `dotnet/runtime` repository.

- iOS, not version-specific
 - `ios-arm64`
- iOS 10
 - `ios.10-arm64`
- iOS 11
 - `ios.11-arm64`
- iOS 12
 - `ios.12-arm64`
- iOS 13
 - `ios.13-arm64`
- iOS 14
 - `ios.14-arm64`
- iOS 15
 - `ios.15-arm64`

Android RIDs

Only common values are listed. For the latest and complete version, see the [runtime.json](#) file in the `dotnet/runtime` repository.

- Android, not version-specific
 - `android-arm64`
- Android 21
 - `android.21-arm64`
- Android 22
 - `android.22-arm64`
- Android 23
 - `android.23-arm64`
- Android 24
 - `android.24-arm64`

- Android 25
 - android.25-arm64
- Android 26
 - android.26-arm64
- Android 27
 - android.27-arm64
- Android 28
 - android.28-arm64
- Android 29
 - android.29-arm64
- Android 30
 - android.30-arm64
- Android 31
 - android.31-arm64
- Android 32
 - android.32-arm64

See also

- [Runtime IDs](#)

How resource manifest files are named

9/20/2022 • 3 minutes to read • [Edit Online](#)

When MSBuild compiles a .NET Core project, XML resource files, which have the `.resx` file extension, are converted into binary `.resources` files. The binary files are embedded into the output of the compiler and can be read by the `ResourceManager`. This article describes how MSBuild chooses a name for each `.resources` file.

TIP

If you explicitly add a resource item to your project file, and it's also included with the default include globs for .NET Core, you will get a build error. To manually include resource files as `EmbeddedResource` items, set the `EnableDefaultEmbeddedResourceItems` property to false.

Default name

In .NET Core 3.0 and later, the default name for a resource manifest is used when both of the following conditions are met:

- The resource file is not explicitly included in the project file as an `EmbeddedResource` item with `LogicalName`, `ManifestResourceName`, or `DependentUpon` metadata.
- The `EmbeddedResourceUseDependentUponConvention` property is not set to `false` in the project file. By default, this property is set to `true`. For more information, see [EmbeddedResourceUseDependentUponConvention](#).

If the resource file is colocated with a source file (`.cs` or `.vb`) of the same root file name, the full name of the first type that's defined in the source file is used for the manifest file name. For example, if `MyNamespace.Form1` is the first type defined in `Form1.cs`, and `Form1.cs` is colocated with `Form1.resx`, the generated manifest name for that resource file is `MyNamespace.Form1.resources`.

LogicalName metadata

If a resource file is explicitly included in the project file as an `EmbeddedResource` item with `LogicalName` metadata, the `LogicalName` value is used as the manifest name. `LogicalName` takes precedence over any other metadata or setting.

For example, the manifest name for the resource file that's defined in the following project file snippet is `SomeName.resources`.

```
<EmbeddedResource Include="X.resx" LogicalName="SomeName.resources" />
```

-or-

```
<EmbeddedResource Include="X.fr-FR.resx" LogicalName="SomeName.resources" />
```

NOTE

- If `LogicalName` is not specified, an `EmbeddedResource` with two dots (.) in the file name doesn't work, which means that `GetManifestResourceNames` doesn't return that file.

The following example works correctly:

```
<EmbeddedResource Include="X.resx" />
```

The following example doesn't work:

```
<EmbeddedResource Include="X.fr-FR.resx" />
```

ManifestResourceName metadata

If a resource file is explicitly included in the project file as an `EmbeddedResource` item with `ManifestResourceName` metadata (and `LogicalName` is absent), the `ManifestResourceName` value, combined with the file extension `.resources`, is used as the manifest file name.

For example, the manifest name for the resource file that's defined in the following project file snippet is `SomeName.resources`.

```
<EmbeddedResource Include="X.resx" ManifestResourceName="SomeName" />
```

The manifest name for the resource file that's defined in the following project file snippet is `SomeName.fr-FR.resources`.

```
<EmbeddedResource Include="X.fr-FR.resx" ManifestResourceName="SomeName.fr-FR" />
```

DependentUpon metadata

If a resource file is explicitly included in the project file as an `EmbeddedResource` item with `DependentUpon` metadata (and `LogicalName` and `ManifestResourceName` are absent), information from the source file defined by `DependentUpon` is used for the resource manifest file name. Specifically, the name of the first type that's defined in the source file is used in the manifest name as follows: `Namespace.Classname[.Culture].resources`.

For example, the manifest name for the resource file that's defined in the following project file snippet is `Namespace.Classname.resources` (where `Namespace.Classname` is the first class that's defined in `MyTypes.cs`).

```
<EmbeddedResource Include="X.resx" DependentUpon="MyTypes.cs" />
```

The manifest name for the resource file that's defined in the following project file snippet is `Namespace.Classname.fr-FR.resources` (where `Namespace.Classname` is the first class that's defined in `MyTypes.cs`).

```
<EmbeddedResource Include="X.fr-FR.resx" DependentUpon="MyTypes.cs" />
```

EmbeddedResourceUseDependentUponConvention property

If `EmbeddedResourceUseDependentUponConvention` is set to `false` in the project file, each resource manifest file

name is based off the root namespace for the project and the relative path from the project root to the `.resx` file.

More specifically, the generated resource manifest file name is

`RootNamespace.RelativePathWithDotsForSlashes.[Culture.]resources`. This is also the logic used to generate manifest names in .NET Core versions prior to 3.0.

NOTE

- If `RootNamespace` is not defined, it defaults to the project name.
- If `LogicalName`, `ManifestResourceName`, or `DependentUpon` metadata is specified for an `EmbeddedResource` item in the project file, this naming rule does not apply to that resource file.

See also

- [How Manifest Resource Naming Works](#)
- [MSBuild properties for .NET SDK projects](#)
- [MSBuild breaking changes](#)

Introduction to .NET and Docker

9/20/2022 • 3 minutes to read • [Edit Online](#)

.NET Core can easily run in a Docker container. Containers provide a lightweight way to isolate your application from the rest of the host system, sharing just the kernel, and using resources given to your application. If you're unfamiliar with Docker, it's highly recommended that you read through Docker's [overview documentation](#).

For more information about how to install Docker, see the download page for [Docker Desktop: Community Edition](#).

Docker basics

There are a few concepts you should be familiar with. The Docker client has a CLI that you can use to manage images and containers. As previously stated, you should take the time to read through the [Docker overview](#) documentation.

Images

An image is an ordered collection of filesystem changes that form the basis of a container. The image doesn't have a state and is read-only. Much the time an image is based on another image, but with some customization. For example, when you create a new image for your application, you would base it on an existing image that already contains the .NET Core runtime.

Because containers are created from images, images have a set of run parameters (such as a starting executable) that run when the container starts.

Containers

A container is a runnable instance of an image. As you build your image, you deploy your application and dependencies. Then, multiple containers can be instantiated, each isolated from one another. Each container instance has its own filesystem, memory, and network interface.

Registries

Container registries are a collection of image repositories. You can base your images on a registry image. You can create containers directly from an image in a registry. The [relationship between Docker containers, images, and registries](#) is an important concept when [architecting and building containerized applications or microservices](#). This approach greatly shortens the time between development and deployment.

Docker has a public registry hosted at the [Docker Hub](#) that you can use. [.NET Core related images](#) are listed at the Docker Hub.

The [Microsoft Container Registry \(MCR\)](#) is the official source of Microsoft-provided container images. The MCR is built on Azure CDN to provide globally-replicated images. However, the MCR does not have a public-facing website and the primary way to learn about Microsoft-provided container images is through the [Microsoft Docker Hub pages](#).

Dockerfile

A **Dockerfile** is a file that defines a set of instructions that creates an image. Each instruction in the **Dockerfile** creates a layer in the image. For the most part, when you rebuild the image, only the layers that have changed are rebuilt. The **Dockerfile** can be distributed to others and allows them to recreate a new image in the same manner you created it. While this allows you to distribute the *instructions* on how to create the image, the main way to distribute your image is to publish it to a registry.

.NET Core images

Official .NET Core Docker images are published to the Microsoft Container Registry (MCR) and are discoverable at the [Microsoft .NET Core Docker Hub repository](#). Each repository contains images for different combinations of the .NET (SDK or Runtime) and OS that you can use.

Microsoft provides images that are tailored for specific scenarios. For example, the [ASP.NET Core repository](#) provides images that are built for running ASP.NET Core apps in production.

Azure services

Various Azure services support containers. You create a Docker image for your application and deploy it to one of the following services:

- [Azure Kubernetes Service \(AKS\)](#)
Scale and orchestrate Windows & Linux containers using Kubernetes.
- [Azure App Service](#)
Deploy web apps or APIs using containers in a PaaS environment.
- [Azure Container Apps](#)
Run your container workloads without managing servers, orchestration, or infrastructure and leverage native support for [Dapr](#) and [KEDA](#) for observability and scaling to zero.
- [Azure Container Instances](#)
Create individual containers in the cloud without any higher-level management services.
- [Azure Batch](#)
Run repetitive compute jobs using containers.
- [Azure Service Fabric](#)
Lift, shift, and modernize .NET applications to microservices using Windows & Linux containers.
- [Azure Container Registry](#)
Store and manage container images across all types of Azure deployments.

Next steps

- [Learn how to containerize a .NET Core application.](#)
- [Learn how to containerize an ASP.NET Core application.](#)
- [Try the Learn ASP.NET Core Microservice tutorial.](#)
- [Learn about Container Tools in Visual Studio](#)

Tutorial: Containerize a .NET app

9/20/2022 • 13 minutes to read • [Edit Online](#)

In this tutorial, you'll learn how to containerize a .NET application with Docker. Containers have many features and benefits, such as being an immutable infrastructure, providing a portable architecture, and enabling scalability. The image can be used to create containers for your local development environment, private cloud, or public cloud.

In this tutorial, you:

- Create and publish a simple .NET app
- Create and configure a Dockerfile for .NET
- Build a Docker image
- Create and run a Docker container

You'll understand the Docker container build and deploy tasks for a .NET application. The *Docker platform* uses the *Docker engine* to quickly build and package apps as *Docker images*. These images are written in the *Dockerfile* format to be deployed and run in a layered container.

NOTE

This tutorial is **not** for ASP.NET Core apps. If you're using ASP.NET Core, see the [Learn how to containerize an ASP.NET Core application](#) tutorial.

Prerequisites

Install the following prerequisites:

- [.NET SDK](#)
If you have .NET installed, use the `dotnet --info` command to determine which SDK you're using.
- [Docker Community Edition](#)
- A temporary working folder for the *Dockerfile* and .NET example app. In this tutorial, the name *docker-working* is used as the working folder.

Create .NET app

You need a .NET app that the Docker container will run. Open your terminal, create a working folder if you haven't already, and enter it. In the working folder, run the following command to create a new project in a subdirectory named *app*.

```
dotnet new console -o App -n DotNet.Docker
```

Your folder tree will look like the following:

```
└ docker-working
  └─ App
    ├ DotNet.Docker.csproj
    ├ Program.cs
    └─ obj
      └─ DotNet.Docker.csproj.nuget.dgspec.json
      └─ DotNet.Docker.csproj.nuget.g.props
      └─ DotNet.Docker.csproj.nuget.g.targets
      └─ project.assets.json
      └─ project.nuget.cache
```

The `dotnet new` command creates a new folder named *App* and generates a "Hello World" console application. Change directories and navigate into the *App* folder, from your terminal session. Use the `dotnet run` command to start the app. The application will run, and print `Hello World!` below the command:

```
dotnet run
Hello World!
```

The default template creates an app that prints to the terminal and then immediately terminates. For this tutorial, you'll use an app that loops indefinitely. Open the *Program.cs* file in a text editor.

TIP

If you're using Visual Studio Code, from the previous terminal session type the following command:

```
code .
```

This will open the *App* folder that contains the project in Visual Studio Code.

The *Program.cs* should look like the following C# code:

```
Console.WriteLine("Hello World!");
```

Replace the file with the following code that counts numbers every second:

```
var counter = 0;
var max = args.Length != 0 ? Convert.ToInt32(args[0]) : -1;
while (max == -1 || counter < max)
{
    Console.WriteLine($"Counter: {++counter}");
    await Task.Delay(TimeSpan.FromMilliseconds(1_000));
}
```

Save the file and test the program again with `dotnet run`. Remember that this app runs indefinitely. Use the cancel command `Ctrl+C` to stop it. The following is an example output:

```
dotnet run
Counter: 1
Counter: 2
Counter: 3
Counter: 4
^C
```

If you pass a number on the command line to the app, it will only count up to that amount and then exit. Try it

with `dotnet run -- 5` to count to five.

IMPORTANT

Any parameters after `--` are not passed to the `dotnet run` command and instead are passed to your application.

Publish .NET app

Before adding the .NET app to the Docker image, first it must be published. It is best to have the container run the published version of the app. To publish the app, run the following command:

```
dotnet publish -c Release
```

This command compiles your app to the *publish* folder. The path to the *publish* folder from the working folder should be `.\App\bin\Release\net6.0\publish\`

- [Windows](#)
- [Linux](#)

From the *App* folder, get a directory listing of the publish folder to verify that the *DotNet.Docker.dll* file was created.

```
dir .\bin\Release\net6.0\publish

Directory: C:\Users\dapine\AppData\Local\Temp\dotnet-docker-1\app\bin\Release\net6.0\publish

Mode                 LastWriteTime       Length Name
----                 -----          -----
-a---        3/8/2022 10:43 AM         431 DotNet.Docker.deps.json
-a---        3/8/2022 10:43 AM        6144 DotNet.Docker.dll
-a---        3/8/2022 10:43 AM      149504 DotNet.Docker.exe
-a---        3/8/2022 10:43 AM      10516 DotNet.Docker.pdb
-a---        3/8/2022 10:43 AM        253 DotNet.Docker.runtimeconfig.json
```

Create the Dockerfile

The *Dockerfile* file is used by the `docker build` command to create a container image. This file is a text file named *Dockerfile* that doesn't have an extension.

Create a file named *Dockerfile* in the directory containing the *.csproj* and open it in a text editor. This tutorial will use the ASP.NET Core runtime image (which contains the .NET runtime image) and corresponds with the .NET console application.

```

FROM mcr.microsoft.com/dotnet/sdk:6.0 AS build-env
WORKDIR /app

# Copy everything
COPY . .
# Restore as distinct layers
RUN dotnet restore
# Build and publish a release
RUN dotnet publish -c Release -o out

# Build runtime image
FROM mcr.microsoft.com/dotnet/aspnet:6.0
WORKDIR /app
COPY --from=build-env /app/out .
ENTRYPOINT ["dotnet", "DotNet.Docker.dll"]

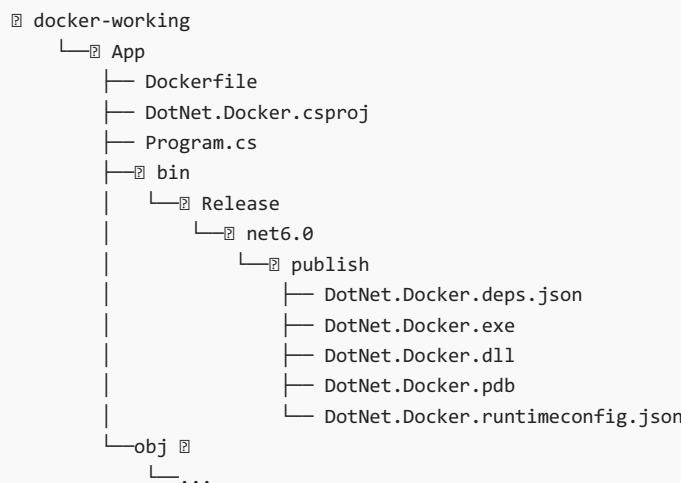
```

NOTE

The ASP.NET Core runtime image is used intentionally here, although the `mcr.microsoft.com/dotnet/runtime:6.0` image could have been used.

The `FROM` keyword requires a fully qualified Docker container image name. The Microsoft Container Registry (MCR, `mcr.microsoft.com`) is a syndicate of Docker Hub — which hosts publicly accessible containers. The `dotnet` segment is the container repository, whereas the `sdk` or `aspnet` segment is the container image name. The image is tagged with `6.0`, which is used for versioning. Thus, `mcr.microsoft.com/dotnet/aspnet:6.0` is the .NET 6.0 runtime. Make sure that you pull the runtime version that matches the runtime targeted by your SDK. For example, the app created in the previous section used the .NET 6.0 SDK and the base image referred to in the *Dockerfile* is tagged with `6.0`.

Save the *Dockerfile* file. The directory structure of the working folder should look like the following. Some of the deeper-level files and folders have been omitted to save space in the article:



From your terminal, run the following command:

```
docker build -t counter-image -f Dockerfile .
```

Docker will process each line in the *Dockerfile*. The `.` in the `docker build` command sets the build context of the image. The `-f` switch is the path to the *Dockerfile*. This command builds the image and creates a local repository named `counter-image` that points to that image. After this command finishes, run `docker images` to see a list of images installed:

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
counter-image	latest	2f15637dc1f6	10 minutes ago	208MB

The `counter-image` repository is the name of the image. The `latest` tag is the tag that is used to identify the image. The `2f15637dc1f6` is the image ID. The `10 minutes ago` is the time the image was created. The `208MB` is the size of the image. The final steps of the *Dockerfile* are to create a container from the image and run the app, copy the published app to the container, and define the entry point.

```
FROM mcr.microsoft.com/dotnet/aspnet:6.0
WORKDIR /app
COPY --from=build-env /app/out .
ENTRYPOINT ["dotnet", "DotNet.Docker.dll"]
```

The `COPY` command tells Docker to copy the specified folder on your computer to a folder in the container. In this example, the *publish* folder is copied to a folder named *app* in the container.

The `WORKDIR` command changes the **current directory** inside of the container to *app*.

The next command, `ENTRYPOINT`, tells Docker to configure the container to run as an executable. When the container starts, the `ENTRYPOINT` command runs. When this command ends, the container will automatically stop.

TIP

For added security, you can opt out of the diagnostic pipeline. When you opt-out this allows the container to run as read-only. To do this, specify a `DOTNET_EnableDiagnostics` environment variable as `0` (just before the `ENTRYPOINT` step):

```
ENV DOTNET_EnableDiagnostics=0
```

For more information on various .NET environment variables, see [.NET environment variables](#).

NOTE

.NET 6 standardizes on the prefix `DOTNET_` instead of `COMPlus_` for environment variables that configure .NET run-time behavior. However, the `COMPlus_` prefix will continue to work. If you're using a previous version of the .NET runtime, you should still use the `COMPlus_` prefix for environment variables.

From your terminal, run `docker build -t counter-image -f Dockerfile .` and when that command finishes, run `docker images`.

```

docker build -t counter-image -f Dockerfile .
[+] Building 3.1s (14/14) FINISHED
=> [internal] load build definition from Dockerfile          0.5s
=> => transferring dockerfile: 32B                          0.0s
=> [internal] load .dockerignore                           0.6s
=> => transferring context: 2B                            0.0s
=> [internal] load metadata for mcr.microsoft.com/dotnet/aspnet:6.0   0.8s
=> [internal] load metadata for mcr.microsoft.com/dotnet/sdk:6.0      1.1s
=> [stage-1 1/3] FROM mcr.microsoft.com/dotnet/aspnet:6.0@sha256:f1539d71 0.0s
=> [internal] load build context                           0.4s
=> => transferring context: 4.00kB                      0.1s
=> [build-env 1/5] FROM mcr.microsoft.com/dotnet/sdk:6.0@sha256:16e355af1 0.0s
=> CACHED [stage-1 2/3] WORKDIR /App                     0.0s
=> CACHED [build-env 2/5] WORKDIR /App                  0.0s
=> CACHED [build-env 3/5] COPY . ./                   0.0s
=> CACHED [build-env 4/5] RUN dotnet restore            0.0s
=> CACHED [build-env 5/5] RUN dotnet publish -c Release -o out 0.0s
=> CACHED [stage-1 3/3] COPY --from=build-env /App/out . 0.0s
=> exporting to image                                0.4s
=> => exporting layers                             0.0s
=> => writing image sha256:2f15637d                0.1s
=> => naming to docker.io/library/counter-image

docker images
REPOSITORY           TAG      IMAGE ID      CREATED       SIZE
counter-image        latest   2f15637dc1f6  10 minutes ago  208MB

```

Each command in the *Dockerfile* generated a layer and created an **IMAGE ID**. The final **IMAGE ID** (yours will be different) is **2f15637dc1f6** and next you'll create a container based on this image.

Create a container

Now that you have an image that contains your app, you can create a container. You can create a container in two ways. First, create a new container that is stopped.

```
docker create --name core-counter counter-image
```

The `docker create` command from above will create a container based on the **counter-image** image. The output of that command shows you the **CONTAINER ID** (yours will be different) of the created container:

```
cf01364df4539812684c64277f5363a8fb354ef4c90785dc0845769a6c5b0f8e
```

To see a list of *all* containers, use the `docker ps -a` command:

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
cf01364df453	counter-image	"dotnet DotNet.Docke..."	18 seconds ago	Created		core-counter

Manage the container

The container was created with a specific name `core-counter`, this name is used to manage the container. The following example uses the `docker start` command to start the container, and then uses the `docker ps` command to only show containers that are running:

```
docker start core-counter
core-counter

docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
cf01364df453 counter-image "dotnet DotNet.Docke..." 53 seconds ago Up 10 seconds
core-
counter
```

Similarly, the `docker stop` command will stop the container. The following example uses the `docker stop` command to stop the container, and then uses the `docker ps` command to show that no containers are running:

```
docker stop core-counter
core-counter

docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
```

Connect to a container

After a container is running, you can connect to it to see the output. Use the `docker start` and `docker attach` commands to start the container and peek at the output stream. In this example, the `Ctrl+C` keystroke is used to detach from the running container. This keystroke will end the process in the container unless otherwise specified, which would stop the container. The `--sig-proxy=false` parameter ensures that `Ctrl+C` will not stop the process in the container.

After you detach from the container, reattach to verify that it's still running and counting.

```
docker start core-counter
core-counter

docker attach --sig-proxy=false core-counter
Counter: 7
Counter: 8
Counter: 9
^C

docker attach --sig-proxy=false core-counter
Counter: 17
Counter: 18
Counter: 19
^C
```

Delete a container

For this article, you don't want containers hanging around that don't do anything. Delete the container you previously created. If the container is running, stop it.

```
docker stop core-counter
```

The following example lists all containers. It then uses the `docker rm` command to delete the container and then checks a second time for any running containers.

```

docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
PORTS      NAMES
2f6424a7ddce   counter-image   "dotnet DotNet.Dock..."   7 minutes ago    Exited (143) 20 seconds ago
core-counter

docker rm core-counter
core-counter

docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS

```

Single run

Docker provides the `docker run` command to create and run the container as a single command. This command eliminates the need to run `docker create` and then `docker start`. You can also set this command to automatically delete the container when the container stops. For example, use `docker run -it --rm` to do two things, first, automatically use the current terminal to connect to the container, and then when the container finishes, remove it:

```

docker run -it --rm counter-image
Counter: 1
Counter: 2
Counter: 3
Counter: 4
Counter: 5
^C

```

The container also passes parameters into the execution of the .NET app. To instruct the .NET app to count only to 3 pass in 3.

```

docker run -it --rm counter-image 3
Counter: 1
Counter: 2
Counter: 3

```

With `docker run -it`, the `Ctrl+C` command will stop process that is running in the container, which in turn, stops the container. Since the `--rm` parameter was provided, the container is automatically deleted when the process is stopped. Verify that it doesn't exist:

```

docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS

```

Change the ENTRYPPOINT

The `docker run` command also lets you modify the `ENTRYPOINT` command from the *Dockerfile* and run something else, but only for that container. For example, use the following command to run `bash` or `cmd.exe`. Edit the command as necessary.

- [Windows](#)
- [Linux](#)

In this example, `ENTRYPOINT` is changed to `cmd.exe`. `Ctrl+C` is pressed to end the process and stop the container.

```
docker run -it --rm --entrypoint "cmd.exe" counter-image

Microsoft Windows [Version 10.0.17763.379]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\>dir
Volume in drive C has no label.
Volume Serial Number is 3005-1E84

Directory of C:\

04/09/2019  08:46 AM      <DIR>          app
03/07/2019  10:25 AM            5,510 License.txt
04/02/2019  01:35 PM      <DIR>          Program Files
04/09/2019  01:06 PM      <DIR>          Users
04/02/2019  01:35 PM      <DIR>          Windows
               1 File(s)       5,510 bytes
               4 Dir(s)  21,246,517,248 bytes free

C:\>^C
```

Essential commands

Docker has many different commands that create, manage, and interact with containers and images. These Docker commands are essential to managing your containers:

- [docker build](#)
- [docker run](#)
- [docker ps](#)
- [docker stop](#)
- [docker rm](#)
- [docker rmi](#)
- [docker image](#)

Clean up resources

During this tutorial, you created containers and images. If you want, delete these resources. Use the following commands to

1. List all containers

```
docker ps -a
```

2. Stop containers that are running by their name.

```
docker stop counter-image
```

3. Delete the container

```
docker rm counter-image
```

Next, delete any images that you no longer want on your machine. Delete the image created by your *Dockerfile* and then delete the .NET image the *Dockerfile* was based on. You can use the IMAGE ID or the **REPOSITORY:TAG** formatted string.

```
docker rmi counter-image:latest  
docker rmi mcr.microsoft.com/dotnet/aspnet:6.0
```

Use the `docker images` command to see a list of images installed.

TIP

Image files can be large. Typically, you would remove temporary containers you created while testing and developing your app. You usually keep the base images with the runtime installed if you plan on building other images based on that runtime.

Next steps

- [Learn how to containerize an ASP.NET Core application.](#)
- [Try the ASP.NET Core Microservice Tutorial.](#)
- [Review the Azure services that support containers.](#)
- [Read about Dockerfile commands.](#)
- [Explore the Container Tools for Visual Studio](#)

GitHub Actions and .NET

9/20/2022 • 4 minutes to read • [Edit Online](#)

In this overview, you'll learn what role [GitHub Actions](#) play in .NET application development. GitHub Actions allow your source code repositories to automate continuous integration (CI) and continuous delivery (CD). Beyond that, GitHub Actions expose more advanced scenarios — providing hooks for automation with code reviews, branch management, and issue triaging. With your .NET source code in GitHub you can leverage GitHub Actions in many ways.

GitHub Actions

GitHub Actions represent standalone commands, such as:

- [actions/checkout](#) - This action checks-out your repository under `$GITHUB_WORKSPACE`, so your workflow can access it.
- [actions/setup-dotnet](#) - This action sets up a .NET CLI environment for use in actions.
- [dotnet/versionsweeper](#) - This action sweeps .NET repos for out-of-support target versions of .NET.

While these commands are isolated to a single action, they're powerful through *workflow composition*. In workflow composition, you define the *events* that trigger the workflow. Once a workflow is running, there are various *jobs* it's instructed to perform — with each job defining any number of *steps*. The *steps* delegate out to GitHub Actions, or alternatively call command-line scripts.

For more information, see [Introduction to GitHub Actions](#). Think of a workflow file as a composition that represents the various steps to build, test, and/or publish an application. Many [.NET CLI commands](#) are available, most of which could be used in the context of a GitHub Action.

Custom GitHub Actions

While there are plenty of GitHub Actions available in the [Marketplace](#), you may want to author your own. You can create GitHub Actions that run .NET applications. For more information, see [Tutorial: Create a GitHub Action with .NET](#).

Workflow file

GitHub Actions are utilized through a workflow file. The workflow file must be located in the `.github/workflows` directory of the repository, and is expected to be YAML (either `*.yml` or `*.yaml`). Workflow files define the *workflow composition*. A workflow is a configurable automated process made up of one or more jobs. For more information, see [Workflow syntax for GitHub Actions](#).

Example workflow files

There are many examples of .NET workflow files provided as [tutorials](#) and [quickstarts](#). Here are several good examples of workflow file names:

Workflow file name

Description

[`build-validation.yml`](#)

Compiles (or builds) the source code. If the source code doesn't compile, this will fail.

[`build-and-test.yml`](#)

Exercises the unit tests within the repository. In order to run tests, the source code must first be compiled — this is really both a build and test workflow (it would supersede the *build-validation.yml*/workflow). Failing unit tests will cause workflow failure.

[publish-app.yml](#)

Packages, and publishes the source code to a destination.

[codeql-analysis.yml](#)

Analyzes your code for security vulnerabilities and coding errors. Any discovered vulnerabilities could cause failure.

Encrypted secrets

To use *encrypted secrets* in your workflow files, you reference the secrets using the [workflow expression syntax](#) from the `secrets` context object.

```
${{ secrets.MY_SECRET_VALUE }} # The MY_SECRET_VALUE must exist in the repository as a secret
```

Secret values are never printed in the logs. Instead, their names are printed with an asterisk representing their values. For example, as each step runs within a job, all of the values it uses are output to the action log. Secret values render similar to the following:

```
MY_SECRET_VALUE: ***
```

IMPORTANT

The `secrets` context provides the GitHub authentication token that is scoped to the repository, branch, and action. It's provided by GitHub without any user intervention:

```
${{ secrets.GITHUB_TOKEN }}
```

For more information, see [Using encrypted secrets in a workflow](#).

Events

Workflows are triggered by many different types of events. In addition to Webhook events, which are the most common, there are also scheduled events and manual events.

Example webhook event

The following example shows how to specify a webhook event trigger for a workflow:

```

name: code coverage

on:
  push:
    branches:
      - main
  pull_request:
    branches:
      - main, staging

jobs:
  coverage:
    runs-on: ubuntu-latest

    # steps omitted for brevity

```

In the preceding workflow, the `push` and `pull_request` events will trigger the workflow to run.

Example scheduled event

The following example shows how to specify a scheduled (cron job) event trigger for a workflow:

```

name: scan
on:
  schedule:
    - cron: '0 0 1 * *'
      # additional events omitted for brevity

jobs:
  build:
    runs-on: ubuntu-latest

    # steps omitted for brevity

```

In the preceding workflow, the `schedule` event specifies the `cron` of `'0 0 1 * *'` which will trigger the workflow to run on the first day of every month. Running workflows on a schedule is great for workflows that take a long time to run, or perform actions that require less frequent attention.

Example manual event

The following example shows how to specify a manual event trigger for a workflow:

```

name: build
on:
  workflow_dispatch:
    inputs:
      reason:
        description: 'The reason for running the workflow'
        required: true
        default: 'Manual run'
    # additional events omitted for brevity

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
      - name: 'Print manual run reason'
        if: ${{ github.event_name == 'workflow_dispatch' }}
        run: |
          echo 'Reason: ${{ github.event.inputs.reason }}'

    # additional steps omitted for brevity

```

In the preceding workflow, the `workflow_dispatch` event requires a `reason` as input. GitHub sees this and its UI dynamically changes to prompt the user into provided the reason for manually running the workflow. The `steps` will print the provided reason from the user.

For more information, see [Events that trigger workflows](#).

.NET CLI

The .NET command-line interface (CLI) is a cross-platform toolchain for developing, building, running, and publishing .NET applications. The .NET CLI is used to `run` as part of individual `steps` within a workflow file. Common command include:

- [dotnet workflow install](#)
- [dotnet restore](#)
- [dotnet build](#)
- [dotnet test](#)
- [dotnet publish](#)

For more information, see [.NET CLI overview](#)

See also

For a more in-depth look at GitHub Actions with .NET, consider the following resources:

- *eBook(s):*
 - [DevOps for ASP.NET Core Developers](#)
- *Quickstart(s):*
 - [Quickstart: Create a build validation GitHub Action](#)
 - [Quickstart: Create a test validation GitHub Action](#)
 - [Quickstart: Create a publish app GitHub Action](#)
 - [Quickstart: Create a security scan GitHub Action](#)
- *Tutorial(s):*
 - [Tutorial: Create a GitHub Action with .NET](#)

Use the .NET SDK in Continuous Integration (CI) environments

9/20/2022 • 8 minutes to read • [Edit Online](#)

This article outlines how to use the .NET SDK and its tools on a build server. The .NET toolset works both interactively, where a developer types commands at a command prompt, and automatically, where a Continuous Integration (CI) server runs a build script. The commands, options, inputs, and outputs are the same, and the only things you supply are a way to acquire the tooling and a system to build your app. This article focuses on scenarios of tool acquisition for CI with recommendations on how to design and structure your build scripts.

Installation options for CI build servers

If you're using GitHub, the installation is very straightforward. You can rely on GitHub Actions to install the .NET SDK in your workflow. The recommended way to install the .NET SDK in a workflow is with the `actions/setup-dotnet`. For more information, see the [Setup .NET Core SDK](#) action in the GitHub marketplace. For examples of how this works, see [Quickstart: Create a build validation GitHub workflow](#).

Native installers

Native installers are available for macOS, Linux, and Windows. The installers require admin (`sudo`) access to the build server. The advantage of using a native installer is that it installs all of the native dependencies required for the tooling to run. Native installers also provide a system-wide installation of the SDK.

macOS users should use the PKG installers. On Linux, there's a choice of using a feed-based package manager, such as `apt-get` for Ubuntu or `yum` for CentOS, or using the packages themselves, DEB or RPM. On Windows, use the MSI installer.

The latest stable binaries are found at [.NET downloads](#). If you wish to use the latest (and potentially unstable) pre-release tooling, use the links provided at the [dotnet/installer GitHub repository](#). For Linux distributions, `.tar.gz` archives (also known as `.tarballs`) are available; use the installation scripts within the archives to install .NET.

Installer script

Using the installer script allows for non-administrative installation on your build server and easy automation for obtaining the tooling. The script takes care of downloading the tooling and extracting it into a default or specified location for use. You can also specify a version of the tooling that you wish to install and whether you want to install the entire SDK or only the shared runtime.

The installer script is automated to run at the start of the build to fetch and install the desired version of the SDK. The *desired version* is whatever version of the SDK your projects require to build. The script allows you to install the SDK in a local directory on the server, run the tools from the installed location, and then clean up (or let the CI service clean up) after the build. This provides encapsulation and isolation to your entire build process. The installation script reference is found in the [dotnet-install](#) article.

NOTE

Azure DevOps Services

When using the installer script, native dependencies aren't installed automatically. You must install the native dependencies if the operating system doesn't have them. For more information, see [.NET dependencies and requirements](#).

CI setup examples

This section describes a manual setup using a PowerShell or bash script, along with descriptions of software as a service (SaaS) CI solutions. The SaaS CI solutions covered are [Travis CI](#), [AppVeyor](#), and [Azure Pipelines](#). For GitHub Actions, see [GitHub Actions and .NET](#)

Manual setup

Each SaaS service has its methods for creating and configuring a build process. If you use a different SaaS solution than those listed or require customization beyond the pre-packaged support, you must perform at least some manual configuration.

In general, a manual setup requires you to acquire a version of the tools (or the latest nightly builds of the tools) and run your build script. You can use a PowerShell or bash script to orchestrate the .NET commands or use a project file that outlines the build process. The [orchestration section](#) provides more detail on these options.

After you create a script that performs a manual CI build server setup, use it on your dev machine to build your code locally for testing purposes. Once you confirm that the script is running well locally, deploy it to your CI build server. A relatively simple PowerShell script demonstrates how to obtain the .NET SDK and install it on a Windows build server:

You provide the implementation for your build process at the end of the script. The script acquires the tools and then executes your build process.

- [PowerShell](#)
- [Bash](#)

```

$ErrorActionPreference="Stop"
$ProgressPreference="SilentlyContinue"

# $LocalDotnet is the path to the locally-installed SDK to ensure the
#   correct version of the tools are executed.
$LocalDotnet=""
# $InstallDir and $CliVersion variables can come from options to the
#   script.
$InstallDir = "./cli-tools"
$CliVersion = "6.0.7"

# Test the path provided by $InstallDir to confirm it exists. If it
#   does, it's removed. This is not strictly required, but it's a
#   good way to reset the environment.
if (Test-Path $InstallDir)
{
    rm -Recurse $InstallDir
}
New-Item -Type "directory" -Path $InstallDir

Write-Host "Downloading the CLI installer..."

# Use the Invoke-WebRequest PowerShell cmdlet to obtain the
#   installation script and save it into the installation directory.
Invoke-WebRequest `

    -Uri "https://dot.net/v1/dotnet-install.ps1" `

    -OutFile "$InstallDir/dotnet-install.ps1"

Write-Host "Installing the CLI requested version ($CliVersion) ..."

# Install the SDK of the version specified in $CliVersion into the
#   specified location ($InstallDir).
& $InstallDir/dotnet-install.ps1 -Version $CliVersion `

    -InstallDir $InstallDir

Write-Host "Downloading and installation of the SDK is complete."

# $LocalDotnet holds the path to dotnet.exe for future use by the
#   script.
$LocalDotnet = "$InstallDir/dotnet"

# Run the build process now. Implement your build script here.

```

Travis CI

You can configure [Travis CI](#) to install the .NET SDK using the `csharp` language and the `dotnet` key. For more information, see the official Travis CI docs on [Building a C#, F#, or Visual Basic Project](#). Note as you access the Travis CI information that the community-maintained `language: csharp` language identifier works for all .NET languages, including F#, and Mono.

Travis CI runs both macOS and Linux jobs in a *build matrix*, where you specify a combination of runtime, environment, and exclusions/inclusions to cover your build combinations for your app. For more information, see the [Customizing the Build](#) article in the Travis CI documentation. The MSBuild-based tools include the LTS and Current runtimes in the package; so by installing the SDK, you receive everything you need to build.

AppVeyor

[AppVeyor](#) installs the .NET 6.0.0 SDK with the `Visual Studio 2022` build worker image. Other build images with different versions of the .NET SDK are available. For more information, see the [appveyor.yml example](#) and the [Build worker images](#) article in the AppVeyor docs.

The .NET SDK binaries are downloaded and unzipped in a subdirectory using the install script, and then they're added to the `PATH` environment variable. Add a build matrix to run integration tests with multiple versions of the .NET SDK:

```
environment:  
matrix:  
  - CLI_VERSION: 6.0.7  
  - CLI_VERSION: Latest  
  
install:  
  # See appveyor.yml example for install script
```

Azure DevOps Services

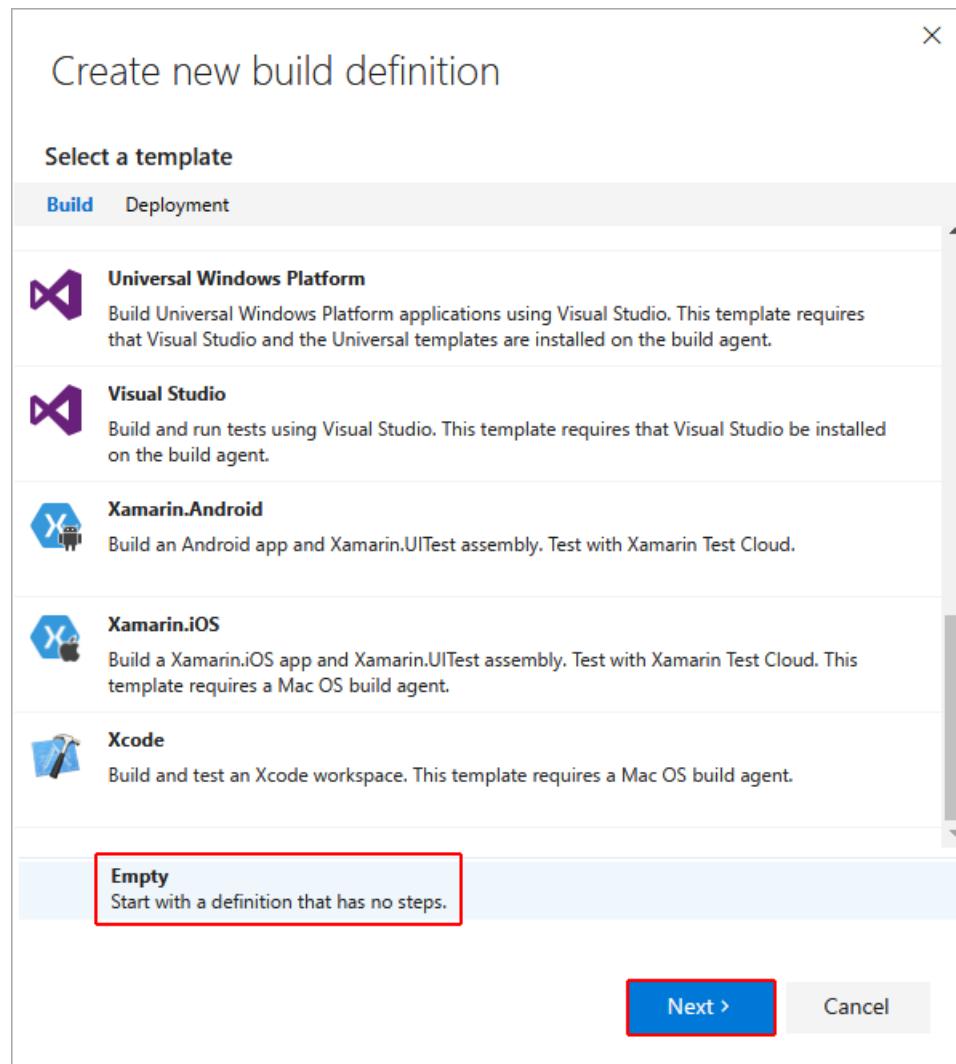
Configure Azure DevOps Services to build .NET projects using one of these approaches:

1. Run the script from the [manual setup step](#) using your commands.
2. Create a build composed of several Azure DevOps Services built-in build tasks that are configured to use .NET tools.

Both solutions are valid. Using a manual setup script, you control the version of the tools that you receive, since you download them as part of the build. The build is run from a script that you must create. This article only covers the manual option. For more information on composing a build with Azure DevOps Services build tasks, see the [Azure Pipelines](#) documentation.

To use a manual setup script in Azure DevOps Services, create a new build definition and specify the script to run for the build step. This is accomplished using the Azure DevOps Services user interface:

1. Start by creating a new build definition. Once you reach the screen that provides you an option to define what kind of a build you wish to create, select the **Empty** option.



2. After configuring the repository to build, you're directed to the build definitions. Select **Add build step**:

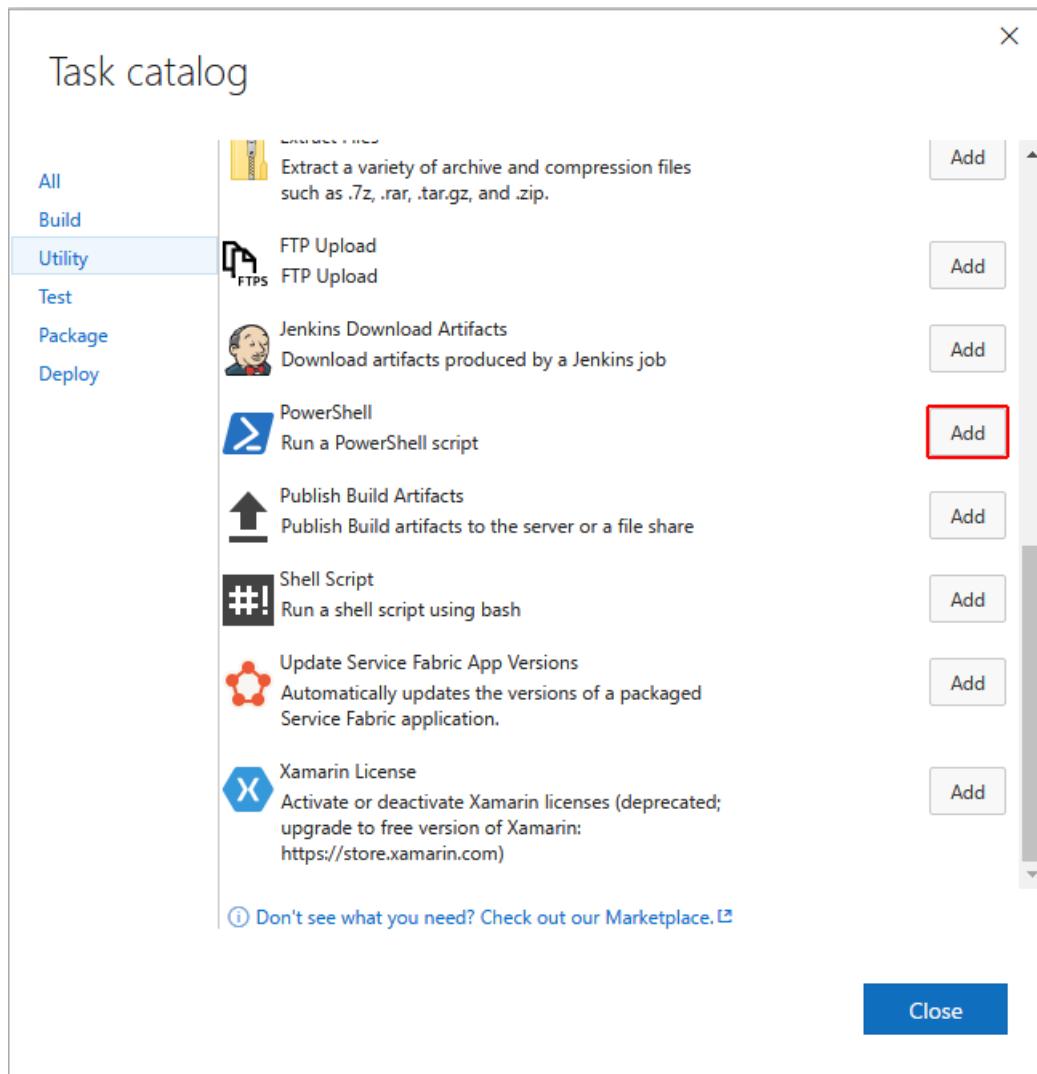
Build Definitions /*

Build Options Repository Variables Trigg

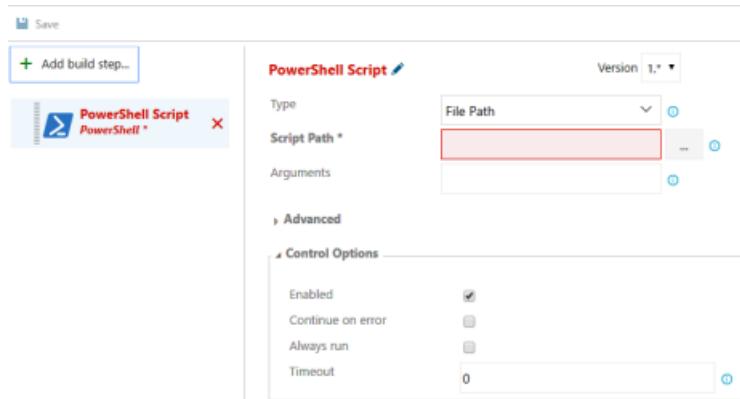
Save

+ Add build step...

3. You're presented with the **Task catalog**. The catalog contains tasks that you use in the build. Since you have a script, select the Add button for **PowerShell: Run a PowerShell script**.



4. Configure the build step. Add the script from the repository that you're building:



Orchestrating the build

Most of this document describes how to acquire the .NET tools and configure various CI services without providing information on how to orchestrate, or *actually build*, your code with .NET. The choices on how to structure the build process depend on many factors that can't be covered in a general way here. For more information on orchestrating your builds with each technology, explore the resources and samples provided in the documentation sets of [Travis CI](#), [AppVeyor](#), and [Azure Pipelines](#).

Two general approaches that you take in structuring the build process for .NET code using the .NET tools are using MSBuild directly or using the .NET command-line commands. Which approach you should take is determined by your comfort level with the approaches and trade-offs in complexity. MSBuild provides you the ability to express your build process as tasks and targets, but it comes with the added complexity of learning MSBuild project file syntax. Using the .NET command-line tools is perhaps simpler, but it requires you to write orchestration logic in a scripting language like `bash` or PowerShell.

See also

- [GitHub Actions and .NET](#)
- [.NET downloads - Linux](#)

.NET-related GitHub Actions

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article lists some of the first party .NET GitHub actions that are hosted on the [dotnet GitHub organization](#).

NOTE

This article is a work-in-progress, and may not list all the available .NET GitHub Actions.

.NET version sweeper

[dotnet/versionswsweeper](#)

This action sweeps .NET repos for out-of-support target versions of .NET.

The .NET docs team uses the .NET version sweeper GitHub Action to automate issue creation. The Action runs on a schedule (as a cron job). When it detects that .NET projects target out-of-support versions, it creates issues to report its findings. The output is configurable and helpful for tracking .NET version support concerns.

The Action is available on [GitHub Marketplace](#).

.NET code analysis

[dotnet/code-analysis](#)

This action runs the code analysis rules that are included in the .NET SDK as part of continuous integration (CI). The action runs both [code-quality \(CAXXXX\) rules](#) and [code-style \(IDEXXXX\) rules](#). Consider using this GitHub Action in the following scenarios:

- You only want to see compiler diagnostics when you compile locally, but you still want to catch code analysis issues in a separate phase.
- You want to improve compile-time performance by offloading expensive analyzers, such as the dataflow analysis-based security analyzers, to the CI phase.
- You want to run the default .NET SDK code analyzers when you compile locally, but you want to run an extended set of code analyzers in the CI phase.

You can configure the action in various ways, including whether you want violations to break the CI build. For more information, see the [README file](#). For more information about .NET code analysis, see [Overview of .NET code analysis](#).

Tutorial: Create a GitHub Action with .NET

9/20/2022 • 12 minutes to read • [Edit Online](#)

Learn how to create a .NET app that can be used as a GitHub Action. [GitHub Actions](#) enable workflow automation and composition. With GitHub Actions, you can build, test, and deploy source code from GitHub. Additionally, actions expose the ability to programmatically interact with issues, create pull requests, perform code reviews, and manage branches. For more information on continuous integration with GitHub Actions, see [Building and testing .NET](#).

In this tutorial, you learn how to:

- Prepare a .NET app for GitHub Actions
- Define action inputs and outputs
- Compose a workflow

Prerequisites

- A [GitHub account](#)
- The [.NET 6 SDK or later](#)
- A .NET integrated development environment (IDE)
 - Feel free to use the [Visual Studio IDE](#)

The intent of the app

The app in this tutorial performs code metric analysis by:

- Scanning and discovering `*.csproj` and `*.vbproj` project files.
- Analyzing the discovered source code within these projects for:
 - Cyclomatic complexity
 - Maintainability index
 - Depth of inheritance
 - Class coupling
 - Number of lines of source code
 - Approximated lines of executable code
- Creating (or updating) a `CODE_METRICS.md` file.

The app is *not* responsible for creating a pull request with the changes to the `CODE_METRICS.md` file. These changes are managed as part of the [workflow composition](#).

References to the source code in this tutorial have portions of the app omitted for brevity. The complete app code is [available on GitHub](#).

Explore the app

The .NET console app uses the `CommandLineParser` NuGet package to parse arguments into the `ActionInputs` object.

```

using System;
using CommandLine;

namespace DotNet.GitHubAction
{
    public class ActionInputs
    {
        string _repositoryName = null!;
        string _branchName = null!;

        public ActionInputs()
        {
            if (Environment.GetEnvironmentVariable("GREETINGS") is { Length: > 0 } greetings)
            {
                Console.WriteLine(greetings);
            }
        }

        [Option('o', "owner",
            Required = true,
            HelpText = "The owner, for example: \"dotnet\". Assign from `github.repository_owner` .")]
        public string Owner { get; set; } = null!;

        [Option('n', "name",
            Required = true,
            HelpText = "The repository name, for example: \"samples\". Assign from `github.repository` .")]
        public string Name
        {
            get => _repositoryName;
            set => ParseAndAssign(value, str => _repositoryName = str);
        }

        [Option('b', "branch",
            Required = true,
            HelpText = "The branch name, for example: \"refs/heads/main\". Assign from `github.ref` .")]
        public string Branch
        {
            get => _branchName;
            set => ParseAndAssign(value, str => _branchName = str);
        }

        [Option('d', "dir",
            Required = true,
            HelpText = "The root directory to start recursive searching from.")]
        public string Directory { get; set; } = null!;

        [Option('w', "workspace",
            Required = true,
            HelpText = "The workspace directory, or repository root directory.")]
        public string WorkspaceDirectory { get; set; } = null!;

        static void ParseAndAssign(string? value, Action<string> assign)
        {
            if (value is { Length: > 0 } && assign is not null)
            {
                assign(value.Split("/")[^1]);
            }
        }
    }
}

```

The preceding action inputs class defines several required inputs for the app to run successfully. The constructor will write the `"GREETINGS"` environment variable value, if one is available in the current execution environment. The `Name` and `Branch` properties are parsed and assigned from the last segment of a `"/"` delimited string.

With the defined action inputs class, focus on the `Program.cs` file.

```

using System;
using System.Linq;
using System.Threading.Tasks;
using CommandLine;
using DotNet.GitHubAction;
using DotNet.GitHubAction.Extensions;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using static CommandLine.Parser;

using IHost host = Host.CreateDefaultBuilder(args)
    .ConfigureServices(_ , services) => services.AddGitHubActionServices()
    .Build();

static TService Get<TService>(IHost host)
    where TService : notnull =>
    host.Services.GetRequiredService<TService>();

var parser = Default.ParseArguments<ActionInputs>(() => new(), args);
parser.WithNotParsed(
    errors =>
{
    Get<ILoggerFactory>(host)
        .CreateLogger("DotNet.GitHubAction.Program")
        .LogError(
            string.Join(
                Environment.NewLine, errors.Select(error => error.ToString())));
    Environment.Exit(2);
});

await parser.WithParsedAsync(options => StartAnalysisAsync(options, host));
await host.RunAsync();

static async Task StartAnalysisAsync(ActionInputs inputs, IHost host)
{
    // Omitted for brevity, here is the preudo code:
    // - Read projects
    // - Calculate code metric analytics
    // - Write the CODE_METRICS.md file
    // - Set the outputs

    var updatedMetrics = true;
    var title = "Updated 2 projects";
    var summary = "Calculated code metrics on two projects.";

    // Do the work here...

    Console.WriteLine($"::set-output name=updated-metrics::{updatedMetrics}");
    Console.WriteLine($"::set-output name=summary-title::{title}");
    Console.WriteLine($"::set-output name=summary-details::{summary}");

    await Task.CompletedTask;

    Environment.Exit(0);
}

```

The `Program` file is simplified for brevity, to explore the full sample source, see [Program.cs](#). The mechanics in place demonstrate the boilerplate code required to use:

- [Top-level statements](#)
- [Generic Host](#)
- [Dependency injection](#)

External project or package references can be used, and registered with dependency injection. The `Get<TService>` is a static local function, which requires the `IHost` instance, and is used to resolve required services. With the `CommandLine.Parser.Default` singleton, the app gets a `parser` instance from the `args`. When the arguments are unable to be parsed, the app exits with a non-zero exit code. For more information, see [Setting exit codes for actions](#).

When the args are successfully parsed, the app was called correctly with the required inputs. In this case, a call to the primary functionality `StartAnalysisAsync` is made.

To write output values, you must follow the format recognized by [GitHub Actions: Setting an output parameter](#).

Prepare the .NET app for GitHub Actions

GitHub Actions support two variations of app development, either

- JavaScript (optionally [TypeScript](#))
- Docker container (any app that runs on [Docker](#))

The virtual environment where the GitHub Action is hosted may or may not have .NET installed. For information about what is preinstalled in the target environment, see [GitHub Actions Virtual Environments](#). While it's possible to run .NET CLI commands from the GitHub Actions workflows, for a more fully functioning .NET-based GitHub Action, we recommend that you containerize the app. For more information, see [Containerize a .NET app](#).

The Dockerfile

A [Dockerfile](#) is a set of instructions to build an image. For .NET applications, the *Dockerfile* usually sits in the root of the directory next to a solution file.

```
# Set the base image as the .NET 6.0 SDK (this includes the runtime)
FROM mcr.microsoft.com/dotnet/sdk:6.0 as build-env

# Copy everything and publish the release (publish implicitly restores and builds)
WORKDIR /app
COPY . .
RUN dotnet publish ./DotNet.GitHubAction/DotNet.GitHubAction.csproj -c Release -o out --no-self-contained

# Label the container
LABEL maintainer="David Pine <david.pine@microsoft.com>"
LABEL repository="https://github.com/dotnet/samples"
LABEL homepage="https://github.com/dotnet/samples"

# Label as GitHub action
LABEL com.github.actions.name="The name of your GitHub Action"
# Limit to 160 characters
LABEL com.github.actions.description="The description of your GitHub Action."
# See branding:
# https://docs.github.com/actions/creating-actions/metadata-syntax-for-github-actions#branding
LABEL com.github.actions.icon="activity"
LABEL com.github.actions.color="orange"

# Relayer the .NET SDK, anew with the build output
FROM mcr.microsoft.com/dotnet/sdk:6.0
COPY --from=build-env /app/out .
ENTRYPOINT [ "dotnet", "/DotNet.GitHubAction.dll" ]
```

NOTE

The .NET app in this tutorial relies on the .NET SDK as part of its functionality. The *Dockerfile* creates a new set of Docker layers, independent from the previous ones. It starts from scratch with the SDK image, and adds the build output from the previous set of layers. For applications that *do not* require the .NET SDK as part of their functionality, they should rely on just the .NET Runtime instead. This greatly reduces the size of the image.

```
FROM mcr.microsoft.com/dotnet/runtime:6.0
```

WARNING

Pay close attention to every step within the *Dockerfile*, as it does differ from the standard *Dockerfile* created from the "add docker support" functionality. In particular, the last few steps vary by not specifying a new `WORKDIR` which would change the path to the app's `ENTRYPOINT`.

The preceding *Dockerfile* steps include:

- Setting the base image from `mcr.microsoft.com/dotnet/sdk:6.0` as the alias `build-env`.
- Copying the contents and publishing the .NET app:
 - The app is published using the `dotnet publish` command.
- Applying labels to the container.
- Relayering the .NET SDK image from `mcr.microsoft.com/dotnet/sdk:6.0`
- Copying the published build output from the `build-env`.
- Defining the entry point, which delegates to `dotnet /DotNet.GitHubAction.dll`.

TIP

The MCR in `mcr.microsoft.com` stands for "Microsoft Container Registry", and is Microsoft's syndicated container catalog from the official Docker hub. For more information, see [Microsoft syndicates container catalog](#).

Caution

If you use a `global.json` file to pin the SDK version, you should explicitly refer to that version in your *Dockerfile*. For example, if you've used `global.json` to pin SDK version `5.0.300`, your *Dockerfile* should use `mcr.microsoft.com/dotnet/sdk:5.0.300`. This prevents breaking the GitHub Actions when a new minor revision is released.

Define action inputs and outputs

In the [Explore the app](#) section, you learned about the `ActionInputs` class. This object represents the inputs for the GitHub Action. For GitHub to recognize that the repository is a GitHub Action, you need to have an `action.yml` file at the root of the repository.

```

name: 'The title of your GitHub Action'
description: 'The description of your GitHub Action'
branding:
  icon: activity
  color: orange
inputs:
  owner:
    description:
      'The owner of the repo. Assign from github.repository_owner. Example, "dotnet".'
      required: true
  name:
    description:
      'The repository name. Example, "samples".'
      required: true
  branch:
    description:
      'The branch name. Assign from github.ref. Example, "refs/heads/main".'
      required: true
  dir:
    description:
      'The root directory to work from. Examples, "path/to/code".'
    required: false
    default: '/github/workspace'
outputs:
  summary-title:
    description:
      'The title of the code metrics action.'
  summary-details:
    description:
      'A detailed summary of all the projects that were flagged.'
updated-metrics:
  description:
    'A boolean value, indicating whether or not the action updated metrics.'
runs:
  using: 'docker'
  image: 'Dockerfile'
  args:
    - '-o'
    - ${{ inputs.owner }}
    - '-n'
    - ${{ inputs.name }}
    - '-b'
    - ${{ inputs.branch }}
    - '-d'
    - ${{ inputs.dir }}

```

The preceding `action.yml` file defines:

- The `name` and `description` of the GitHub Action
- The `branding`, which is used in the [GitHub Marketplace](#) to help more uniquely identify your action
- The `inputs`, which maps one-to-one with the `ActionInputs` class
- The `outputs`, which is written to in the `Program` and used as part of [Workflow composition](#)
- The `runs` node, which tells GitHub that the app is a `docker` application and what arguments to pass to it

For more information, see [Metadata syntax for GitHub Actions](#).

Pre-defined environment variables

With GitHub Actions, you'll get a lot of [environment variables](#) by default. For instance, the variable `GITHUB_REF` will always contain a reference to the branch or tag that triggered the workflow run. `GITHUB_REPOSITORY` has the owner and repository name, for example, `dotnet/docs`.

You should explore the pre-defined environment variables and use them accordingly.

Workflow composition

With the [.NET app containerized](#), and the [action inputs and outputs](#) defined, you're ready to consume the action. GitHub Actions are *not* required to be published in the GitHub Marketplace to be used. Workflows are defined in the `.github/workflows` directory of a repository as YAML files.

```
# The name of the work flow. Badges will use this name
name: '.NET code metrics'

on:
  push:
    branches: [ main ]
    paths:
      - 'github-actions/DotNet.GitHubAction/**'           # run on all changes to this dir
      - '!github-actions/DotNet.GitHubAction/CODE_METRICS.md' # ignore this file
  workflow_dispatch:
    inputs:
      reason:
        description: 'The reason for running the workflow'
        required: true
        default: 'Manual run'

jobs:
  analysis:
    runs-on: ubuntu-latest
    permissions:
      contents: write
      pull-requests: write

    steps:
      - uses: actions/checkout@v2

      - name: 'Print manual run reason'
        if: ${{ github.event_name == 'workflow_dispatch' }}
        run:
          echo 'Reason: ${{ github.event.inputs.reason }}'

      - name: .NET code metrics
        id: dotnet-code-metrics
        uses: dotnet/samples/github-actions/DotNet.GitHubAction@main
        env:
          GREETINGS: 'Hello, .NET developers!' # ${{ secrets.GITHUB_TOKEN }}
        with:
          owner: ${{ github.repository_owner }}
          name: ${{ github.repository }}
          branch: ${{ github.ref }}
          dir: ${{ './github-actions/DotNet.GitHubAction' }}

      - name: Create pull request
        uses: peter-evans/create-pull-request@v3.4.1
        if: ${{ steps.dotnet-code-metrics.outputs.updated-metrics }} == 'true'
        with:
          title: '${{ steps.dotnet-code-metrics.outputs.summary-title }}'
          body: '${{ steps.dotnet-code-metrics.outputs.summary-details }}'
          commit-message: '.NET code metrics, automated pull request.'
```

IMPORTANT

For containerized GitHub Actions, you're required to use `runs-on: ubuntu-latest`. For more information, see [Workflow syntax](#) `jobs.<job_id>.runs-on`.

The preceding workflow YAML file defines three primary nodes:

- The `name` of the workflow. This name is also what's used when creating a [workflow status badge](#).
- The `on` node defines when and how the action is triggered.
- The `jobs` node outlines the various jobs and steps within each job. Individual steps consume GitHub Actions.

For more information, see [Creating your first workflow](#).

Focusing on the `steps` node, the composition is more obvious:

```
steps:
- uses: actions/checkout@v2

- name: 'Print manual run reason'
  if: ${{ github.event_name == 'workflow_dispatch' }}
  run: |
    echo 'Reason: ${{ github.event.inputs.reason }}'

- name: .NET code metrics
  id: dotnet-code-metrics
  uses: dotnet/samples/github-actions/DotNet.GitHubAction@main
  env:
    GREETINGS: 'Hello, .NET developers!' # ${{ secrets.GITHUB_TOKEN }}
  with:
    owner: ${{ github.repository_owner }}
    name: ${{ github.repository }}
    branch: ${{ github.ref }}
    dir: ${{ './github-actions/DotNet.GitHubAction' }}

- name: Create pull request
  uses: peter-evans/create-pull-request@v3.4.1
  if: ${{ steps.dotnet-code-metrics.outputs.updated-metrics }} == 'true'
  with:
    title: '${{ steps.dotnet-code-metrics.outputs.summary-title }}'
    body: '${{ steps.dotnet-code-metrics.outputs.summary-details }}'
    commit-message: '.NET code metrics, automated pull request.'
```

The `jobs.steps` represents the *workflow composition*. Steps are orchestrated such that they're sequential, communicative, and composable. With various GitHub Actions representing steps, each having inputs and outputs, workflows can be composed.

In the preceding steps, you can observe:

1. The repository is [checked out](#).
2. A message is printed to the workflow log, when [manually ran](#).
3. A step identified as `dotnet-code-metrics`:
 - `uses: dotnet/samples/github-actions/DotNet.GitHubAction@main` is the location of the containerized .NET app in this tutorial.
 - `env` creates an environment variable `"GREETING"`, which is printed in the execution of the app.
 - `with` specifies each of the required action inputs.
4. A conditional step, named `Create pull request` runs when the `dotnet-code-metrics` step specifies an output parameter of `updated-metrics` with a value of `true`.

IMPORTANT

GitHub allows for the creation of [encrypted secrets](#). Secrets can be used within workflow composition, using the `${{ secrets.SECRET_NAME }}` syntax. In the context of a GitHub Action, there is a GitHub token that is automatically populated by default: `${{ secrets.GITHUB_TOKEN }}`. For more information, see [Context and expression syntax for GitHub Actions](#).

Put it all together

The [dotnet/samples](#) GitHub repository is home to many .NET sample source code projects, including [the app in this tutorial](#).

The generated [CODE_METRICS.md](#) file is navigable. This file represents the hierarchy of the projects it analyzed. Each project has a top-level section, and an emoji that represents the overall status of the highest cyclomatic complexity for nested objects. As you navigate the file, each section exposes drill-down opportunities with a summary of each area. The markdown has collapsible sections as an added convenience.

The hierarchy progresses from:

- Project file to assembly
- Assembly to namespace
- Namespace to named-type
- Each named-type has a table, and each table has:
 - Links to line numbers for fields, methods, and properties
 - Individual ratings for code metrics

In action

The workflow specifies that `on` a `push` to the `main` branch, the action is triggered to run. When it runs, the [Actions](#) tab in GitHub will report the live log stream of its execution. Here is an example log from the `.NET code metrics` run:

```
build
succeeded 2 days ago in 1m 46s
Q. Search logs ⚙

> ✓ Set up job 17s
> ✓ Build dotnet/samples/github-actions/DotNet.GitHubAction@main 59s
> ✓ Run actions/checkout@v2 7s
> ✓ Print manual run reason 0s
✓ .NET code metrics 7s

1 ► Run dotnet/samples/github-actions/DotNet.GitHubAction@main
10 /usr/bin/docker run -name e4d19bb1b7972d4fedb3859de4be257d74_81e30b --label 5588e4 --workdir /github/workspace --rm -e GREETINGS -e INPUT_OWNER -e INPUT_NAME -e INPUT_BRANCH -e INPUT_DIR -e INPUT_WORKSPACE -e HOME -e GITHUB_REF -e GITHUB_REPOSITORY -e GITHUB_REPOSITORY_OWNER -e GITHUB_RUN_ID -e GITHUB_RUN_NUMBER -e GITHUB_RETENTION_DAYS -e GITHUB_ACTOR -e GITHUB_WORKFLOW -e GITHUB_HEAD_REF -e GITHUB_BASE_REF -e GITHUB_EVENT_NAME -e GITHUB_SERVER_URL -e GITHUB_API_URL -e GITHUB_GRAPHQL_URL -e GITHUB_WORKSPACE -e GITHUB_ACTION -e GITHUB_EVENT_PATH -e GITHUB_ACTION_REPOSITORY -e GITHUB_ACTION_REF -e GITHUB_PATH -e GITHUB_ENV -e RUNNER_OS -e RUNNER_TOOL_CACHE -e RUNNER_TEMP -e RUNNER_WORKSPACE -e ACTIONS_RUNTIME_URL -e ACTIONS_RUNTIME_TOKEN -e ACTIONS_CACHE_URL -e GITHUB_ACTIONS=true -e CI=true -v "/var/run/docker.sock":"/var/run/docker.sock" -v "/home/runner/work/_temp/_github_home":"/github/home" -v "/home/runner/work/_temp/_github_workflow":"/github/workflow" -v "/home/runner/work/_temp/_runner_file_commands":"/github/file_commands" -v "/home/runner/work/samples/samples":"/github/workspace" 5588e4:d19bb1b7972d4fedb3859de4be257d74 "-o" "dotnet" "-n" "dotnet/samples" "-b" "refs/heads/main" "-d" ".github/actions/DotNet.GitHubAction" "-w" "/github/workspace"
11 Hello, .NET developers!
12 Hello, .NET developers!
13 info: DotNet.GitHubAction.Analyzers.ProjectMetricDataAnalyzer[0]
14 Computing analytics on /github/workspace/github-actions/.DotNet.GitHubAction/DotNet.CodeAnalysis/DotNet.CodeAnalysis.csproj.
15 info: DotNet.GitHubAction.Analyzers.ProjectMetricDataAnalyzer[0]
16 Computing analytics on /github/workspace/github-actions/.DotNet.GitHubAction/DotNet.GitHubAction/DotNet.GitHubAction.csproj.
17 info: StartAnalysisAsync[0]
18 Updating CODE_METRICS.md markdown file with latest code metric data.
19 - */github/workspace/github-actions/.DotNet.GitHubAction/DotNet.CodeAnalysis/DotNet.CodeAnalysis.csproj*
20 - */github/workspace/github-actions/.DotNet.GitHubAction/DotNet.GitHubAction/DotNet.GitHubAction.csproj*
21

> ✓ Create pull request 16s
> ✓ Post Run actions/checkout@v2 0s
> ✓ Complete job 0s
```

Performance improvements

If you followed along the sample, you might have noticed that every time this action is used, it will do a `docker`

build for that image. So, every trigger is faced with some time to build the container before running it. Before releasing your GitHub Actions to the marketplace, you should:

1. (automatically) Build the Docker image
2. Push the docker image to the GitHub Container Registry (or any other public container registry)
3. Change the action to not build the image, but to use an image from a public registry.

```
# Rest of action.yml content removed for readability
# using Dockerfile
runs:
  using: 'docker'
  image: 'Dockerfile' # Change this line
# using container image from public registry
runs:
  using: 'docker'
  image: 'docker://ghcr.io/some-user/some-registry' # Starting with docker:// is important!!
```

For more information, see [GitHub Docs: Working with the Container registry](#).

See also

- [.NET Generic Host](#)
- [Dependency injection in .NET](#)
- [DevOps for ASP.NET Core Developers](#)
- [Code metrics values](#)
- [Open-source GitHub Action build in .NET](#) with a [workflow](#) for building and pushing the docker image automatically.

Next steps

[.NET GitHub Actions sample code](#)

Quickstart: Create a build validation GitHub workflow

9/20/2022 • 3 minutes to read • [Edit Online](#)

In this quickstart, you will learn how to create a GitHub workflow to validate the compilation of your .NET source code in GitHub. Compiling your .NET code is one of the most basic validation steps that you can take to help ensure the quality of updates to your code. If code doesn't compile (or build), it's an easy deterrent and should be a clear sign that the code needs to be fixed.

Prerequisites

- A [GitHub account](#).
- A .NET source code repository.

Create a workflow file

In the GitHub repository, add a new YAML file to the `.github/workflows` directory. Choose a meaningful file name, something that will clearly indicate what the workflow is intended to do. For more information, see [Workflow file](#).

IMPORTANT

GitHub requires that workflow composition files to be placed within the `.github/workflows` directory.

Workflow files typically define a composition of one or more GitHub Action via the `jobs.<job_id>/steps[*]`. For more information, see, [Workflow syntax for GitHub Actions](#).

Create a new file named `build-validation.yml`, copy and paste the following YML contents into it:

```

name: build

on:
  push:
  pull_request:
    branches: [ main ]
    paths:
      - '**.cs'
      - '**.csproj'

env:
  DOTNET_VERSION: '5.0.301' # The .NET SDK version to use

jobs:
  build:

    name: build-${{matrix.os}}
    runs-on: ${{ matrix.os }}
    strategy:
      matrix:
        os: [ubuntu-latest, windows-latest, macOS-latest]

    steps:
      - uses: actions/checkout@v2
      - name: Setup .NET Core
        uses: actions/setup-dotnet@v1
        with:
          dotnet-version: ${{ env.DOTNET_VERSION }}

      - name: Install dependencies
        run: dotnet restore

      - name: Build
        run: dotnet build --configuration Release --no-restore

```

In the preceding workflow composition:

- The `name: build` defines the name, "build" will appear in workflow status badges.

```
name: build
```

- The `on` node signifies the events that trigger the workflow:

```

on:
  push:
  pull_request:
    branches: [ main ]
    paths:
      - '**.cs'
      - '**.csproj'

```

- Triggered when a `push` or `pull_request` occurs on the `main` branch where any files changed ending with the `.cs` or `.csproj` file extensions.
- The `env` node defines named environment variables (env var).

```

env:
  DOTNET_VERSION: '5.0.301' # The .NET SDK version to use

```

- The environment variable `DOTNET_VERSION` is assigned the value `'5.0.301'`. The environment variable

is later referenced to specify the `dotnet-version` of the `actions/setup-dotnet@v1` GitHub Action.

- The `jobs` node builds out the steps for the workflow to take.

```
jobs:  
  build:  
  
    name: build-${{matrix.os}}  
    runs-on: ${{ matrix.os }}  
    strategy:  
      matrix:  
        os: [ubuntu-latest, windows-latest, macOS-latest]  
  
    steps:  
      - uses: actions/checkout@v2  
      - name: Setup .NET Core  
        uses: actions/setup-dotnet@v1  
        with:  
          dotnet-version: ${{ env.DOTNET_VERSION }}  
  
      - name: Install dependencies  
        run: dotnet restore  
  
      - name: Build  
        run: dotnet build --configuration Release --no-restore
```

- There is a single job, named `build-<os>` where the `<os>` is the operating system name from the `strategy/matrix`. The `name` and `runs-on` elements are dynamic for each value in the `matrix/os`. This will run on the latest versions of Ubuntu, Windows, and macOS.
- The `actions/setup-dotnet@v1` GitHub Action is required to set up the .NET SDK with the specified version from the `DOTNET_VERSION` environment variable.
- (Optional) Additional steps may be required, depending on your .NET workload. They're omitted from this example, but you may need additional tools installed to build your apps.
- For example, when building an ASP.NET Core Blazor WebAssembly application with Ahead-of-Time (AoT) compilation you'd install the corresponding workload before running `restore/build/publish` operations.

```
- name: Install WASM Tools Workload  
run: dotnet workload install wasm-tools
```

For more information on .NET workloads, see [dotnet workload install](#).

- The `dotnet restore` command is called.
- The `dotnet build` command is called.

In this case, think of a workflow file as a composition that represents the various steps to build an application. Many [.NET CLI commands](#) are available, most of which could be used in the context of a GitHub Action.

Create a workflow status badge

It's common nomenclature for GitHub repositories to have a `README.md` file at the root of the repository directory. Likewise, it's nice to report the latest status for various workflows. All workflows can generate a status badge, which are visually appealing within the `README.md` file. To add the workflow status badge:

1. From the GitHub repository select the **Actions** navigation option.

2. All repository workflows are displayed on the left-side, select the desired workflow and the ellipsis (...) button.

- The ellipsis (...) button expands the menu options for the selected workflow.

3. Select the **Create status badge** menu option.

The screenshot shows the GitHub interface for the repository `dotnet/docs`. On the left, there's a sidebar with a list of workflows: All workflows, MSDocs build verifier, Markdownlinter, OPS status checker, Snippets 5000, bc-notification, and generate what's new article. The workflow `target supported version` is selected and highlighted with a red box. On the right, the workflow details are displayed: `target supported version`, `version-sweep.yml`. Below this, it says "4 workflow runs" and "This workflow has a `workflow_dispatch` event trigger". Four workflow runs are listed, each with a green checkmark, indicating they have passed. The most recent run was 29 days ago. A context menu is open over the first run, with the "Create status badge" option highlighted by a red box. Other options in the menu include "Disable workflow" and "Run workflow".

4. Select the **Copy status badge Markdown** button.

The screenshot shows the "Create status badge" dialog box overlaid on the GitHub repository page. The dialog has the title "Create status badge" and contains the workflow name "target supported version" and the status "passing". It also includes fields for "Branch" (set to "Default branch") and "Event" (set to "Default"). Below these, the generated Markdown code is shown: `[![target supported version](https://github.com/dotnet/docs/actions/workflow/s/version-sweep.yml/badge.svg)](https://github.com/dotnet/docs/actions/workflows/version-sweep.yml)`. At the bottom of the dialog, a green button labeled "Copy status badge Markdown" is highlighted with a red box. The background of the GitHub page shows the same list of workflows and their runs as in the previous screenshot.

5. Paste the Markdown into the `README.md` file, save the file, commit and push the changes.

For more, see [Adding a workflow status badge](#).

Example build workflow status badge

PASSING	FAILING	NO STATUS

See also

- [dotnet restore](#)
- [dotnet build](#)
- [actions/checkout](#)
- [actions/setup-dotnet](#)

Next steps

[Quickstart: Create a .NET test GitHub workflow](#)

Quickstart: Create a test validation GitHub workflow

9/20/2022 • 3 minutes to read • [Edit Online](#)

In this quickstart, you will learn how to create a GitHub workflow to test your .NET source code. Automatically testing your .NET code within GitHub is referred to as continuous integration (CI), where pull requests or changes to the source trigger workflows to exercise. Along with [building the source code](#), testing ensures that the compiled source code functions as the author intended. More often than not, unit tests serve as immediate feedback-loop to help ensure the validity of changes to source code.

Prerequisites

- A [GitHub account](#).
- A .NET source code repository.

Create a workflow file

In the GitHub repository, add a new YAML file to the `.github/workflows` directory. Choose a meaningful file name, something that will clearly indicate what the workflow is intended to do. For more information, see [Workflow file](#).

IMPORTANT

GitHub requires that workflow composition files to be placed within the `.github/workflows` directory.

Workflow files typically define a composition of one or more GitHub Action via the `jobs.<job_id>/steps[*]`. For more information, see, [Workflow syntax for GitHub Actions](#).

Create a new file named `build-and-test.yml`, copy and paste the following YML contents into it:

```

name: build and test

on:
  push:
  pull_request:
    branches: [ main ]
    paths:
      - '**.cs'
      - '**.csproj'

env:
  DOTNET_VERSION: '5.0.301' # The .NET SDK version to use

jobs:
  build-and-test:

    name: build-and-test-${{matrix.os}}
    runs-on: ${{ matrix.os }}
    strategy:
      matrix:
        os: [ubuntu-latest, windows-latest, macOS-latest]

    steps:
      - uses: actions/checkout@v2
      - name: Setup .NET Core
        uses: actions/setup-dotnet@v1
        with:
          dotnet-version: ${{ env.DOTNET_VERSION }}

      - name: Install dependencies
        run: dotnet restore

      - name: Build
        run: dotnet build --configuration Release --no-restore

      - name: Test
        run: dotnet test --no-restore --verbosity normal

```

In the preceding workflow composition:

- The `name: build and test` defines the name, "build and test" will appear in workflow status badges.

```
name: build and test
```

- The `on` node signifies the events that trigger the workflow:

```

on:
  push:
  pull_request:
    branches: [ main ]
    paths:
      - '**.cs'
      - '**.csproj'

```

- Triggered when a `push` or `pull_request` occurs on the `main` branch where any files changed ending with the `.cs` or `.csproj` file extensions.

- The `env` node defines named environment variables (env var).

```

env:
  DOTNET_VERSION: '5.0.301' # The .NET SDK version to use

```

- The environment variable `DOTNET_VERSION` is assigned the value `'5.0.301'`. The environment variable is later referenced to specify the `dotnet-version` of the `actions/setup-dotnet@v1` GitHub Action.
- The `jobs` node builds out the steps for the workflow to take.

```

jobs:
  build-and-test:

    name: build-and-test-${{matrix.os}}
    runs-on: ${{ matrix.os }}
    strategy:
      matrix:
        os: [ubuntu-latest, windows-latest, macOS-latest]

    steps:
      - uses: actions/checkout@v2
      - name: Setup .NET Core
        uses: actions/setup-dotnet@v1
        with:
          dotnet-version: ${{ env.DOTNET_VERSION }}

      - name: Install dependencies
        run: dotnet restore

      - name: Build
        run: dotnet build --configuration Release --no-restore

      - name: Test
        run: dotnet test --no-restore --verbosity normal
  
```

- There is a single job, named `build-<os>` where the `<os>` is the operating system name from the `strategy/matrix`. The `name` and `runs-on` elements are dynamic for each value in the `matrix/os`. This will run on the latest versions of Ubuntu, Windows, and macOS.
- The `actions/setup-dotnet@v1` GitHub Action is used to setup the .NET SDK with the specified version from the `DOTNET_VERSION` environment variable.
- The `dotnet restore` command is called.
- The `dotnet build` command is called.
- The `dotnet test` command is called.

Create a workflow status badge

It's common nomenclature for GitHub repositories to have a `README.md` file at the root of the repository directory. Likewise, it's nice to report the latest status for various workflows. All workflows can generate a status badge, which are visually appealing within the `README.md` file. To add the workflow status badge:

1. From the GitHub repository select the **Actions** navigation option.
2. All repository workflows are displayed on the left-side, select the desired workflow and the ellipsis (...) button.
 - The ellipsis (...) button expands the menu options for the selected workflow.
3. Select the **Create status badge** menu option.

The screenshot shows the GitHub Actions workflow runs page for the 'target supported version' workflow. On the left, a sidebar lists various workflows, with 'target supported version' highlighted and selected. The main area displays four workflow runs, each with a green checkmark indicating success. A context menu is open over the first run, showing options like 'Create status badge', 'Disable workflow', and 'Run workflow'. The 'Create status badge' button is highlighted with a red box.

4. Select the **Copy status badge Markdown** button.

The screenshot shows the 'Create status badge' modal. In the 'Event' dropdown, 'Default' is selected. The 'Status' dropdown has 'passing' selected. Below the dropdowns, the copied Markdown code is pasted into the text area:

```
[![target supported version](https://github.com/dotnet/docs/actions/workflows/version-sweep.yml/badge.svg)](https://github.com/dotnet/docs/actions/workflows/version-sweep.yml)
```

The 'Copy status badge Markdown' button is highlighted with a red box.

5. Paste the Markdown into the *README.md* file, save the file, commit and push the changes.

For more, see [Adding a workflow status badge](#).

Example test workflow status badge

PASSING	FAILING	NO STATUS

See also

- [dotnet restore](#)
- [dotnet build](#)

- [dotnet test](#)
- [Unit testing .NET apps](#)
- [actions/checkout](#)
- [actions/setup-dotnet](#)

Next steps

[Quickstart: Create a GitHub workflow to publish your .NET app](#)

Quickstart: Create a GitHub workflow to publish an app

9/20/2022 • 4 minutes to read • [Edit Online](#)

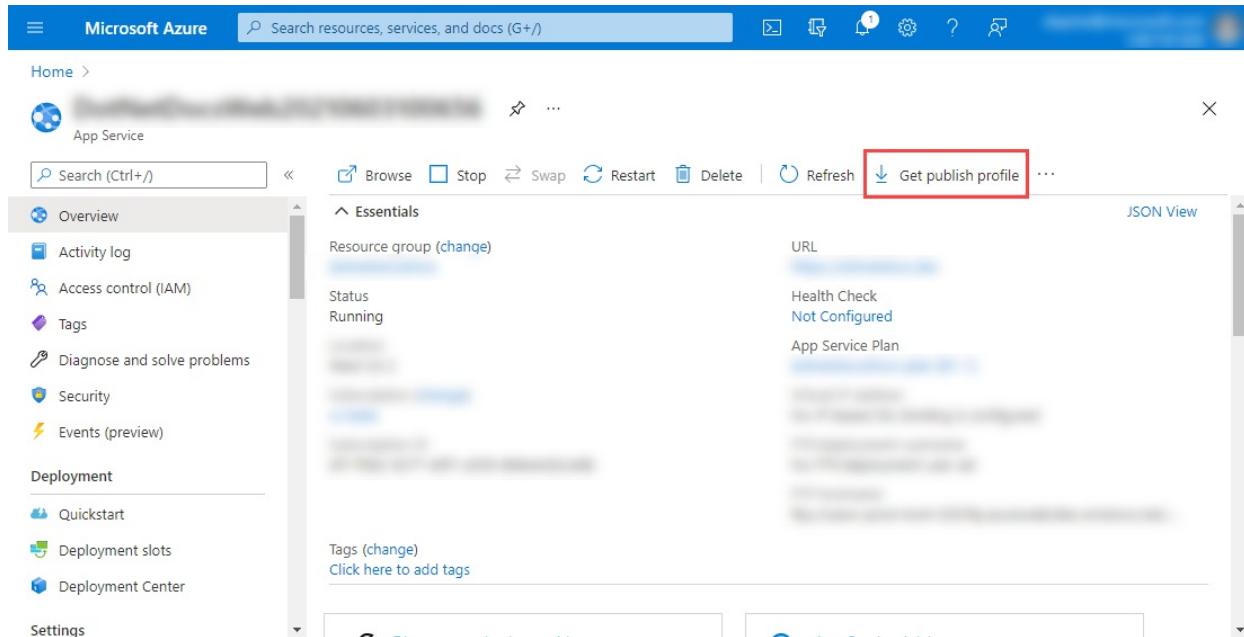
In this quickstart, you will learn how to create a GitHub workflow to publish your .NET app from source code. Automatically publishing your .NET app from GitHub to a destination is referred to as a continuous deployment (CD). There are many possible destinations to publish an application, in this quickstart you'll publish to Azure.

Prerequisites

- A [GitHub account](#).
- A .NET source code repository.
- An Azure account with an active subscription. [Create an account for free](#).
- An ASP.NET Core web app.
- An Azure App Service resource.

Add publish profile

To publish the app to Azure, open the Azure portal for the App Service instance of the application. In the resource **Overview**, select **Get publish profile** and save the `*.PublishSetting` file locally.



The screenshot shows the Azure portal interface for managing an App Service. On the left, there's a navigation sidebar with links like Home, Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Security, Events (preview), Deployment (Quickstart, Deployment slots, Deployment Center), and Settings. The main content area is titled 'App Service' and shows an 'Overview' card with details such as Resource group (changed), Status (Running), URL, Health Check (Not Configured), and App Service Plan. Below the card, there are sections for Tags (change) and Click here to add tags. At the top right of the main area, there are several actions: Browse, Stop, Swap, Restart, Delete, Refresh, and Get publish profile (which is highlighted with a red box). A 'JSON View' link is also present. The top navigation bar includes a search bar, a menu icon, and other common portal navigation icons.

WARNING

The publish profile contains sensitive information, such as credentials for accessing your Azure App Service resource. This information should always be treated very carefully.

In the GitHub repository, navigate to **Settings** and select **Secrets** from the left navigation menu. Select **New repository secret**, to add a new secret.

Actions secrets / New secret

Name

Value

Add secret

Enter `AZURE_PUBLISH_PROFILE` as the **Name**, and paste the XML content from the publish profile into the **Value** text area. Select **Add secret**. For more information, see [Encrypted secrets](#).

Create a workflow file

In the GitHub repository, add a new YAML file to the `.github/workflows` directory. Choose a meaningful file name, something that will clearly indicate what the workflow is intended to do. For more information, see [Workflow file](#).

IMPORTANT

GitHub requires that workflow composition files to be placed within the `.github/workflows` directory.

Workflow files typically define a composition of one or more GitHub Action via the `jobs.<job_id>/steps[*]`. For more information, see, [Workflow syntax for GitHub Actions](#).

Create a new file named `publish-app.yml`, copy and paste the following YML contents into it:

```

name: publish

on:
  push:
    branches: [ production ]

env:
  AZURE_WEBAPP_NAME: DotNetWeb
  AZURE_WEBAPP_PACKAGE_PATH: '.' # Set this to the path to your web app project, defaults to the repository root
  DOTNET_VERSION: '5.0.301' # The .NET SDK version to use

jobs:
  publish:

    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v2
      - name: Setup .NET Core
        uses: actions/setup-dotnet@v1
        with:
          dotnet-version: ${{ env.DOTNET_VERSION }}

      - name: Install dependencies
        run: dotnet restore

      - name: Build
        run: |
          cd DotNet.WebApp
          dotnet build --configuration Release --no-restore
          dotnet publish -c Release -o ..//dotnet-webapp -r linux-x64 --self-contained true /p:UseAppHost=true

      - name: Test
        run: |
          cd DotNet.WebApp.Tests
          dotnet test --no-restore --verbosity normal

      - uses: azure/webapps-deploy@v2
        name: Deploy
        with:
          app-name: ${{ env.AZURE_WEBAPP_NAME }}
          publish-profile: ${{ secrets.AZURE_PUBLISH_PROFILE }}
          package: '${{ env.AZURE_WEBAPP_PACKAGE_PATH }}//dotnet-webapp'

```

In the preceding workflow composition:

- The `name: publish` defines the name, "publish" will appear in workflow status badges.

```
name: publish
```

- The `on` node signifies the events that trigger the workflow:

```
on:
  push:
    branches: [ production ]
```

- Triggered when a `push` occurs on the `production` branch.
- The `env` node defines named environment variables (env var).

```

env:
  AZURE_WEBAPP_NAME: DotNetWeb
  AZURE_WEBAPP_PACKAGE_PATH: '.' # Set this to the path to your web app project, defaults to the
repository root:
  DOTNET_VERSION: '5.0.301' # The .NET SDK version to use

```

- The environment variable `AZURE_WEBAPP_NAME` is assigned the value `DotNetWeb`.
- The environment variable `AZURE_WEBAPP_PACKAGE_PATH` is assigned the value `'.'`.
- The environment variable `DOTNET_VERSION` is assigned the value `'5.0.301'`. The environment variable is later referenced to specify the `dotnet-version` of the `actions/setup-dotnet@v1` GitHub Action.
- The `jobs` node builds out the steps for the workflow to take.

```

jobs:
  publish:
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v2
      - name: Setup .NET Core
        uses: actions/setup-dotnet@v1
        with:
          dotnet-version: ${{ env.DOTNET_VERSION }}

      - name: Install dependencies
        run: dotnet restore

      - name: Build
        run: |
          cd DotNet.WebApp
          dotnet build --configuration Release --no-restore
          dotnet publish -c Release -o ../dotnet-webapp -r linux-x64 --self-contained true
/p:UseAppHost=true
      - name: Test
        run: |
          cd DotNet.WebApp.Tests
          dotnet test --no-restore --verbosity normal

      - uses: azure/webapps-deploy@v2
        name: Deploy
        with:
          app-name: ${{ env.AZURE_WEBAPP_NAME }}
          publish-profile: ${{ secrets.AZURE_PUBLISH_PROFILE }}
          package: '${{ env.AZURE_WEBAPP_PACKAGE_PATH }}/dotnet-webapp'

```

- There is a single job, named `publish` that will run on the latest version of Ubuntu.
- The `actions/setup-dotnet@v1` GitHub Action is used to set up the .NET SDK with the specified version from the `DOTNET_VERSION` environment variable.
- The `dotnet restore` command is called.
- The `dotnet build` command is called.
- The `dotnet publish` command is called.
- The `dotnet test` command is called.
- The `azure/webapps-deploy@v2` GitHub Action deploys the app with the given `publish-profile` and `package`.
- The `publish-profile` is assigned from the `AZURE_PUBLISH_PROFILE` repository secret.

Create a workflow status badge

It's common nomenclature for GitHub repositories to have a *README.md* file at the root of the repository directory. Likewise, it's nice to report the latest status for various workflows. All workflows can generate a status badge, which are visually appealing within the *README.md* file. To add the workflow status badge:

1. From the GitHub repository select the **Actions** navigation option.
2. All repository workflows are displayed on the left-side, select the desired workflow and the ellipsis (...) button.
 - The ellipsis (...) button expands the menu options for the selected workflow.
3. Select the **Create status badge** menu option.

The screenshot shows the GitHub Actions interface for the repository 'dotnet/docs'. On the left, there's a sidebar with a list of workflows: 'MSDocs build verifier', 'Markdownlint', 'OPS status checker', 'Snippets 5000', 'bc-notification', and 'generate what's new article'. Below this is a red box highlighting the 'target supported version' workflow. On the right, the main area displays the 'target supported version' workflow with four runs listed. A context menu is open over the third run, with the 'Create status badge' option highlighted by a red box. Other options in the menu include 'Disable workflow' and 'Run workflow'.

4. Select the **Copy status badge Markdown** button.

The screenshot shows the GitHub Actions interface for the repository 'dotnet/docs'. The 'target supported version' workflow is selected and highlighted with a red box. A modal window titled 'Create status badge' is open, showing the workflow name 'target supported version' and its status 'passing'. It also includes fields for 'Branch' (set to 'Default branch') and 'Event' (set to 'Default'). Below these fields is a code editor containing the Markdown for the status badge. A green button labeled 'Copy status badge Markdown' is highlighted with a red box. The background shows the list of workflow runs from the previous screenshot.

5. Paste the Markdown into the *README.md* file, save the file, commit and push the changes.

For more, see [Adding a workflow status badge](#).

Example publish workflow status badge

PASSING	FAILING	NO STATUS
publish passing	publish failing	publish no status

See also

- [dotnet restore](#)
- [dotnet build](#)
- [dotnet test](#)
- [dotnet publish](#)

Next steps

[Quickstart: Create a CodeQL GitHub workflow](#)

Quickstart: Create a security scan GitHub workflow

9/20/2022 • 3 minutes to read • [Edit Online](#)

In this quickstart, you will learn how to create a CodeQL GitHub workflow to automate the discovery of vulnerabilities in your .NET codebase.

In CodeQL, code is treated as data. Security vulnerabilities, bugs, and other errors are modeled as queries that can be executed against databases extracted from code.

— [GitHub CodeQL: About](#)

Prerequisites

- A [GitHub account](#).
- A .NET source code repository.

Create a workflow file

In the GitHub repository, add a new YAML file to the `.github/workflows` directory. Choose a meaningful file name, something that will clearly indicate what the workflow is intended to do. For more information, see [Workflow file](#).

IMPORTANT

GitHub requires that workflow composition files to be placed within the `.github/workflows` directory.

Workflow files typically define a composition of one or more GitHub Action via the `jobs.<job_id>/steps[*]`. For more information, see, [Workflow syntax for GitHub Actions](#).

Create a new file named `codeql-analysis.yml`, copy and paste the following YML contents into it:

```

name: "CodeQL"

on:
  push:
    branches: [main]
    paths:
      - '**.cs'
      - '**.csproj'
  pull_request:
    branches: [main]
    paths:
      - '**.cs'
      - '**.csproj'
  schedule:
    - cron: '0 8 * * 4'

jobs:
  analyze:
    name: analyze
    runs-on: ubuntu-latest

    strategy:
      fail-fast: false
      matrix:
        language: ['csharp']

    steps:
      - name: Checkout repository
        uses: actions/checkout@v2
        with:
          fetch-depth: 2

      - run: git checkout HEAD^2
        if: ${{ github.event_name == 'pull_request' }}

      - name: Initialize CodeQL
        uses: github/codeql-action/init@v1
        with:
          languages: ${{ matrix.language }}

      - name: Autobuild
        uses: github/codeql-action/autobuild@v1

      - name: Perform CodeQL Analysis
        uses: github/codeql-action/analyze@v1

```

In the preceding workflow composition:

- The `name: CodeQL` defines the name, "CodeQL" will appear in workflow status badges.

```
name: "CodeQL"
```

- The `on` node signifies the events that trigger the workflow:

```

on:
  push:
    branches: [main]
    paths:
      - '**.cs'
      - '**.csproj'
  pull_request:
    branches: [main]
    paths:
      - '**.cs'
      - '**.csproj'
  schedule:
    - cron: '0 8 * * 4'

```

- Triggered when a `push` or `pull_request` occurs on the `main` branch where any files changed ending with the `.cs` or `.csproj` file extensions.
- As a cron job (on a schedule) — to run at 8:00 UTC every Thursday.
- The `jobs` node builds out the steps for the workflow to take.

```

jobs:
  analyze:
    name: analyze
    runs-on: ubuntu-latest

    strategy:
      fail-fast: false
      matrix:
        language: ['csharp']

    steps:
      - name: Checkout repository
        uses: actions/checkout@v2
        with:
          fetch-depth: 2

      - run: git checkout HEAD^2
        if: ${{ github.event_name == 'pull_request' }}

      - name: Initialize CodeQL
        uses: github/codeql-action/init@v1
        with:
          languages: ${{ matrix.language }}

      - name: Autobuild
        uses: github/codeql-action/autobuild@v1

      - name: Perform CodeQL Analysis
        uses: github/codeql-action/analyze@v1

```

- There is a single job, named `analyze` that will run on the latest version of Ubuntu.
- The `strategy` defines C# as the `language`.
- The `github/codeql-action/init@v1` GitHub Action is used to initialize CodeQL.
- The `github/codeql-action/autobuild@v1` GitHub Action builds the .NET project.
- The `github/codeql-action/analyze@v1` GitHub Action performs the CodeQL analysis.

For more information, see [GitHub Actions: Configure code scanning](#).

Create a workflow status badge

It's common nomenclature for GitHub repositories to have a *README.md* file at the root of the repository directory. Likewise, it's nice to report the latest status for various workflows. All workflows can generate a status badge, which are visually appealing within the *README.md* file. To add the workflow status badge:

1. From the GitHub repository select the **Actions** navigation option.
2. All repository workflows are displayed on the left-side, select the desired workflow and the ellipsis (...) button.
 - The ellipsis (...) button expands the menu options for the selected workflow.
3. Select the **Create status badge** menu option.

The screenshot shows the GitHub Actions interface for the repository 'dotnet/docs'. On the left, there's a sidebar with a list of workflows: 'MSDocs build verifier', 'Markdownlint', 'OPS status checker', 'Snippets 5000', 'bc-notification', and 'generate what's new article'. Below this is a red box highlighting the 'target supported version' workflow. On the right, the main area displays the 'target supported version' workflow with four runs listed. A context menu is open over the third run, with the 'Create status badge' option highlighted by a red box. Other options in the menu include 'Disable workflow' and 'Run workflow'.

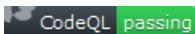
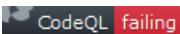
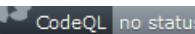
4. Select the **Copy status badge Markdown** button.

The screenshot shows the GitHub Actions interface for the repository 'dotnet/docs'. The 'target supported version' workflow is selected and highlighted with a red box. A modal window titled 'Create status badge' is open, showing the workflow name 'target supported version' and its status 'passing'. It also includes fields for 'Branch' (set to 'Default branch') and 'Event' (set to 'Default'). Below these fields is a code editor containing the Markdown for the status badge. A green button labeled 'Copy status badge Markdown' is highlighted with a red box. The background shows the list of workflow runs from the previous screenshot.

5. Paste the Markdown into the *README.md* file, save the file, commit and push the changes.

For more, see [Adding a workflow status badge](#).

Example CodeQL workflow status badge

PASSING	FAILING	NO STATUS
 CodeQL passing	 CodeQL failing	 CodeQL no status

See also

- [Secure coding guidelines](#)
- [actions/checkout](#)
- [actions/setup-dotnet](#)

Next steps

[Tutorial: Create a GitHub Action with .NET](#)

Common Type System & Common Language Specification

9/20/2022 • 2 minutes to read • [Edit Online](#)

Again, two terms that are freely used in the .NET world, they actually are crucial to understand how a .NET implementation enables multi-language development and to understand how it works.

Common Type System

To start from the beginning, remember that a .NET implementation is *language agnostic*. This doesn't just mean that a programmer can write their code in any language that can be compiled to IL. It also means that they need to be able to interact with code written in other languages that are able to be used on a .NET implementation.

In order to do this transparently, there has to be a common way to describe all supported types. This is what the Common Type System (CTS) is in charge of doing. It was made to do several things:

- Establish a framework for cross-language execution.
- Provide an object-oriented model to support implementing various languages on a .NET implementation.
- Define a set of rules that all languages must follow when it comes to working with types.
- Provide a library that contains the basic primitive types that are used in application development (such as, `Boolean`, `Byte`, `Char` etc.)

CTS defines two main kinds of types that should be supported: reference and value types. Their names point to their definitions.

Reference types' objects are represented by a reference to the object's actual value; a reference here is similar to a pointer in C/C++. It simply refers to a memory location where the objects' values are. This has a profound impact on how these types are used. If you assign a reference type to a variable and then pass that variable into a method, for instance, any changes to the object will be reflected on the main object; there is no copying.

Value types are the opposite, where the objects are represented by their values. If you assign a value type to a variable, you are essentially copying a value of the object.

CTS defines several categories of types, each with their specific semantics and usage:

- Classes
- Structures
- Enums
- Interfaces
- Delegates

CTS also defines all other properties of the types, such as access modifiers, what are valid type members, how inheritance and overloading works and so on. Unfortunately, going deep into any of those is beyond the scope of an introductory article such as this, but you can consult [More resources](#) section at the end for links to more in-depth content that covers these topics.

Common Language Specification

To enable full interoperability scenarios, all objects that are created in code must rely on some commonality in the languages that are consuming them (are their *callers*). Since there are numerous different languages, .NET

has specified those commonalities in something called the **Common Language Specification** (CLS). CLS defines a set of features that are needed by many common applications. It also provides a sort of recipe for any language that is implemented on top of .NET on what it needs to support.

CLS is a subset of the CTS. This means that all of the rules in the CTS also apply to the CLS, unless the CLS rules are more strict. If a component is built using only the rules in the CLS, that is, it exposes only the CLS features in its API, it is said to be **CLS-compliant**. For instance, the `<framework-librares>` are CLS-compliant precisely because they need to work across all of the languages that are supported on .NET.

You can consult the documents in the [More Resources](#) section below to get an overview of all the features in the CLS.

More resources

- [Common Type System](#)
- [Common Language Specification](#)

Common type system

9/20/2022 • 23 minutes to read • [Edit Online](#)

The common type system defines how types are declared, used, and managed in the common language runtime, and is also an important part of the runtime's support for cross-language integration. The common type system performs the following functions:

- Establishes a framework that helps enable cross-language integration, type safety, and high-performance code execution.
- Provides an object-oriented model that supports the complete implementation of many programming languages.
- Defines rules that languages must follow, which helps ensure that objects written in different languages can interact with each other.
- Provides a library that contains the primitive data types (such as [Boolean](#), [Byte](#), [Char](#), [Int32](#), and [UInt64](#)) used in application development.

Types in .NET

All types in .NET are either value types or reference types.

Value types are data types whose objects are represented by the object's actual value. If an instance of a value type is assigned to a variable, that variable is given a fresh copy of the value.

Reference types are data types whose objects are represented by a reference (similar to a pointer) to the object's actual value. If a reference type is assigned to a variable, that variable references (points to) the original value. No copy is made.

The common type system in .NET supports the following five categories of types:

- [Classes](#)
- [Structures](#)
- [Enumerations](#)
- [Interfaces](#)
- [Delegates](#)

Classes

A class is a reference type that can be derived directly from another class and that is derived implicitly from [System.Object](#). The class defines the operations that an object (which is an instance of the class) can perform (methods, events, or properties) and the data that the object contains (fields). Although a class generally includes both definition and implementation (unlike interfaces, for example, which contain only definition without implementation), it can have one or more members that have no implementation.

The following table describes some of the characteristics that a class may have. Each language that supports the runtime provides a way to indicate that a class or class member has one or more of these characteristics. However, individual programming languages that target .NET may not make all these characteristics available.

CHARACTERISTIC	DESCRIPTION
sealed	Specifies that another class cannot be derived from this type.
implements	Indicates that the class uses one or more interfaces by providing implementations of interface members.
abstract	Indicates that the class cannot be instantiated. To use it, you must derive another class from it.
inherits	Indicates that instances of the class can be used anywhere the base class is specified. A derived class that inherits from a base class can use the implementation of any public members provided by the base class, or the derived class can override the implementation of the public members with its own implementation.
exported or not exported	Indicates whether a class is visible outside the assembly in which it is defined. This characteristic applies only to top-level classes and not to nested classes.

NOTE

A class can also be nested in a parent class or structure. Nested classes also have member characteristics. For more information, see [Nested Types](#).

Class members that have no implementation are abstract members. A class that has one or more abstract members is itself abstract; new instances of it cannot be created. Some languages that target the runtime let you mark a class as abstract even if none of its members are abstract. You can use an abstract class when you want to encapsulate a basic set of functionality that derived classes can inherit or override when appropriate. Classes that are not abstract are referred to as concrete classes.

A class can implement any number of interfaces, but it can inherit from only one base class in addition to [System.Object](#), from which all classes inherit implicitly. All classes must have at least one constructor, which initializes new instances of the class. If you do not explicitly define a constructor, most compilers will automatically provide a parameterless constructor.

Structures

A structure is a value type that derives implicitly from [System.ValueType](#), which in turn is derived from [System.Object](#). A structure is useful for representing values whose memory requirements are small, and for passing values as by-value parameters to methods that have strongly typed parameters. In .NET, all primitive data types ([Boolean](#), [Byte](#), [Char](#), [DateTime](#), [Decimal](#), [Double](#), [Int16](#), [Int32](#), [Int64](#), [SByte](#), [Single](#), [UInt16](#), [UInt32](#), and [UInt64](#)) are defined as structures.

Like classes, structures define both data (the fields of the structure) and the operations that can be performed on that data (the methods of the structure). This means that you can call methods on structures, including the virtual methods defined on the [System.Object](#) and [System.ValueType](#) classes, and any methods defined on the value type itself. In other words, structures can have fields, properties, and events, as well as static and nonstatic methods. You can create instances of structures, pass them as parameters, store them as local variables, or store them in a field of another value type or reference type. Structures can also implement interfaces.

Value types also differ from classes in several respects. First, although they implicitly inherit from [System.ValueType](#), they cannot directly inherit from any type. Similarly, all value types are sealed, which means that no other type can be derived from them. They also do not require constructors.

For each value type, the common language runtime supplies a corresponding boxed type, which is a class that has the same state and behavior as the value type. An instance of a value type is boxed when it is passed to a method that accepts a parameter of type [System.Object](#). It is unboxed (that is, converted from an instance of a class back to an instance of a value type) when control returns from a method call that accepts a value type as a by-reference parameter. Some languages require that you use special syntax when the boxed type is required; others automatically use the boxed type when it is needed. When you define a value type, you are defining both the boxed and the unboxed type.

Enumerations

An enumeration is a value type that inherits directly from [System.Enum](#) and that supplies alternate names for the values of an underlying primitive type. An enumeration type has a name, an underlying type that must be one of the built-in signed or unsigned integer types (such as [Byte](#), [Int32](#), or [UInt64](#)), and a set of fields. The fields are static literal fields, each of which represents a constant. The same value can be assigned to multiple fields. When this occurs, you must mark one of the values as the primary enumeration value for reflection and string conversion.

You can assign a value of the underlying type to an enumeration and vice versa (no cast is required by the runtime). You can create an instance of an enumeration and call the methods of [System.Enum](#), as well as any methods defined on the enumeration's underlying type. However, some languages might not let you pass an enumeration as a parameter when an instance of the underlying type is required (or vice versa).

The following additional restrictions apply to enumerations:

- They cannot define their own methods.
- They cannot implement interfaces.
- They cannot define properties or events.
- They cannot be generic, unless they are generic only because they are nested within a generic type. That is, an enumeration cannot have type parameters of its own.

NOTE

Nested types (including enumerations) created with Visual Basic, C#, and C++ include the type parameters of all enclosing generic types, and are therefore generic even if they do not have type parameters of their own. For more information, see "Nested Types" in the [Type.MakeGenericType](#) reference topic.

The [FlagsAttribute](#) attribute denotes a special kind of enumeration called a bit field. The runtime itself does not distinguish between traditional enumerations and bit fields, but your language might do so. When this distinction is made, bitwise operators can be used on bit fields, but not on enumerations, to generate unnamed values. Enumerations are generally used for lists of unique elements, such as days of the week, country or region names, and so on. Bit fields are generally used for lists of qualities or quantities that might occur in combination, such as `Red And Big And Fast`.

The following example shows how to use both bit fields and traditional enumerations.

```
using System;
using System.Collections.Generic;

// A traditional enumeration of some root vegetables.
public enum SomeRootVegetables
{
    HorseRadish,
    Radish,
    Turnip
}
```

```

// A bit field or flag enumeration of harvesting seasons.
[Flags]
public enum Seasons
{
    None = 0,
    Summer = 1,
    Autumn = 2,
    Winter = 4,
    Spring = 8,
    All = Summer | Autumn | Winter | Spring
}

public class Example
{
    public static void Main()
    {
        // Hash table of when vegetables are available.
        Dictionary<SomeRootVegetables, Seasons> AvailableIn = new Dictionary<SomeRootVegetables, Seasons>();

        AvailableIn[SomeRootVegetables.HorseRadish] = Seasons.All;
        AvailableIn[SomeRootVegetables.Radish] = Seasons.Spring;
        AvailableIn[SomeRootVegetables.Turnip] = Seasons.Spring |
            Seasons.Autumn;

        // Array of the seasons, using the enumeration.
        Seasons[] theSeasons = new Seasons[] { Seasons.Summer, Seasons.Autumn,
            Seasons.Winter, Seasons.Spring };

        // Print information of what vegetables are available each season.
        foreach (Seasons season in theSeasons)
        {
            Console.WriteLine(String.Format(
                "The following root vegetables are harvested in {0}:\n",
                season.ToString("G")));
            foreach (KeyValuePair<SomeRootVegetables, Seasons> item in AvailableIn)
            {
                // A bitwise comparison.
                if (((Seasons)item.Value & season) > 0)
                    Console.WriteLine(String.Format(" {0:G}\n",
                        (SomeRootVegetables)item.Key));
            }
        }
    }
}

// The example displays the following output:
//   The following root vegetables are harvested in Summer:
//       HorseRadish
//   The following root vegetables are harvested in Autumn:
//       Turnip
//       HorseRadish
//   The following root vegetables are harvested in Winter:
//       HorseRadish
//   The following root vegetables are harvested in Spring:
//       Turnip
//       Radish
//       HorseRadish

```

```

Imports System.Collections.Generic

' A traditional enumeration of some root vegetables.
Public Enum SomeRootVegetables
    HorseRadish
    Radish
    Turnip
End Enum

' A bit field or flag enumeration of harvesting seasons.
<Flags()> Public Enum Seasons
    None = 0
    Summer = 1
    Autumn = 2
    Winter = 4
    Spring = 8
    All = Summer Or Autumn Or Winter Or Spring
End Enum

' Entry point.
Public Class Example
    Public Shared Sub Main()
        ' Hash table of when vegetables are available.
        Dim AvailableIn As New Dictionary(Of SomeRootVegetables, Seasons)()

        AvailableIn(SomeRootVegetables.HorseRadish) = Seasons.All
        AvailableIn(SomeRootVegetables.Radish) = Seasons.Spring
        AvailableIn(SomeRootVegetables.Turnip) = Seasons.Spring Or _
                                                Seasons.Autumn

        ' Array of the seasons, using the enumeration.
        Dim theSeasons() As Seasons = {Seasons.Summer, Seasons.Autumn, _
                                       Seasons.Winter, Seasons.Spring}

        ' Print information of what vegetables are available each season.
        For Each season As Seasons In theSeasons
            Console.WriteLine(String.Format( _
                "The following root vegetables are harvested in {0}:", _
                season.ToString("G")))
            For Each item As KeyValuePair(Of SomeRootVegetables, Seasons) In AvailableIn
                ' A bitwise comparison.
                If (CType(item.Value, Seasons) And season) > 0 Then
                    Console.WriteLine(" " + _
                        CType(item.Key, SomeRootVegetables).ToString("G"))
                End If
            Next
        Next
    End Sub
End Class

```

' The example displays the following output:

- ' The following root vegetables are harvested in Summer:
 - ' HorseRadish
- ' The following root vegetables are harvested in Autumn:
 - ' Turnip
 - ' HorseRadish
- ' The following root vegetables are harvested in Winter:
 - ' HorseRadish
- ' The following root vegetables are harvested in Spring:
 - ' Turnip
 - ' Radish
 - ' HorseRadish

Interfaces

An interface defines a contract that specifies a "can do" relationship or a "has a" relationship. Interfaces are often used to implement functionality, such as comparing and sorting (the [IComparable](#) and [IComparable<T>](#)

interfaces), testing for equality (the [IEquatable<T>](#) interface), or enumerating items in a collection (the [IEnumerable](#) and [IEnumerator<T>](#) interfaces). Interfaces can have properties, methods, and events, all of which are abstract members; that is, although the interface defines the members and their signatures, it leaves it to the type that implements the interface to define the functionality of each interface member. This means that any class or structure that implements an interface must supply definitions for the abstract members declared in the interface. An interface can require any implementing class or structure to also implement one or more other interfaces.

The following restrictions apply to interfaces:

- An interface can be declared with any accessibility, but interface members must all have public accessibility.
- Interfaces cannot define constructors.
- Interfaces cannot define fields.
- Interfaces can define only instance members. They cannot define static members.

Each language must provide rules for mapping an implementation to the interface that requires the member, because more than one interface can declare a member with the same signature, and these members can have separate implementations.

Delegates

Delegates are reference types that serve a purpose similar to that of function pointers in C++. They are used for event handlers and callback functions in .NET. Unlike function pointers, delegates are secure, verifiable, and type safe. A delegate type can represent any instance method or static method that has a compatible signature.

A parameter of a delegate is compatible with the corresponding parameter of a method if the type of the delegate parameter is more restrictive than the type of the method parameter, because this guarantees that an argument passed to the delegate can be passed safely to the method.

Similarly, the return type of a delegate is compatible with the return type of a method if the return type of the method is more restrictive than the return type of the delegate, because this guarantees that the return value of the method can be cast safely to the return type of the delegate.

For example, a delegate that has a parameter of type [IEnumerable](#) and a return type of [Object](#) can represent a method that has a parameter of type [Object](#) and a return value of type [IEnumerable](#). For more information and example code, see [Delegate.CreateDelegate\(Type, Object, MethodInfo\)](#).

A delegate is said to be bound to the method it represents. In addition to being bound to the method, a delegate can be bound to an object. The object represents the first parameter of the method, and is passed to the method every time the delegate is invoked. If the method is an instance method, the bound object is passed as the implicit `this` parameter (`Me` in Visual Basic); if the method is static, the object is passed as the first formal parameter of the method, and the delegate signature must match the remaining parameters. For more information and example code, see [System.Delegate](#).

All delegates inherit from [System.MulticastDelegate](#), which inherits from [System.Delegate](#). The C#, Visual Basic, and C++ languages do not allow inheritance from these types. Instead, they provide keywords for declaring delegates.

Because delegates inherit from [MulticastDelegate](#), a delegate has an invocation list, which is a list of methods that the delegate represents and that are executed when the delegate is invoked. All methods in the list receive the arguments supplied when the delegate is invoked.

NOTE

The return value is not defined for a delegate that has more than one method in its invocation list, even if the delegate has a return type.

In many cases, such as with callback methods, a delegate represents only one method, and the only actions you have to take are creating the delegate and invoking it.

For delegates that represent multiple methods, .NET provides methods of the [Delegate](#) and [MulticastDelegate](#) delegate classes to support operations such as adding a method to a delegate's invocation list (the [Delegate.Combine](#) method), removing a method (the [Delegate.Remove](#) method), and getting the invocation list (the [Delegate.GetInvocationList](#) method).

NOTE

It is not necessary to use these methods for event-handler delegates in C#, C++, and Visual Basic, because these languages provide syntax for adding and removing event handlers.

Type definitions

A type definition includes the following:

- Any attributes defined on the type.
- The type's accessibility (visibility).
- The type's name.
- The type's base type.
- Any interfaces implemented by the type.
- Definitions for each of the type's members.

Attributes

Attributes provide additional user-defined metadata. Most commonly, they are used to store additional information about a type in its assembly, or to modify the behavior of a type member in either the design-time or run-time environment.

Attributes are themselves classes that inherit from [System.Attribute](#). Languages that support the use of attributes each have their own syntax for applying attributes to a language element. Attributes can be applied to almost any language element; the specific elements to which an attribute can be applied are defined by the [AttributeUsageAttribute](#) that is applied to that attribute class.

Type accessibility

All types have a modifier that governs their accessibility from other types. The following table describes the type accessibilities supported by the runtime.

ACCESSIBILITY	DESCRIPTION
public	The type is accessible by all assemblies.
assembly	The type is accessible only from within its assembly.

The accessibility of a nested type depends on its accessibility domain, which is determined by both the declared

accessibility of the member and the accessibility domain of the immediately containing type. However, the accessibility domain of a nested type cannot exceed that of the containing type.

The accessibility domain of a nested member `M` declared in a type `T` within a program `P` is defined as follows (noting that `M` might itself be a type):

- If the declared accessibility of `M` is `public`, the accessibility domain of `M` is the accessibility domain of `T`.
- If the declared accessibility of `M` is `protected internal`, the accessibility domain of `M` is the intersection of the accessibility domain of `T` with the program text of `P` and the program text of any type derived from `T` declared outside `P`.
- If the declared accessibility of `M` is `protected`, the accessibility domain of `M` is the intersection of the accessibility domain of `T` with the program text of `T` and any type derived from `T`.
- If the declared accessibility of `M` is `internal`, the accessibility domain of `M` is the intersection of the accessibility domain of `T` with the program text of `P`.
- If the declared accessibility of `M` is `private`, the accessibility domain of `M` is the program text of `T`.

Type Names

The common type system imposes only two restrictions on names:

- All names are encoded as strings of Unicode (16-bit) characters.
- Names are not permitted to have an embedded (16-bit) value of 0x0000.

However, most languages impose additional restrictions on type names. All comparisons are done on a byte-by-byte basis, and are therefore case-sensitive and locale-independent.

Although a type might reference types from other modules and assemblies, a type must be fully defined within one .NET module. (Depending on compiler support, however, it can be divided into multiple source code files.) Type names need be unique only within a namespace. To fully identify a type, the type name must be qualified by the namespace that contains the implementation of the type.

Base types and interfaces

A type can inherit values and behaviors from another type. The common type system does not allow types to inherit from more than one base type.

A type can implement any number of interfaces. To implement an interface, a type must implement all the virtual members of that interface. A virtual method can be implemented by a derived type and can be invoked either statically or dynamically.

Type members

The runtime enables you to define members of your type, which specifies the behavior and state of a type. Type members include the following:

- [Fields](#)
- [Properties](#)
- [Methods](#)
- [Constructors](#)
- [Events](#)
- [Nested types](#)

Fields

A field describes and contains part of the type's state. Fields can be of any type supported by the runtime. Most commonly, fields are either `private` or `protected`, so that they are accessible only from within the class or from a derived class. If the value of a field can be modified from outside its type, a property set accessor is typically used. Publicly exposed fields are usually read-only and can be of two types:

- Constants, whose value is assigned at design time. These are static members of a class, although they are not defined using the `static` (`Shared` in Visual Basic) keyword.
- Read-only variables, whose values can be assigned in the class constructor.

The following example illustrates these two usages of read-only fields.

```
using System;

public class Constants
{
    public const double Pi = 3.1416;
    public readonly DateTime BirthDate;

    public Constants(DateTime birthDate)
    {
        this.BirthDate = birthDate;
    }
}

public class Example
{
    public static void Main()
    {
        Constants con = new Constants(new DateTime(1974, 8, 18));
        Console.WriteLine(Constants.Pi + "\n");
        Console.WriteLine(con.BirthDate.ToString("d") + "\n");
    }
}
// The example displays the following output if run on a system whose current
// culture is en-US:
//      3.1416
//      8/18/1974
```

```
Public Class Constants
    Public Const Pi As Double = 3.1416
    Public ReadOnly BirthDate As Date

    Public Sub New(birthDate As Date)
        Me.BirthDate = birthDate
    End Sub
End Class

Public Module Example
    Public Sub Main()
        Dim con As New Constants(#8/18/1974#)
        Console.WriteLine(Constants.Pi.ToString())
        Console.WriteLine(con.BirthDate.ToString("d"))
    End Sub
End Module
' The example displays the following output if run on a system whose current
' culture is en-US:
'      3.1416
'      8/18/1974
```

Properties

A property names a value or state of the type and defines methods for getting or setting the property's value. Properties can be primitive types, collections of primitive types, user-defined types, or collections of user-defined types. Properties are often used to keep the public interface of a type independent from the type's actual representation. This enables properties to reflect values that are not directly stored in the class (for example, when a property returns a computed value) or to perform validation before values are assigned to private fields. The following example illustrates the latter pattern.

```
using System;

public class Person
{
    private int m_Age;

    public int Age
    {
        get { return m_Age; }
        set {
            if (value < 0 || value > 125)
            {
                throw new ArgumentOutOfRangeException("The value of the Age property must be between 0 and 125.");
            }
            else
            {
                m_Age = value;
            }
        }
    }
}
```

```
Public Class Person
    Private m_Age As Integer

    Public Property Age As Integer
        Get
            Return m_Age
        End Get
        Set
            If value < 0 Or value > 125 Then
                Throw New ArgumentOutOfRangeException("The value of the Age property must be between 0 and 125.")
            Else
                m_Age = value
            End If
        End Set
    End Property
End Class
```

In addition to including the property itself, the Microsoft intermediate language (MSIL) for a type that contains a readable property includes a `get_ propertyName` method, and the MSIL for a type that contains a writable property includes a `set_ propertyName` method.

Methods

A method describes operations that are available on the type. A method's signature specifies the allowable types of all its parameters and of its return value.

Although most methods define the precise number of parameters required for method calls, some methods support a variable number of parameters. The final declared parameter of these methods is marked with the [ParamArrayAttribute](#) attribute. Language compilers typically provide a keyword, such as `params` in C# and `ParamArray` in Visual Basic, that makes explicit use of [ParamArrayAttribute](#) unnecessary.

Constructors

A constructor is a special kind of method that creates new instances of a class or structure. Like any other method, a constructor can include parameters; however, constructors have no return value (that is, they return `void`).

If the source code for a class does not explicitly define a constructor, the compiler includes a parameterless constructor. However, if the source code for a class defines only parameterized constructors, the Visual Basic and C# compilers do not generate a parameterless constructor.

If the source code for a structure defines constructors, they must be parameterized; a structure cannot define a parameterless constructor, and compilers do not generate parameterless constructors for structures or other value types. All value types do have an implicit parameterless constructor. This constructor is implemented by the common language runtime and initializes all fields of the structure to their default values.

Events

An event defines an incident that can be responded to, and defines methods for subscribing to, unsubscribing from, and raising the event. Events are often used to inform other types of state changes. For more information, see [Events](#).

Nested types

A nested type is a type that is a member of some other type. Nested types should be tightly coupled to their containing type and must not be useful as a general-purpose type. Nested types are useful when the declaring type uses and creates instances of the nested type, and use of the nested type is not exposed in public members.

Nested types are confusing to some developers and should not be publicly visible unless there is a compelling reason for visibility. In a well-designed library, developers should rarely have to use nested types to instantiate objects or declare variables.

Characteristics of type members

The common type system allows type members to have a variety of characteristics; however, languages are not required to support all these characteristics. The following table describes member characteristics.

CHARACTERISTIC	CAN APPLY TO	DESCRIPTION
abstract	Methods, properties, and events	The type does not supply the method's implementation. Types that inherit or implement abstract methods must supply an implementation for the method. The only exception is when the derived type is itself an abstract type. All abstract methods are virtual.

CHARACTERISTIC	CAN APPLY TO	DESCRIPTION
private, family, assembly, family and assembly, family or assembly, or public	All	<p>Defines the accessibility of the member:</p> <p>private Accessible only from within the same type as the member, or within a nested type.</p> <p>family Accessible from within the same type as the member, and from derived types that inherit from it.</p> <p>assembly Accessible only in the assembly in which the type is defined.</p> <p>family and assembly Accessible only from types that qualify for both family and assembly access.</p> <p>family or assembly Accessible only from types that qualify for either family or assembly access.</p> <p>public Accessible from any type.</p>
final	Methods, properties, and events	The virtual method cannot be overridden in a derived type.
initialize-only	Fields	The value can only be initialized, and cannot be written after initialization.
instance	Fields, methods, properties, and events	If a member is not marked as <code>static</code> (C# and C++), <code>Shared</code> (Visual Basic), <code>virtual</code> (C# and C++), or <code>Overridable</code> (Visual Basic), it is an instance member (there is no instance keyword). There will be as many copies of such members in memory as there are objects that use it.
literal	Fields	The value assigned to the field is a fixed value, known at compile time, of a built-in value type. Literal fields are sometimes referred to as constants.

CHARACTERISTIC	CAN APPLY TO	DESCRIPTION
newslot or override	All	<p>Defines how the member interacts with inherited members that have the same signature:</p> <p>newslot Hides inherited members that have the same signature.</p> <p>override Replaces the definition of an inherited virtual method.</p> <p>The default is newslot.</p>
static	Fields, methods, properties, and events	The member belongs to the type it is defined on, not to a particular instance of the type; the member exists even if an instance of the type is not created, and it is shared among all instances of the type.
virtual	Methods, properties, and events	The method can be implemented by a derived type and can be invoked either statically or dynamically. If dynamic invocation is used, the type of the instance that makes the call at run time (rather than the type known at compile time) determines which implementation of the method is called. To invoke a virtual method statically, the variable might have to be cast to a type that uses the desired version of the method.

Overloading

Each type member has a unique signature. Method signatures consist of the method name and a parameter list (the order and types of the method's arguments). Multiple methods with the same name can be defined within a type as long as their signatures differ. When two or more methods with the same name are defined, the method is said to be overloaded. For example, in [System.Char](#), the [IsDigit](#) method is overloaded. One method takes a [Char](#). The other method takes a [String](#) and an [Int32](#).

NOTE

The return type is not considered part of a method's signature. That is, methods cannot be overloaded if they differ only by return type.

Inherit, override, and hide members

A derived type inherits all members of its base type; that is, these members are defined on, and available to, the derived type. The behavior or qualities of inherited members can be modified in two ways:

- A derived type can hide an inherited member by defining a new member with the same signature. This might be done to make a previously public member private or to define new behavior for an inherited method that is marked as `final`.
- A derived type can override an inherited virtual method. The overriding method provides a new definition of the method that will be invoked based on the type of the value at run time rather than the

type of the variable known at compile time. A method can override a virtual method only if the virtual method is not marked as `final` and the new method is at least as accessible as the virtual method.

See also

- [.NET API Browser](#)
- [Common Language Runtime](#)
- [Type Conversion in .NET](#)

Language independence and language-independent components

9/20/2022 • 69 minutes to read • [Edit Online](#)

.NET is language independent. This means that, as a developer, you can develop in one of the many languages that target .NET implementations, such as C#, F#, and Visual Basic. You can access the types and members of class libraries developed for .NET implementations without having to know the language in which they were originally written and without having to follow any of the original language's conventions. If you're a component developer, your component can be accessed by any .NET app, regardless of its language.

NOTE

This first part of this article discusses creating language-independent components, that is, components that can be consumed by apps that are written in any language. You can also create a single component or app from source code written in multiple languages; see [Cross-Language Interoperability](#) in the second part of this article.

To fully interact with other objects written in any language, objects must expose to callers only those features that are common to all languages. This common set of features is defined by the Common Language Specification (CLS), which is a set of rules that apply to generated assemblies. The Common Language Specification is defined in Partition I, Clauses 7 through 11 of the [ECMA-335 Standard: Common Language Infrastructure](#).

If your component conforms to the Common Language Specification, it is guaranteed to be CLS-compliant and can be accessed from code in assemblies written in any programming language that supports the CLS. You can determine whether your component conforms to the Common Language Specification at compile time by applying the [CLSCompliantAttribute](#) attribute to your source code. For more information, see [The CLSCompliantAttribute attribute](#).

CLS compliance rules

This section discusses the rules for creating a CLS-compliant component. For a complete list of rules, see Partition I, Clause 11 of the [ECMA-335 Standard: Common Language Infrastructure](#).

NOTE

The Common Language Specification discusses each rule for CLS compliance as it applies to consumers (developers who are programmatically accessing a component that is CLS-compliant), frameworks (developers who are using a language compiler to create CLS-compliant libraries), and extenders (developers who are creating a tool such as a language compiler or a code parser that creates CLS-compliant components). This article focuses on the rules as they apply to frameworks. Note, though, that some of the rules that apply to extenders may also apply to assemblies that are created using [Reflection.Emit](#).

To design a component that's language independent, you only need to apply the rules for CLS compliance to your component's public interface. Your private implementation does not have to conform to the specification.

IMPORTANT

The rules for CLS compliance apply only to a component's public interface, not to its private implementation.

For example, unsigned integers other than `Byte` are not CLS-compliant. Because the `Person` class in the following example exposes an `Age` property of type `UInt16`, the following code displays a compiler warning.

```
using System;

[assembly: CLSCompliant(true)]

public class Person
{
    private UInt16 personAge = 0;

    public UInt16 Age
    { get { return personAge; } }

}

// The attempt to compile the example displays the following compiler warning:
//   Public1.cs(10,18): warning CS3003: Type of 'Person.Age' is not CLS-compliant
```

```
<Assembly: CLSCompliant(True)>

Public Class Person
    Private personAge As UInt16

    Public ReadOnly Property Age As UInt16
        Get
            Return personAge
        End Get
    End Property
End Class
' The attempt to compile the example displays the following compiler warning:
'   Public1.vb(9) : warning BC40027: Return type of function 'Age' is not CLS-compliant.
'
'   Public ReadOnly Property Age As UInt16
'       ~~~~
```

You can make the `Person` class CLS-compliant by changing the type of the `Age` property from `UInt16` to `Int16`, which is a CLS-compliant, 16-bit signed integer. You don't have to change the type of the private `personAge` field.

```
using System;

[assembly: CLSCompliant(true)]

public class Person
{
    private Int16 personAge = 0;

    public Int16 Age
    { get { return personAge; } }

}
```

```

<Assembly: CLSCompliant(True)>

Public Class Person
    Private personAge As UInt16

    Public ReadOnly Property Age As Int16
        Get
            Return CType(personAge, Int16)
        End Get
    End Property
End Class

```

A library's public interface consists of the following:

- Definitions of public classes.
- Definitions of the public members of public classes, and definitions of members accessible to derived classes (that is, protected members).
- Parameters and return types of public methods of public classes, and parameters and return types of methods accessible to derived classes.

The rules for CLS compliance are listed in the following table. The text of the rules is taken verbatim from the [ECMA-335 Standard: Common Language Infrastructure](#), which is Copyright 2012 by Ecma International. More detailed information about these rules is found in the following sections.

CATEGORY	SEE	RULE	RULE NUMBER
Accessibility	Member accessibility	Accessibility shall not be changed when overriding inherited methods, except when overriding a method inherited from a different assembly with accessibility <code>family-or-assembly</code> . In this case, the override shall have accessibility <code>family</code> .	10

Category	See	Rule	Rule Number
Accessibility	Member accessibility	The visibility and accessibility of types and members shall be such that types in the signature of any member shall be visible and accessible whenever the member itself is visible and accessible. For example, a public method that is visible outside its assembly shall not have an argument whose type is visible only within the assembly. The visibility and accessibility of types composing an instantiated generic type used in the signature of any member shall be visible and accessible whenever the member itself is visible and accessible. For example, an instantiated generic type present in the signature of a member that is visible outside its assembly shall not have a generic argument whose type is visible only within the assembly.	12
Arrays	Arrays	Arrays shall have elements with a CLS-compliant type, and all dimensions of the array shall have lower bounds of zero. Only the fact that an item is an array and the element type of the array shall be required to distinguish between overloads. When overloading is based on two or more array types, the element types shall be named types.	16
Attributes	Attributes	Attributes shall be of type System.Attribute , or a type inheriting from it.	41

CATEGORY	SEE	RULE	RULE NUMBER
Attributes	Attributes	The CLS only allows a subset of the encodings of custom attributes. The only types that shall appear in these encodings are (see Partition IV): System.Type , System.String , System.Char , System.Boolean , System.Byte , System.Int16 , System.Int32 , System.Int64 , System.Single , System.Double , and any enumeration type based on a CLS-compliant base integer type.	34
Attributes	Attributes	The CLS does not allow publicly visible required modifiers (<code>modreq</code> , see Partition II), but does allow optional modifiers (<code>modopt</code> , see Partition II) it does not understand.	35
Constructors	Constructors	An object constructor shall call some instance constructor of its base class before any access occurs to inherited instance data. (This does not apply to value types, which need not have constructors.)	21
Constructors	Constructors	An object constructor shall not be called except as part of the creation of an object, and an object shall not be initialized twice.	22
Enumerations	Enumerations	The underlying type of an enum shall be a built-in CLS integer type, the name of the field shall be "value_#", and that field shall be marked <code>RTSpecialName</code> .	7
Enumerations	Enumerations	There are two distinct kinds of enums, indicated by the presence or absence of the System.FlagsAttribute (see Partition IV Library) custom attribute. One represents named integer values; the other represents named bit flags that can be combined to generate an unnamed value. The value of an <code>enum</code> is not limited to the specified values.	8

CATEGORY	SEE	RULE	RULE NUMBER
Enumerations	Enumerations	Literal static fields of an enum shall have the type of the enum itself.	9
Events	Events	The methods that implement an event shall be marked <code>SpecialName</code> in the metadata.	29
Events	Events	The accessibility of an event and of its accessors shall be identical.	30
Events	Events	The <code>add</code> and <code>remove</code> methods for an event shall both either be present or absent.	31
Events	Events	The <code>add</code> and <code>remove</code> methods for an event shall each take one parameter whose type defines the type of the event and that shall be derived from System.Delegate .	32
Events	Events	Events shall adhere to a specific naming pattern. The <code>SpecialName</code> attribute referred to in CLS rule 29 shall be ignored in appropriate name comparisons and shall adhere to identifier rules.	33
Exceptions	Exceptions	Objects that are thrown shall be of type System.Exception or a type inheriting from it. Nonetheless, CLS-compliant methods are not required to block the propagation of other types of exceptions.	40
General	CLS compliance rules	CLS rules apply only to those parts of a type that are accessible or visible outside of the defining assembly.	1
General	CLS compliance rules	Members of non-CLS compliant types shall not be marked CLS-compliant.	2

CATEGORY	SEE	RULE	RULE NUMBER
Generics	Generic types and members	Nested types shall have at least as many generic parameters as the enclosing type. Generic parameters in a nested type correspond by position to the generic parameters in its enclosing type.	42
Generics	Generic types and members	The name of a generic type shall encode the number of type parameters declared on the non-nested type, or newly introduced to the type if nested, according to the rules defined above.	43
Generics	Generic types and members	A generic type shall redeclare sufficient constraints to guarantee that any constraints on the base type, or interfaces would be satisfied by the generic type constraints.	44
Generics	Generic types and members	Types used as constraints on generic parameters shall themselves be CLS-compliant.	45
Generics	Generic types and members	The visibility and accessibility of members (including nested types) in an instantiated generic type shall be considered to be scoped to the specific instantiation rather than the generic type declaration as a whole. Assuming this, the visibility and accessibility rules of CLS rule 12 still apply.	46
Generics	Generic types and members	For each abstract or virtual generic method, there shall be a default concrete (nonabstract) implementation	47
Interfaces	Interfaces	CLS-compliant interfaces shall not require the definition of non-CLS compliant methods in order to implement them.	18

CATEGORY	SEE	RULE	RULE NUMBER
Interfaces	Interfaces	CLS-compliant interfaces shall not define static methods, nor shall they define fields.	19
Members	Type members in general	Global static fields and methods are not CLS-compliant.	36
Members	--	The value of a literal static is specified by using field initialization metadata. A CLS-compliant literal must have a value specified in field initialization metadata that is of exactly the same type as the literal (or of the underlying type, if that literal is an <code>enum</code>).	13
Members	Type members in general	The vararg constraint is not part of the CLS, and the only calling convention supported by the CLS is the standard managed calling convention.	15
Naming conventions	Naming conventions	Assemblies shall follow Annex 7 of Technical Report 15 of the Unicode Standard3.0 governing the set of characters permitted to start and be included in identifiers, available online at Unicode Normalization Forms . Identifiers shall be in the canonical format defined by Unicode Normalization Form C. For CLS purposes, two identifiers are the same if their lowercase mappings (as specified by the Unicode locale-insensitive, one-to-one lowercase mappings) are the same. That is, for two identifiers to be considered different under the CLS they shall differ in more than simply their case. However, in order to override an inherited definition the CLI requires the precise encoding of the original declaration be used.	4

CATEGORY	SEE	RULE	RULE NUMBER
Overloading	Naming conventions	All names introduced in a CLS-compliant scope shall be distinct independent of kind, except where the names are identical and resolved via overloading. That is, while the CTS allows a single type to use the same name for a method and a field, the CLS does not.	5
Overloading	Naming conventions	Fields and nested types shall be distinct by identifier comparison alone, even though the CTS allows distinct signatures to be distinguished. Methods, properties, and events that have the same name (by identifier comparison) shall differ by more than just the return type, except as specified in CLS Rule 39	6
Overloading	Overloads	Only properties and methods can be overloaded.	37
Overloading	Overloads	Properties and methods can be overloaded based only on the number and types of their parameters, except the conversion operators named <code>op_Implicit</code> and <code>op_Explicit</code> , which can also be overloaded based on their return type.	38
Overloading	--	If two or more CLS-compliant methods declared in a type have the same name and, for a specific set of type instantiations, they have the same parameter and return types, then all these methods shall be semantically equivalent at those type instantiations.	48
Properties	Properties	The methods that implement the getter and setter methods of a property shall be marked <code>SpecialName</code> in the metadata.	24

CATEGORY	SEE	RULE	RULE NUMBER
Properties	Properties	A property's accessors shall all be static, all be virtual, or all be instance.	26
Properties	Properties	The type of a property shall be the return type of the getter and the type of the last argument of the setter. The types of the parameters of the property shall be the types of the parameters to the getter and the types of all but the final parameter of the setter. All of these types shall be CLS-compliant, and shall not be managed pointers (that is, shall not be passed by reference).	27
Properties	Properties	Properties shall adhere to a specific naming pattern. The <code>SpecialName</code> attribute referred to in CLS rule 24 shall be ignored in appropriate name comparisons and shall adhere to identifier rules. A property shall have a getter method, a setter method, or both.	28
Type conversion	Type conversion	If either <code>op_Implicit</code> or <code>op_Explicit</code> is provided, an alternate means of providing the coercion shall be provided.	39
Types	Types and type member signatures	Boxed value types are not CLS-compliant.	3
Types	Types and type member signatures	All types appearing in a signature shall be CLS-compliant. All types composing an instantiated generic type shall be CLS-compliant.	11
Types	Types and type member signatures	Typed references are not CLS-compliant.	14
Types	Types and type member signatures	Unmanaged pointer types are not CLS-compliant.	17

Category	See	Rule	Rule Number
Types	Types and type member signatures	CLS-compliant classes, value types, and interfaces shall not require the implementation of non-CLS-compliant members	20
Types	Types and type member signatures	<code>System.Object</code> is CLS-compliant. Any other CLS-compliant class shall inherit from a CLS-compliant class.	23

Index to subsections:

- [Types and type member signatures](#)
- [Naming conventions](#)
- [Type conversion](#)
- [Arrays](#)
- [Interfaces](#)
- [Enumerations](#)
- [Type members in general](#)
- [Member accessibility](#)
- [Generic types and members](#)
- [Constructors](#)
- [Properties](#)
- [Events](#)
- [Overloads](#)
- [Exceptions](#)
- [Attributes](#)

Types and type-member signatures

The `System.Object` type is CLS-compliant and is the base type of all object types in the .NET type system.

Inheritance in .NET is either implicit (for example, the `String` class implicitly inherits from the `Object` class) or explicit (for example, the `CultureNotFoundException` class explicitly inherits from the `ArgumentException` class, which explicitly inherits from the `Exception` class. For a derived type to be CLS compliant, its base type must also be CLS-compliant.

The following example shows a derived type whose base type is not CLS-compliant. It defines a base `Counter` class that uses an unsigned 32-bit integer as a counter. Because the class provides counter functionality by wrapping an unsigned integer, the class is marked as non-CLS-compliant. As a result, a derived class, `NonZeroCounter`, is also not CLS-compliant.

```
using System;

[assembly: CLSCompliant(true)]

[CLSCompliant(false)]
public class Counter
{
    UInt32 ctr;

    public Counter()
    {
        ctr = 0;
    }

    protected Counter(UInt32 ctr)
    {
        this.ctr = ctr;
    }

    public override string ToString()
    {
        return String.Format("{0}"). , ctr);
    }

    public UInt32 Value
    {
        get { return ctr; }
    }

    public void Increment()
    {
        ctr += (uint) 1;
    }
}

public class NonZeroCounter : Counter
{
    public NonZeroCounter(int startIndex) : this((uint) startIndex)
    {

    }

    private NonZeroCounter(UInt32 startIndex) : base(startIndex)
    {

    }
}

// Compilation produces a compiler warning like the following:
//     Type3.cs(37,14): warning CS3009: 'NonZeroCounter': base type 'Counter' is not
//                     CLS-compliant
//     Type3.cs(7,14): (Location of symbol related to previous warning)
```

```

<Assembly: CLSCompliant(True)>

<CLSCompliant(False)> _
Public Class Counter
    Dim ctr As UInt32

    Public Sub New
        ctr = 0
    End Sub

    Protected Sub New(ctr As UInt32)
        ctr = ctr
    End Sub

    Public Overrides Function ToString() As String
        Return String.Format("{0}", ctr)
    End Function

    Public ReadOnly Property Value As UInt32
        Get
            Return ctr
        End Get
    End Property

    Public Sub Increment()
        ctr += CType(1, UInt32)
    End Sub
End Class

Public Class NonZeroCounter : Inherits Counter
    Public Sub New(startIndex As Integer)
        MyBase.New(CType(startIndex, UInt32))
    End Sub

    Private Sub New(startIndex As UInt32)
        MyBase.New(CType(startIndex, UInt32))
    End Sub
End Class

' Compilation produces a compiler warning like the following:
' Type3.vb(34) : warning BC40026: 'NonZeroCounter' is not CLS-compliant
' because it derives from 'Counter', which is not CLS-compliant.
'

' Public Class NonZeroCounter : Inherits Counter
' ~~~~~

```

All types that appear in member signatures, including a method's return type or a property type, must be CLS-compliant. In addition, for generic types:

- All types that compose an instantiated generic type must be CLS-compliant.
- All types used as constraints on generic parameters must be CLS-compliant.

The .NET [common type system](#) includes many built-in types that are supported directly by the common language runtime and are specially encoded in an assembly's metadata. Of these intrinsic types, the types listed in the following table are CLS-compliant.

CLS-COMPLIANT TYPE	DESCRIPTION
Byte	8-bit unsigned integer
Int16	16-bit signed integer

CLS-COMPLIANT TYPE	DESCRIPTION
Int32	32-bit signed integer
Int64	64-bit signed integer
Half	Half-precision floating-point value
Single	Single-precision floating-point value
Double	Double-precision floating-point value
Boolean	true or false value type
Char	UTF-16 encoded code unit
Decimal	Non-floating-point decimal number
IntPtr	Pointer or handle of a platform-defined size
String	Collection of zero, one, or more Char objects

The intrinsic types listed in the following table are not CLS-Compliant.

NON-COMPLIANT TYPE	DESCRIPTION	CLS-COMPLIANT ALTERNATIVE
SByte	8-bit signed integer data type	Int16
UInt16	16-bit unsigned integer	Int32
UInt32	32-bit unsigned integer	Int64
UInt64	64-bit unsigned integer	Int64 (may overflow), BigInteger, or Double
UIntPtr	Unsigned pointer or handle	IntPtr

The .NET Class Library or any other class library may include other types that aren't CLS-compliant, for example:

- Boxed value types. The following C# example creates a class that has a public property of type `int*` named `value`. Because an `int*` is a boxed value type, the compiler flags it as non-CLS-compliant.

```

using System;

[assembly:CLSCCompliant(true)]

public unsafe class TestClass
{
    private int* val;

    public TestClass(int number)
    {
        val = (int*) number;
    }

    public int* Value {
        get { return val; }
    }
}

// The compiler generates the following output when compiling this example:
//     warning CS3003: Type of 'TestClass.Value' is not CLS-compliant

```

- Typed references, which are special constructs that contain a reference to an object and a reference to a type. Typed references are represented in .NET by the [TypedReference](#) class.

If a type is not CLS-compliant, you should apply the [CLSCCompliantAttribute](#) attribute with an `isCompliant` value of `false` to it. For more information, see [The CLSCCompliantAttribute attribute](#) section.

The following example illustrates the problem of CLS compliance in a method signature and in generic type instantiation. It defines an `InvoiceItem` class with a property of type `UInt32`, a property of type `Nullable<UInt32>`, and a constructor with parameters of type `UInt32` and `Nullable<UInt32>`. You get four compiler warnings when you try to compile this example.

```
using System;

[assembly: CLSCompliant(true)]

public class InvoiceItem
{
    private uint invId = 0;
    private uint itemId = 0;
    private Nullable<uint> qty;

    public InvoiceItem(uint sku, Nullable<uint> quantity)
    {
        itemId = sku;
        qty = quantity;
    }

    public Nullable<uint> Quantity
    {
        get { return qty; }
        set { qty = value; }
    }

    public uint InvoiceId
    {
        get { return invId; }
        set { invId = value; }
    }
}

// The attempt to compile the example displays the following output:
// Type1.cs(13,23): warning CS3001: Argument type 'uint' is not CLS-compliant
// Type1.cs(13,33): warning CS3001: Argument type 'uint?' is not CLS-compliant
// Type1.cs(19,26): warning CS3003: Type of 'InvoiceItem.Quantity' is not CLS-compliant
// Type1.cs(25,16): warning CS3003: Type of 'InvoiceItem.InvoiceId' is not CLS-compliant
```

```

<Assembly: CLSCompliant(True)>

Public Class InvoiceItem

    Private invId As UInteger = 0
    Private itemId As UInteger = 0
    Private qty AS Nullable(Of UInteger)

    Public Sub New(sku As UInteger, quantity As Nullable(Of UInteger))
        itemId = sku
        qty = quantity
    End Sub

    Public Property Quantity As Nullable(Of UInteger)
        Get
            Return qty
        End Get
        Set
            qty = value
        End Set
    End Property

    Public Property InvoiceId As UInteger
        Get
            Return invId
        End Get
        Set
            invId = value
        End Set
    End Property
End Class

' The attempt to compile the example displays output similar to the following:
' Type1.vb(13) : warning BC40028: Type of parameter 'sku' is not CLS-compliant.
'
'     Public Sub New(sku As UInteger, quantity As Nullable(Of UInteger))
'         ~~~
'
' Type1.vb(13) : warning BC40041: Type 'UInteger' is not CLS-compliant.
'
'     Public Sub New(sku As UInteger, quantity As Nullable(Of UInteger))
'         ~~~~~
'
' Type1.vb(18) : warning BC40041: Type 'UInteger' is not CLS-compliant.
'
'     Public Property Quantity As Nullable(Of UInteger)
'         ~~~~~
'
' Type1.vb(27) : warning BC40027: Return type of function 'InvoiceId' is not CLS-compliant.
'
'     Public Property InvoiceId As UInteger
'         ~~~~~

```

To eliminate the compiler warnings, replace the non-CLS-compliant types in the `InvoiceItem` public interface with compliant types:

```
using System;

[assembly: CLSCompliant(true)]

public class InvoiceItem
{
    private uint invId = 0;
    private uint itemId = 0;
    private Nullable<int> qty;

    public InvoiceItem(int sku, Nullable<int> quantity)
    {
        if (sku <= 0)
            throw new ArgumentOutOfRangeException("The item number is zero or negative.");
        itemId = (uint) sku;

        qty = quantity;
    }

    public Nullable<int> Quantity
    {
        get { return qty; }
        set { qty = value; }
    }

    public int InvoiceId
    {
        get { return (int) invId; }
        set {
            if (value <= 0)
                throw new ArgumentOutOfRangeException("The invoice number is zero or negative.");
            invId = (uint) value; }
    }
}
```

```

<Assembly: CLSCompliant(True)>

Public Class InvoiceItem

    Private invId As UInteger = 0
    Private itemId As UInteger = 0
    Private qty AS Nullable(Of Integer)

    Public Sub New(sku As Integer, quantity As Nullable(Of Integer))
        If sku <= 0 Then
            Throw New ArgumentOutOfRangeException("The item number is zero or negative.")
        End If
        itemId = CUInt(sku)
        qty = quantity
    End Sub

    Public Property Quantity As Nullable(Of Integer)
        Get
            Return qty
        End Get
        Set
            qty = value
        End Set
    End Property

    Public Property InvoiceId As Integer
        Get
            Return CInt(invId)
        End Get
        Set
            invId = CUInt(value)
        End Set
    End Property
End Class

```

In addition to the specific types listed, some categories of types are not CLS compliant. These include unmanaged pointer types and function pointer types. The following example generates a compiler warning because it uses a pointer to an integer to create an array of integers.

```

using System;

[assembly: CLSCompliant(true)]

public class ArrayHelper
{
    unsafe public static Array CreateInstance(Type type, int* ptr, int items)
    {
        Array arr = Array.CreateInstance(type, items);
        int* addr = ptr;
        for (int ctr = 0; ctr < items; ctr++) {
            int value = *addr;
            arr.SetValue(value, ctr);
            addr++;
        }
        return arr;
    }
}
// The attempt to compile this example displays the following output:
//     UnmanagedType.cs(8,57): warning CS3001: Argument type 'int*' is not CLS-compliant

```

For CLS-compliant abstract classes (that is, classes marked as `abstract` in C# or as `MustInherit` in Visual Basic), all members of the class must also be CLS-compliant.

Naming conventions

Because some programming languages are case-insensitive, identifiers (such as the names of namespaces, types, and members) must differ by more than case. Two identifiers are considered equivalent if their lowercase mappings are the same. The following C# example defines two public classes, `Person` and `person`. Because they differ only by case, the C# compiler flags them as not CLS-compliant.

```
using System;

[assembly: CLSCompliant(true)]

public class Person : person
{
}

public class person
{
}

// Compilation produces a compiler warning like the following:
//   Naming1.cs(11,14): warning CS3005: Identifier 'person' differing
//                     only in case is not CLS-compliant
//   Naming1.cs(6,14): (Location of symbol related to previous warning)
```

Programming language identifiers, such as the names of namespaces, types, and members, must conform to the [Unicode Standard](#). This means that:

- The first character of an identifier can be any Unicode uppercase letter, lowercase letter, title case letter, modifier letter, other letter, or letter number. For information on Unicode character categories, see the [System.Globalization.UnicodeCategory](#) enumeration.
- Subsequent characters can be from any of the categories as the first character, and can also include non-spacing marks, spacing combining marks, decimal numbers, connector punctuation, and formatting codes.

Before you compare identifiers, you should filter out formatting codes and convert the identifiers to Unicode Normalization Form C, because a single character can be represented by multiple UTF-16-encoded code units. Character sequences that produce the same code units in Unicode Normalization Form C are not CLS-compliant. The following example defines a property named `Å`, which consists of the character ANGSTROM SIGN (U+212B), and a second property named `Ä`, which consists of the character LATIN CAPITAL LETTER A WITH RING ABOVE (U+00C5). Both the C# and Visual Basic compilers flag the source code as non-CLS-compliant.

```

public class Size
{
    private double a1;
    private double a2;

    public double A
    {
        get { return a1; }
        set { a1 = value; }
    }

    public double Å
    {
        get { return a2; }
        set { a2 = value; }
    }
}

// Compilation produces a compiler warning like the following:
//   Naming2a.cs(16,18): warning CS3005: Identifier 'Size.Å' differing only in case is not
//       CLS-compliant
//   Naming2a.cs(10,18): (Location of symbol related to previous warning)
//   Naming2a.cs(18,8): warning CS3005: Identifier 'Size.Å.get' differing only in case is not
//       CLS-compliant
//   Naming2a.cs(12,8): (Location of symbol related to previous warning)
//   Naming2a.cs(19,8): warning CS3005: Identifier 'Size.Å.set' differing only in case is not
//       CLS-compliant
//   Naming2a.cs(13,8): (Location of symbol related to previous warning)

```

```

<Assembly: CLSCompliant(True)>
Public Class Size
    Private a1 As Double
    Private a2 As Double

    Public Property A As Double
        Get
            Return a1
        End Get
        Set
            a1 = value
        End Set
    End Property

    Public Property Å As Double
        Get
            Return a2
        End Get
        Set
            a2 = value
        End Set
    End Property
End Class

' Compilation produces a compiler warning like the following:
'   Naming1.vb(9) : error BC30269: 'Public Property Å As Double' has multiple definitions
'   with identical signatures.
'
'   Public Property Å As Double
'       ~

```

Member names within a particular scope (such as the namespaces within an assembly, the types within a namespace, or the members within a type) must be unique except for names that are resolved through overloading. This requirement is more stringent than that of the common type system, which allows multiple members within a scope to have identical names as long as they are different kinds of members (for example, one is a method and one is a field). In particular, for type members:

- Fields and nested types are distinguished by name alone.
- Methods, properties, and events that have the same name must differ by more than just return type.

The following example illustrates the requirement that member names must be unique within their scope. It defines a class named `Converter` that includes four members named `Conversion`. Three are methods, and one is a property. The method that includes an `Int64` parameter is uniquely named, but the two methods with an `Int32` parameter are not, because return value is not considered a part of a member's signature. The `Conversion` property also violates this requirement, because properties cannot have the same name as overloaded methods.

```
using System;

[assembly: CLSCompliant(true)]

public class Converter
{
    public double Conversion(int number)
    {
        return (double) number;
    }

    public float Conversion(int number)
    {
        return (float) number;
    }

    public double Conversion(long number)
    {
        return (double) number;
    }

    public bool Conversion
    {
        get { return true; }
    }
}

// Compilation produces a compiler error like the following:
// Naming3.cs(13,17): error CS0111: Type 'Converter' already defines a member called
//                 'Conversion' with the same parameter types
// Naming3.cs(8,18): (Location of symbol related to previous error)
// Naming3.cs(23,16): error CS0102: The type 'Converter' already contains a definition for
//                 'Conversion'
// Naming3.cs(8,18): (Location of symbol related to previous error)
```

```

<Assembly: CLSCompliant(True)>

Public Class Converter
    Public Function Conversion(number As Integer) As Double
        Return CDbl(number)
    End Function

    Public Function Conversion(number As Integer) As Single
        Return CSng(number)
    End Function

    Public Function Conversion(number As Long) As Double
        Return CDbl(number)
    End Function

    Public ReadOnly Property Conversion As Boolean
        Get
            Return True
        End Get
    End Property
End Class
' Compilation produces a compiler error like the following:
' Naming3.vb(8) : error BC30301: 'Public Function Conversion(number As Integer) As Double'
'                 and 'Public Function Conversion(number As Integer) As Single' cannot
'                 overload each other because they differ only by return types.
'
'     Public Function Conversion(number As Integer) As Double
'     ~~~~~
'
' Naming3.vb(20) : error BC30260: 'Conversion' is already declared as 'Public Function
'                 Conversion(number As Integer) As Single' in this class.
'
'     Public ReadOnly Property Conversion As Boolean
'     ~~~~~

```

Individual languages include unique keywords, so languages that target the common language runtime must also provide some mechanism for referencing identifiers (such as type names) that coincide with keywords. For example, `case` is a keyword in both C# and Visual Basic. However, the following Visual Basic example is able to disambiguate a class named `case` from the `case` keyword by using opening and closing braces. Otherwise, the example would produce the error message, "Keyword is not valid as an identifier," and fail to compile.

```

Public Class [case]
    Private _id As Guid
    Private name As String

    Public Sub New(name As String)
        _id = Guid.NewGuid()
        Me.name = name
    End Sub

    Public ReadOnly Property ClientName As String
        Get
            Return name
        End Get
    End Property
End Class

```

The following C# example is able to instantiate the `case` class by using the `@` symbol to disambiguate the identifier from the language keyword. Without it, the C# compiler would display two error messages, "Type expected" and "Invalid expression term 'case'."

```

using System;

public class Example
{
    public static void Main()
    {
        @case c = new @case("John");
        Console.WriteLine(c.ClientName);
    }
}

```

Type conversion

The Common Language Specification defines two conversion operators:

- `op_Implicit`, which is used for widening conversions that do not result in loss of data or precision. For example, the `Decimal` structure includes an overloaded `op_Implicit` operator to convert values of integral types and `Char` values to `Decimal` values.
- `op_Explicit`, which is used for narrowing conversions that can result in loss of magnitude (a value is converted to a value that has a smaller range) or precision. For example, the `Decimal` structure includes an overloaded `op_Explicit` operator to convert `Double` and `Single` values to `Decimal` and to convert `Decimal` values to integral values, `Double`, `Single`, and `Char`.

However, not all languages support operator overloading or the definition of custom operators. If you choose to implement these conversion operators, you should also provide an alternate way to perform the conversion. We recommend that you provide `From Xxx` and `To Xxx` methods.

The following example defines CLS-compliant implicit and explicit conversions. It creates a `UDouble` class that represents an unsigned, double-precision, floating-point number. It provides for implicit conversions from `UDouble` to `Double` and for explicit conversions from `UDouble` to `Single`, `Double` to `UDouble`, and `Single` to `UDouble`. It also defines a `ToDouble` method as an alternative to the implicit conversion operator and the `ToSingle`, `FromDouble`, and `FromSingle` methods as alternatives to the explicit conversion operators.

```

using System;

public struct UDouble
{
    private double number;

    public UDouble(double value)
    {
        if (value < 0)
            throw new InvalidCastException("A negative value cannot be converted to a UDouble.");

        number = value;
    }

    public UDouble(float value)
    {
        if (value < 0)
            throw new InvalidCastException("A negative value cannot be converted to a UDouble.");

        number = value;
    }

    public static readonly UDouble MinValue = (UDouble) 0.0;
    public static readonly UDouble MaxValue = (UDouble) Double.MaxValue;

    public static explicit operator Double(UDouble value)
    {
        return value.number;
    }
}

```

```
}

public static implicit operator Single(UDouble value)
{
    if (value.number > (double) Single.MaxValue)
        throw new InvalidCastException("A UDouble value is out of range of the Single type.");

    return (float) value.number;
}

public static explicit operator UDouble(double value)
{
    if (value < 0)
        throw new InvalidCastException("A negative value cannot be converted to a UDouble.");

    return new UDouble(value);
}

public static implicit operator UDouble(float value)
{
    if (value < 0)
        throw new InvalidCastException("A negative value cannot be converted to a UDouble.");

    return new UDouble(value);
}

public static Double ToDouble(UDouble value)
{
    return (Double) value;
}

public static float ToSingle(UDouble value)
{
    return (float) value;
}

public static UDouble FromDouble(double value)
{
    return new UDouble(value);
}

public static UDouble FromSingle(float value)
{
    return new UDouble(value);
}
```

```

Public Structure UDouble
    Private number As Double

    Public Sub New(value As Double)
        If value < 0 Then
            Throw New InvalidCastException("A negative value cannot be converted to a UDouble.")
        End If
        number = value
    End Sub

    Public Sub New(value As Single)
        If value < 0 Then
            Throw New InvalidCastException("A negative value cannot be converted to a UDouble.")
        End If
        number = value
    End Sub

    Public Shared ReadOnly MinValue As UDouble = CType(0.0, UDouble)
    Public Shared ReadOnly MaxValue As UDouble = Double.MaxValue

    Public Shared Widening Operator CType(value As UDouble) As Double
        Return value.number
    End Operator

    Public Shared Narrowing Operator CType(value As UDouble) As Single
        If value.number > CDbl(Single.MaxValue) Then
            Throw New InvalidCastException("A UDouble value is out of range of the Single type.")
        End If
        Return CSng(value.number)
    End Operator

    Public Shared Narrowing Operator CType(value As Double) As UDouble
        If value < 0 Then
            Throw New InvalidCastException("A negative value cannot be converted to a UDouble.")
        End If
        Return New UDouble(value)
    End Operator

    Public Shared Narrowing Operator CType(value As Single) As UDouble
        If value < 0 Then
            Throw New InvalidCastException("A negative value cannot be converted to a UDouble.")
        End If
        Return New UDouble(value)
    End Operator

    Public Shared Function ToDouble(value As UDouble) As Double
        Return CType(value, Double)
    End Function

    Public Shared Function ToSingle(value As UDouble) As Single
        Return CType(value, Single)
    End Function

    Public Shared Function FromDouble(value As Double) As UDouble
        Return New UDouble(value)
    End Function

    Public Shared Function FromSingle(value As Single) As UDouble
        Return New UDouble(value)
    End Function
End Structure

```

Arrays

CLS-compliant arrays conform to the following rules:

- All dimensions of an array must have a lower bound of zero. The following example creates a non-CLS-

compliant array with a lower bound of one. Despite the presence of the [CLSCompliantAttribute](#) attribute, the compiler does not detect that the array returned by the `Numbers.GetTenPrimes` method is not CLS-compliant.

```
[assembly: CLSCompliant(true)]  
  
public class Numbers  
{  
    public static Array GetTenPrimes()  
    {  
        Array arr = Array.CreateInstance(typeof(Int32), new int[] {10}, new int[] {1});  
        arr.SetValue(1, 1);  
        arr.SetValue(2, 2);  
        arr.SetValue(3, 3);  
        arr.SetValue(5, 4);  
        arr.SetValue(7, 5);  
        arr.SetValue(11, 6);  
        arr.SetValue(13, 7);  
        arr.SetValue(17, 8);  
        arr.SetValue(19, 9);  
        arr.SetValue(23, 10);  
  
        return arr;  
    }  
}
```

```
<Assembly: CLSCompliant(True)>  
  
Public Class Numbers  
    Public Shared Function GetTenPrimes() As Array  
        Dim arr As Array = Array.CreateInstance(GetType(Int32), {10}, {1})  
        arr.SetValue(1, 1)  
        arr.SetValue(2, 2)  
        arr.SetValue(3, 3)  
        arr.SetValue(5, 4)  
        arr.SetValue(7, 5)  
        arr.SetValue(11, 6)  
        arr.SetValue(13, 7)  
        arr.SetValue(17, 8)  
        arr.SetValue(19, 9)  
        arr.SetValue(23, 10)  
  
        Return arr  
    End Function  
End Class
```

- All array elements must consist of CLS-compliant types. The following example defines two methods that return non-CLS-compliant arrays. The first returns an array of [UInt32](#) values. The second returns an [Object](#) array that includes [Int32](#) and [UInt32](#) values. Although the compiler identifies the first array as non-compliant because of its [UInt32](#) type, it fails to recognize that the second array includes non-CLS-compliant elements.

```

using System;

[assembly: CLSCompliant(true)]

public class Numbers
{
    public static UInt32[] GetTenPrimes()
    {
        uint[] arr = { 1u, 2u, 3u, 5u, 7u, 11u, 13u, 17u, 19u };
        return arr;
    }

    public static Object[] GetFivePrimes()
    {
        Object[] arr = { 1, 2, 3, 5u, 7u };
        return arr;
    }
}

// Compilation produces a compiler warning like the following:
//   Array2.cs(8,27): warning CS3002: Return type of 'Numbers.GetTenPrimes()' is not
//       CLS-compliant

```

```

<Assembly: CLSCompliant(True)>

Public Class Numbers
    Public Shared Function GetTenPrimes() As UInt32()
        Return {1ui, 2ui, 3ui, 5ui, 7ui, 11ui, 13ui, 17ui, 19ui}
    End Function

    Public Shared Function GetFivePrimes() As Object()
        Dim arr() As Object = {1, 2, 3, 5ui, 7ui}
        Return arr
    End Function
End Class
' Compilation produces a compiler warning like the following:
'   warning BC40027: Return type of function 'GetTenPrimes' is not CLS-compliant.
'
'   Public Shared Function GetTenPrimes() As UInt32()
'       ~~~~~

```

- Overload resolution for methods that have array parameters is based on the fact that they are arrays and on their element type. For this reason, the following definition of an overloaded `GetSquares` method is CLS-compliant.

```

using System;
using System.Numerics;

[assembly: CLSCompliant(true)]

public class Numbers
{
    public static byte[] GetSquares(byte[] numbers)
    {
        byte[] numbersOut = new byte[numbers.Length];
        for (int ctr = 0; ctr < numbers.Length; ctr++) {
            int square = ((int) numbers[ctr]) * ((int) numbers[ctr]);
            if (square <= Byte.MaxValue)
                numbersOut[ctr] = (byte) square;
            // If there's an overflow, assign MaxValue to the corresponding
            // element.
            else
                numbersOut[ctr] = Byte.MaxValue;
        }
        return numbersOut;
    }

    public static BigInteger[] GetSquares(BigInteger[] numbers)
    {
        BigInteger[] numbersOut = new BigInteger[numbers.Length];
        for (int ctr = 0; ctr < numbers.Length; ctr++)
            numbersOut[ctr] = numbers[ctr] * numbers[ctr];

        return numbersOut;
    }
}

```

```

Imports System.Numerics

<Assembly: CLSCompliant(True)>

Public Module Numbers
    Public Function GetSquares(numbers As Byte()) As Byte()
        Dim numbersOut(numbers.Length - 1) As Byte
        For ctr As Integer = 0 To numbers.Length - 1
            Dim square As Integer = (CInt(numbers(ctr)) * CInt(numbers(ctr)))
            If square <= Byte.MaxValue Then
                numbersOut(ctr) = CByte(square)
                ' If there's an overflow, assign MaxValue to the corresponding
                ' element.
            Else
                numbersOut(ctr) = Byte.MaxValue
            End If
        Next
        Return numbersOut
    End Function

    Public Function GetSquares(numbers As BigInteger()) As BigInteger()
        Dim numbersOut(numbers.Length - 1) As BigInteger
        For ctr As Integer = 0 To numbers.Length - 1
            numbersOut(ctr) = numbers(ctr) * numbers(ctr)
        Next
        Return numbersOut
    End Function
End Module

```

Interfaces

CLS-compliant interfaces can define properties, events, and virtual methods (methods with no implementation). A CLS-compliant interface cannot have any of the following:

- Static methods or static fields. Both the C# and Visual Basic compilers generate compiler errors if you define a static member in an interface.
- Fields. Both the C# and Visual Basic compilers generate compiler errors if you define a field in an interface.
- Methods that are not CLS-compliant. For example, the following interface definition includes a method, `INumber.GetUnsigned`, that is marked as non-CLS-compliant. This example generates a compiler warning.

```
using System;

[assembly:CLSCompliant(true)]

public interface INumber
{
    int Length();
    [CLSCompliant(false)] ulong GetUnsigned();
}

// Attempting to compile the example displays output like the following:
//   Interface2.cs(8,32): warning CS3010: 'INumber.GetUnsigned()': CLS-compliant interfaces
//       must have only CLS-compliant members
```

```
<Assembly: CLSCompliant(True)>

Public Interface INumber
    Function Length As Integer

        <CLSCompliant(False)> Function GetUnsigned As ULong
End Interface

' Attempting to compile the example displays output like the following:
'   Interface2.vb(9) : warning BC40033: Non CLS-compliant 'function' is not allowed in a
'   CLS-compliant interface.

'
'   <CLSCompliant(False)> Function GetUnsigned As ULong
'   ~~~~~
```

Because of this rule, CLS-compliant types are not required to implement non-CLS-compliant members. If a CLS-compliant framework does expose a class that implements a non-CLS compliant interface, it should also provide concrete implementations of all non-CLS-compliant members.

CLS-compliant language compilers must also allow a class to provide separate implementations of members that have the same name and signature in multiple interfaces. Both C# and Visual Basic support [explicit interface implementations](#) to provide different implementations of identically named methods. Visual Basic also supports the `Implements` keyword, which enables you to explicitly designate which interface and member a particular member implements. The following example illustrates this scenario by defining a `Temperature` class that implements the `ICelsius` and `IFahrenheit` interfaces as explicit interface implementations.

```
using System;

[assembly: CLSCompliant(true)]

public interface IFahrenheit
{
    decimal GetTemperature();
}

public interface ICelsius
{
    decimal GetTemperature();
}

public class Temperature : ICelsius, IFahrenheit
{
    private decimal _value;

    public Temperature(decimal value)
    {
        // We assume that this is the Celsius value.
        _value = value;
    }

    decimal IFahrenheit.GetTemperature()
    {
        return _value * 9 / 5 + 32;
    }

    decimal ICelsius.GetTemperature()
    {
        return _value;
    }
}

public class Example
{
    public static void Main()
    {
        Temperature temp = new Temperature(100.0m);
        ICelsius cTemp = temp;
        IFahrenheit fTemp = temp;
        Console.WriteLine("Temperature in Celsius: {0} degrees",
                          cTemp.GetTemperature());
        Console.WriteLine("Temperature in Fahrenheit: {0} degrees",
                          fTemp.GetTemperature());
    }
}

// The example displays the following output:
//      Temperature in Celsius: 100.0 degrees
//      Temperature in Fahrenheit: 212.0 degrees
```

```

<Assembly: CLSCompliant(True)>

Public Interface IFahrenheit
    Function GetTemperature() As Decimal
End Interface

Public Interface ICelsius
    Function GetTemperature() As Decimal
End Interface

Public Class Temperature : Implements ICelsius, IFahrenheit
    Private _value As Decimal

    Public Sub New(value As Decimal)
        ' We assume that this is the Celsius value.
        _value = value
    End Sub

    Public Function GetFahrenheit() As Decimal _
        Implements IFahrenheit.GetTemperature
        Return _value * 9 / 5 + 32
    End Function

    Public Function GetCelsius() As Decimal _
        Implements ICelsius.GetTemperature
        Return _value
    End Function
End Class

Module Example
    Public Sub Main()
        Dim temp As New Temperature(100.0d)
        Console.WriteLine("Temperature in Celsius: {0} degrees",
            temp.GetCelsius())
        Console.WriteLine("Temperature in Fahrenheit: {0} degrees",
            temp.GetFahrenheit())
    End Sub
End Module
' The example displays the following output:
'     Temperature in Celsius: 100.0 degrees
'     Temperature in Fahrenheit: 212.0 degrees

```

Enumerations

CLS-compliant enumerations must follow these rules:

- The underlying type of the enumeration must be an intrinsic CLS-compliant integer ([Byte](#), [Int16](#), [Int32](#), or [Int64](#)). For example, the following code tries to define an enumeration whose underlying type is [UInt32](#) and generates a compiler warning.

```

using System;

[assembly: CLSCompliant(true)]

public enum Size : uint {
    Unspecified = 0,
    XSmall = 1,
    Small = 2,
    Medium = 3,
    Large = 4,
    XLarge = 5
};

public class Clothing
{
    public string Name;
    public string Type;
    public string Size;
}
// The attempt to compile the example displays a compiler warning like the following:
//     Enum3.cs(6,13): warning CS3009: 'Size': base type 'uint' is not CLS-compliant

```

```

<Assembly: CLSCompliant(True)>

Public Enum Size As UInt32
    Unspecified = 0
    XSmall = 1
    Small = 2
    Medium = 3
    Large = 4
    XLarge = 5
End Enum

Public Class Clothing
    Public Name As String
    Public Type As String
    Public Size As Size
End Class
' The attempt to compile the example displays a compiler warning like the following:
'     Enum3.vb(6) : warning BC40032: Underlying type 'UInt32' of Enum is not CLS-compliant.
'
'     Public Enum Size As UInt32
'
~~~~~
```

- An enumeration type must have a single instance field named `value__` that is marked with the [FieldAttributes.RTSpecialName](#) attribute. This enables you to reference the field value implicitly.
- An enumeration includes literal static fields whose types match the type of the enumeration itself. For example, if you define a `State` enumeration with values of `State.On` and `State.Off`, `State.On` and `State.Off` are both literal static fields whose type is `State`.
- There are two kinds of enumerations:
 - An enumeration that represents a set of mutually exclusive, named integer values. This type of enumeration is indicated by the absence of the [System.FlagsAttribute](#) custom attribute.
 - An enumeration that represents a set of bit flags that can combine to generate an unnamed value. This type of enumeration is indicated by the presence of the [System.FlagsAttribute](#) custom attribute.

For more information, see the documentation for the [Enum](#) structure.

- The value of an enumeration is not limited to the range of its specified values. In other words, the range of values in an enumeration is the range of its underlying value. You can use the [Enum.IsDefined](#) method to determine whether a specified value is a member of an enumeration.

Type members in general

The Common Language Specification requires all fields and methods to be accessed as members of a particular class. Therefore, global static fields and methods (that is, static fields or methods that are defined apart from a type) are not CLS-compliant. If you try to include a global field or method in your source code, both the C# and Visual Basic compilers generate a compiler error.

The Common Language Specification supports only the standard managed calling convention. It doesn't support unmanaged calling conventions and methods with variable argument lists marked with the `varargs` keyword. For variable argument lists that are compatible with the standard managed calling convention, use the [ParamArrayAttribute](#) attribute or the individual language's implementation, such as the `params` keyword in C# and the `ParamArray` keyword in Visual Basic.

Member accessibility

Overriding an inherited member cannot change the accessibility of that member. For example, a public method in a base class cannot be overridden by a private method in a derived class. There is one exception: a `protected internal` (in C#) or `Protected Friend` (in Visual Basic) member in one assembly that is overridden by a type in a different assembly. In that case, the accessibility of the override is `Protected`.

The following example illustrates the error that is generated when the [CLSCompliantAttribute](#) attribute is set to `true`, and `Human`, which is a class derived from `Animal`, tries to change the accessibility of the `Species` property from public to private. The example compiles successfully if its accessibility is changed to public.

```
using System;

[assembly: CLSCompliant(true)]

public class Animal
{
    private string _species;

    public Animal(string species)
    {
        _species = species;
    }

    public virtual string Species
    {
        get { return _species; }
    }

    public override string ToString()
    {
        return _species;
    }
}

public class Human : Animal
{
    private string _name;

    public Human(string name) : base("Homo Sapiens")
    {
        _name = name;
    }

    public string Name
    {
        get { return _name; }
    }

    private override string Species
    {
        get { return base.Species; }
    }

    public override string ToString()
    {
        return _name;
    }
}

public class Example
{
    public static void Main()
    {
        Human p = new Human("John");
        Console.WriteLine(p.Species);
        Console.WriteLine(p.ToString());
    }
}

// The example displays the following output:
//      error CS0621: 'Human.Species': virtual or abstract members cannot be private
```

```

<Assembly: CLSCompliant(True)>

Public Class Animal
    Private _species As String

    Public Sub New(species As String)
        _species = species
    End Sub

    Public Overridable ReadOnly Property Species As String
        Get
            Return _species
        End Get
    End Property

    Public Overrides Function ToString() As String
        Return _species
    End Function
End Class

Public Class Human : Inherits Animal
    Private _name As String

    Public Sub New(name As String)
        MyBase.New("Homo Sapiens")
        _name = name
    End Sub

    Public ReadOnly Property Name As String
        Get
            Return _name
        End Get
    End Property

    Private Overrides ReadOnly Property Species As String
        Get
            Return MyBase.Species
        End Get
    End Property

    Public Overrides Function ToString() As String
        Return _name
    End Function
End Class

Public Module Example
    Public Sub Main()
        Dim p As New Human("John")
        Console.WriteLine(p.Species)
        Console.WriteLine(p.ToString())
    End Sub
End Module

' The example displays the following output:
'     'Private Overrides ReadOnly Property Species As String' cannot override
'     'Public Overridable ReadOnly Property Species As String' because
'     they have different access levels.
'

'     Private Overrides ReadOnly Property Species As String

```

Types in the signature of a member must be accessible whenever that member is accessible. For example, this means that a public member cannot include a parameter whose type is private, protected, or internal. The following example illustrates the compiler error that results when a `StringWrapper` class constructor exposes an internal `StringOperationType` enumeration value that determines how a string value should be wrapped.

```

using System;
using System.Text;

public class StringWrapper
{
    string internalString;
    StringBuilder internalSB = null;
    bool useSB = false;

    public StringWrapper(StringOperationType type)
    {
        if (type == StringOperationType.Normal) {
            useSB = false;
        }
        else {
            useSB = true;
            internalSB = new StringBuilder();
        }
    }

    // The remaining source code...
}

internal enum StringOperationType { Normal, Dynamic }
// The attempt to compile the example displays the following output:
//     error CS0051: Inconsistent accessibility: parameter type
//           'StringOperationType' is less accessible than method
//           'StringWrapper.StringWrapper(StringOperationType)'

```

```

Imports System.Text

<Assembly: CLSCompliant(True)>

Public Class StringWrapper

    Dim internalString As String
    Dim internalSB As StringBuilder = Nothing
    Dim useSB As Boolean = False

    Public Sub New(type As StringOperationType)
        If type = StringOperationType.Normal Then
            useSB = False
        Else
            internalSB = New StringBuilder()
            useSB = True
        End If
    End Sub

    ' The remaining source code...
End Class

Friend Enum StringOperationType As Integer
    Normal = 0
    Dynamic = 1
End Enum
' The attempt to compile the example displays the following output:
'     error BC30909: 'type' cannot expose type 'StringOperationType'
'     outside the project through class 'StringWrapper'.
'

'     Public Sub New(type As StringOperationType)
'         ~~~~~

```

Generic types and members

Nested types always have at least as many generic parameters as their enclosing type. These correspond by

position to the generic parameters in the enclosing type. The generic type can also include new generic parameters.

The relationship between the generic type parameters of a containing type and its nested types may be hidden by the syntax of individual languages. In the following example, a generic type `Outer<T>` contains two nested classes, `Inner1A` and `Inner1B<U>`. The calls to the `ToString` method, which each class inherits from `Object.ToString()`, show that each nested class includes the type parameters of its containing class.

```
using System;

[assembly:CLSCompliant(true)]

public class Outer<T>
{
    T value;

    public Outer(T value)
    {
        this.value = value;
    }

    public class Inner1A : Outer<T>
    {
        public Inner1A(T value) : base(value)
        { }
    }

    public class Inner1B<U> : Outer<T>
    {
        U value2;

        public Inner1B(T value1, U value2) : base(value1)
        {
            this.value2 = value2;
        }
    }
}

public class Example
{
    public static void Main()
    {
        var inst1 = new Outer<String>("This");
        Console.WriteLine(inst1);

        var inst2 = new Outer<String>.Inner1A("Another");
        Console.WriteLine(inst2);

        var inst3 = new Outer<String>.Inner1B<int>("That", 2);
        Console.WriteLine(inst3);
    }
}

// The example displays the following output:
//      Outer`1[System.String]
//      Outer`1+Inner1A[System.String]
//      Outer`1+Inner1B`1[System.String,System.Int32]
```

```

<Assembly: CLSCompliant(True)>

Public Class Outer(Of T)
    Dim value As T

    Public Sub New(value As T)
        Me.value = value
    End Sub

    Public Class Inner1A : Inherits Outer(Of T)
        Public Sub New(value As T)
            MyBase.New(value)
        End Sub
    End Class

    Public Class Inner1B(Of U) : Inherits Outer(Of T)
        Dim value2 As U

        Public Sub New(value1 As T, value2 As U)
            MyBase.New(value1)
            Me.value2 = value2
        End Sub
    End Class
End Class

Public Module Example
    Public Sub Main()
        Dim inst1 As New Outer(Of String)("This")
        Console.WriteLine(inst1)

        Dim inst2 As New Outer(Of String).Inner1A("Another")
        Console.WriteLine(inst2)

        Dim inst3 As New Outer(Of String).Inner1B(Of Integer)("That", 2)
        Console.WriteLine(inst3)
    End Sub
End Module

' The example displays the following output:
' Outer`1[System.String]
' Outer`1+Inner1A[System.String]
' Outer`1+Inner1B`1[System.String,System.Int32]

```

Generic type names are encoded in the form *name`n*, where *name* is the type name, ` is a character literal, and *n* is the number of parameters declared on the type, or, for nested generic types, the number of newly introduced type parameters. This encoding of generic type names is primarily of interest to developers who use reflection to access CLS-compliant generic types in a library.

If constraints are applied to a generic type, any types used as constraints must also be CLS-compliant. The following example defines a class named `BaseClass` that is not CLS-compliant and a generic class named `BaseCollection` whose type parameter must derive from `BaseClass`. But because `BaseClass` is not CLS-compliant, the compiler emits a warning.

```
using System;

[assembly:CLSCompliant(true)]

[CLSCompliant(false)] public class BaseClass
{}

public class BaseCollection<T> where T : BaseClass
{}
// Attempting to compile the example displays the following output:
//   warning CS3024: Constraint type 'BaseClass' is not CLS-compliant
```

```
<Assembly: CLSCompliant(True)>

<CLSCompliant(False)> Public Class BaseClass
End Class

Public Class BaseCollection(Of T As BaseClass)
End Class
' Attempting to compile the example displays the following output:
'   warning BC40040: Generic parameter constraint type 'BaseClass' is not
'   CLS-compliant.
'
'   Public Class BaseCollection(Of T As BaseClass)
'   ~~~~~~
```

If a generic type is derived from a generic base type, it must redeclare any constraints so that it can guarantee that constraints on the base type are also satisfied. The following example defines a `Number<T>` that can represent any numeric type. It also defines a `FloatingPoint<T>` class that represents a floating-point value. However, the source code fails to compile, because it does not apply the constraint on `Number<T>` (that `T` must be a value type) to `FloatingPoint<T>`.

```
using System;

[assembly:CLSCompliant(true)]

public class Number<T> where T : struct
{
    // use Double as the underlying type, since its range is a superset of
    // the ranges of all numeric types except BigInteger.
    protected double number;

    public Number(T value)
    {
        try {
            this.number = Convert.ToDouble(value);
        }
        catch (OverflowException e) {
            throw new ArgumentException("value is too large.", e);
        }
        catch (InvalidCastException e) {
            throw new ArgumentException("The value parameter is not numeric.", e);
        }
    }

    public T Add(T value)
    {
        return (T) Convert.ChangeType(number + Convert.ToDouble(value), typeof(T));
    }

    public T Subtract(T value)
    {
        return (T) Convert.ChangeType(number - Convert.ToDouble(value), typeof(T));
    }
}

public class FloatingPoint<T> : Number<T>
{
    public FloatingPoint(T number) : base(number)
    {
        if (typeof(float) == number.GetType() ||
            typeof(double) == number.GetType() ||
            typeof(decimal) == number.GetType())
            this.number = Convert.ToDouble(number);
        else
            throw new ArgumentException("The number parameter is not a floating-point number.");
    }

    // The attempt to compile the example displays the following output:
    //     error CS0453: The type 'T' must be a non-nullable value type in
    //                 order to use it as parameter 'T' in the generic type or method 'Number<T>'
```

```

<Assembly: CLSCompliant(True)>

Public Class Number(Of T As Structure)
    ' Use Double as the underlying type, since its range is a superset of
    ' the ranges of all numeric types except BigInteger.
    Protected number As Double

    Public Sub New(value As T)
        Try
            Me.number = Convert.ToDouble(value)
        Catch e As OverflowException
            Throw New ArgumentException("value is too large.", e)
        Catch e As InvalidCastException
            Throw New ArgumentException("The value parameter is not numeric.", e)
        End Try
    End Sub

    Public Function Add(value As T) As T
        Return CType(Convert.ChangeType(number + Convert.ToDouble(value), GetType(T)), T)
    End Function

    Public Function Subtract(value As T) As T
        Return CType(Convert.ChangeType(number - Convert.ToDouble(value), GetType(T)), T)
    End Function
End Class

Public Class FloatingPoint(Of T) : Inherits Number(Of T)
    Public Sub New(number As T)
        MyBase.New(number)
        If TypeOf number Is Single Or
            TypeOf number Is Double Or
            TypeOf number Is Decimal Then
            Me.number = Convert.ToDouble(number)
        Else
            throw new ArgumentException("The number parameter is not a floating-point number.")
        End If
    End Sub
End Class
' The attempt to compile the example displays the following output:
' error BC32105: Type argument 'T' does not satisfy the 'Structure'
' constraint for type parameter 'T'.
'
' Public Class FloatingPoint(Of T) : Inherits Number(Of T)
' ~

```

The example compiles successfully if the constraint is added to the `FloatingPoint<T>` class.

```
using System;

[assembly:CLSCompliant(true)]

public class Number<T> where T : struct
{
    // use Double as the underlying type, since its range is a superset of
    // the ranges of all numeric types except BigInteger.
    protected double number;

    public Number(T value)
    {
        try {
            this.number = Convert.ToDouble(value);
        }
        catch (OverflowException e) {
            throw new ArgumentException("value is too large.", e);
        }
        catch (InvalidCastException e) {
            throw new ArgumentException("The value parameter is not numeric.", e);
        }
    }

    public T Add(T value)
    {
        return (T) Convert.ChangeType(number + Convert.ToDouble(value), typeof(T));
    }

    public T Subtract(T value)
    {
        return (T) Convert.ChangeType(number - Convert.ToDouble(value), typeof(T));
    }
}

public class FloatingPoint<T> : Number<T> where T : struct
{
    public FloatingPoint(T number) : base(number)
    {
        if (typeof(float) == number.GetType() ||
            typeof(double) == number.GetType() ||
            typeof(decimal) == number.GetType())
            this.number = Convert.ToDouble(number);
        else
            throw new ArgumentException("The number parameter is not a floating-point number.");
    }
}
```

```

<Assembly: CLSCompliant(True)>

Public Class Number(Of T As Structure)
    ' Use Double as the underlying type, since its range is a superset of
    ' the ranges of all numeric types except BigInteger.
    Protected number As Double

    Public Sub New(value As T)
        Try
            Me.number = Convert.ToDouble(value)
        Catch e As OverflowException
            Throw New ArgumentException("value is too large.", e)
        Catch e As InvalidCastException
            Throw New ArgumentException("The value parameter is not numeric.", e)
        End Try
    End Sub

    Public Function Add(value As T) As T
        Return CType(Convert.ChangeType(number + Convert.ToDouble(value), GetType(T)), T)
    End Function

    Public Function Subtract(value As T) As T
        Return CType(Convert.ChangeType(number - Convert.ToDouble(value), GetType(T)), T)
    End Function
End Class

Public Class FloatingPoint(Of T As Structure) : Inherits Number(Of T)
    Public Sub New(number As T)
        MyBase.New(number)
        If TypeOf number Is Single Or
            TypeOf number Is Double Or
            TypeOf number Is Decimal Then
            Me.number = Convert.ToDouble(number)
        Else
            throw new ArgumentException("The number parameter is not a floating-point number.")
        End If
    End Sub
End Class

```

The Common Language Specification imposes a conservative per-instantiation model for nested types and protected members. Open generic types cannot expose fields or members with signatures that contain a specific instantiation of a nested, protected generic type. Non-generic types that extend a specific instantiation of a generic base class or interface cannot expose fields or members with signatures that contain a different instantiation of a nested, protected generic type.

The following example defines a generic type, `C1<T>` (or `C1(of T)` in Visual Basic), and a protected class, `C1<T>.N` (or `C1(of T).N` in Visual Basic). `C1<T>` has two methods, `M1` and `M2`. However, `M1` is not CLS-compliant because it tries to return a `C1<int>.N` (or `C1(of Integer).N`) object from `C1<T>` (or `C1(of T)`). A second class, `C2`, is derived from `C1<long>` (or `C1(of Long)`). It has two methods, `M3` and `M4`. `M3` is not CLS-compliant because it tries to return a `C1<int>.N` (or `C1(of Integer).N`) object from a subclass of `C1<long>`. Language compilers can be even more restrictive. In this example, Visual Basic displays an error when it tries to compile `M4`.

```
using System;

[assembly:CLSCompliant(true)]

public class C1<T>
{
    protected class N { }

    protected void M1(C1<int>.N n) { } // Not CLS-compliant - C1<int>.N not
                                         // accessible from within C1<T> in all
                                         // languages
    protected void M2(C1<T>.N n) { } // CLS-compliant - C1<T>.N accessible
                                         // inside C1<T>
}

public class C2 : C1<long>
{
    protected void M3(C1<int>.N n) { } // Not CLS-compliant - C1<int>.N is not
                                         // accessible in C2 (extends C1<long>)

    protected void M4(C1<long>.N n) { } // CLS-compliant, C1<long>.N is
                                         // accessible in C2 (extends C1<long>)
}
// Attempting to compile the example displays output like the following:
//     Generics4.cs(9,22): warning CS3001: Argument type 'C1<int>.N' is not CLS-compliant
//     Generics4.cs(18,22): warning CS3001: Argument type 'C1<int>.N' is not CLS-compliant
```

```

<Assembly: CLSCompliant(True)>

Public Class C1(Of T)
    Protected Class N
        End Class

        Protected Sub M1(n As C1(Of Integer).N)      ' Not CLS-compliant - C1<int>.N not
            ' accessible from within C1(Of T) in all
            ' languages
        End Sub

        Protected Sub M2(n As C1(Of T).N)      ' CLS-compliant - C1(Of T).N accessible
            End Sub
        End Class

        Public Class C2 : Inherits C1(Of Long)
            Protected Sub M3(n As C1(Of Integer).N)      ' Not CLS-compliant - C1(Of Integer).N is not
                End Sub
                ' accessible in C2 (extends C1(Of Long))

            Protected Sub M4(n As C1(Of Long).N)
                End Sub
            End Class
        End Class

        ' Attempting to compile the example displays output like the following:
        ' error BC30508: 'n' cannot expose type 'C1(Of Integer).N' in namespace
        ' '<Default>' through class 'C1'.
        ' ~~~~~
        ' Protected Sub M1(n As C1(Of Integer).N)      ' Not CLS-compliant - C1<int>.N not
        ' ~~~~~
        ' error BC30389: 'C1(Of T).N' is not accessible in this context because
        ' it is 'Protected'.
        ' ~~~~~
        ' Protected Sub M3(n As C1(Of Integer).N)      ' Not CLS-compliant - C1(Of Integer).N is not
        ' ~~~~~
        ' error BC30389: 'C1(Of T).N' is not accessible in this context because it is 'Protected'.
        ' ~~~~~
        ' Protected Sub M4(n As C1(Of Long).N)
        ' ~~~~~

```

Constructors

Constructors in CLS-compliant classes and structures must follow these rules:

- A constructor of a derived class must call the instance constructor of its base class before it accesses inherited instance data. This requirement is because base class constructors are not inherited by their derived classes. This rule does not apply to structures, which do not support direct inheritance.

Typically, compilers enforce this rule independently of CLS compliance, as the following example shows. It creates a `Doctor` class that is derived from a `Person` class, but the `Doctor` class fails to call the `Person` class constructor to initialize inherited instance fields.

```
using System;

[assembly: CLSCompliant(true)]

public class Person
{
    private string fName, lName, _id;

    public Person(string firstName, string lastName, string id)
    {
        if (String.IsNullOrEmpty(firstName + lastName))
            throw new ArgumentNullException("Either a first name or a last name must be provided.");

        fName = firstName;
        lName = lastName;
        _id = id;
    }

    public string FirstName
    {
        get { return fName; }
    }

    public string LastName
    {
        get { return lName; }
    }

    public string Id
    {
        get { return _id; }
    }

    public override string ToString()
    {
        return String.Format("{0}{1}{2}", fName,
                             String.IsNullOrEmpty(fName) ? "" : " ",
                             lName);
    }
}

public class Doctor : Person
{
    public Doctor(string firstName, string lastName, string id)
    {

    }

    public override string ToString()
    {
        return "Dr. " + base.ToString();
    }
}

// Attempting to compile the example displays output like the following:
//  ctor1.cs(45,11): error CS1729: 'Person' does not contain a constructor that takes 0
//      arguments
//  ctor1.cs(10,11): (Location of symbol related to previous error)
```

```

<Assembly: CLSCompliant(True)>

Public Class Person
    Private fName, lName, _id As String

    Public Sub New(firstName As String, lastName As String, id As String)
        If String.IsNullOrEmpty(firstName + lastName) Then
            Throw New ArgumentNullException("Either a first name or a last name must be provided.")
        End If

        fName = firstName
        lName = lastName
        _id = id
    End Sub

    Public ReadOnly Property FirstName As String
        Get
            Return fName
        End Get
    End Property

    Public ReadOnly Property LastName As String
        Get
            Return lName
        End Get
    End Property

    Public ReadOnly Property Id As String
        Get
            Return _id
        End Get
    End Property

    Public Overrides Function ToString() As String
        Return String.Format("{0}{1}{2}", fName,
            If(String.IsNullOrEmpty(fName), "", " "),
            lName)
    End Function
End Class

Public Class Doctor : Inherits Person
    Public Sub New(firstName As String, lastName As String, id As String)
    End Sub

    Public Overrides Function ToString() As String
        Return "Dr. " + MyBase.ToString()
    End Function
End Class

' Attempting to compile the example displays output like the following:
' Ctor1.vb(46) : error BC30148: First statement of this 'Sub New' must be a call
' to 'MyBase.New' or 'MyClass.New' because base class 'Person' of 'Doctor' does
' not have an accessible 'Sub New' that can be called with no arguments.
'

'     Public Sub New()
'         ~~~

```

- An object constructor cannot be called except to create an object. In addition, an object cannot be initialized twice. For example, this means that [Object.MemberwiseClone](#) and deserialization methods such as [BinaryFormatter.Deserialize](#) must not call constructors.

Properties

Properties in CLS-compliant types must follow these rules:

- A property must have a setter, a getter, or both. In an assembly, these are implemented as special methods, which means that they will appear as separate methods (the getter is named `get_`

propertyname and the setter is `set_ propertyname)` marked as `SpecialName` in the assembly's metadata. The C# and Visual Basic compilers enforce this rule automatically without the need to apply the `CLSCompliantAttribute` attribute.

- A property's type is the return type of the property getter and the last argument of the setter. These types must be CLS compliant, and arguments cannot be assigned to the property by reference (that is, they cannot be managed pointers).
- If a property has both a getter and a setter, they must both be virtual, both static, or both instance. The C# and Visual Basic compilers automatically enforce this rule through their property definition syntax.

Events

An event is defined by its name and its type. The event type is a delegate that is used to indicate the event. For example, the `AppDomain.AssemblyResolve` event is of type `ResolveEventHandler`. In addition to the event itself, three methods with names based on the event name provide the event's implementation and are marked as `SpecialName` in the assembly's metadata:

- A method for adding an event handler, named `add_EventName`. For example, the event subscription method for the `AppDomain.AssemblyResolve` event is named `add_AssemblyResolve`.
- A method for removing an event handler, named `remove_EventName`. For example, the removal method for the `AppDomain.AssemblyResolve` event is named `remove_AssemblyResolve`.
- A method for indicating that the event has occurred, named `raise_EventName`.

NOTE

Most of the Common Language Specification's rules regarding events are implemented by language compilers and are transparent to component developers.

The methods for adding, removing, and raising the event must have the same accessibility. They must also all be static, instance, or virtual. The methods for adding and removing an event have one parameter whose type is the event delegate type. The add and remove methods must both be present or both be absent.

The following example defines a CLS-compliant class named `Temperature` that raises a `TemperatureChanged` event if the change in temperature between two readings equals or exceeds a threshold value. The `Temperature` class explicitly defines a `raise_TemperatureChanged` method so that it can selectively execute event handlers.

```
using System;
using System.Collections;
using System.Collections.Generic;

[assembly: CLSCompliant(true)]

public class TemperatureChangedEventArgs : EventArgs
{
    private Decimal originalTemp;
    private Decimal newTemp;
    private DateTimeOffset when;

    public TemperatureChangedEventArgs(Decimal original, Decimal @new, DateTimeOffset time)
    {
        originalTemp = original;
        newTemp = @new;
        when = time;
    }

    public Decimal OldTemperature
    {
        get { return originalTemp; }
    }
}
```

```

        get { return newTemp; }
    }

    public Decimal CurrentTemperature
    {
        get { return newTemp; }
    }

    public DateTimeOffset Time
    {
        get { return when; }
    }
}

public delegate void TemperatureChanged(Object sender, TemperatureChangedEventArgs e);

public class Temperature
{
    private struct TemperatureInfo
    {
        public Decimal Temperature;
        public DateTimeOffset Recorded;
    }

    public event TemperatureChanged TemperatureChanged;

    private Decimal previous;
    private Decimal current;
    private Decimal tolerance;
    private List<TemperatureInfo> tis = new List<TemperatureInfo>();

    public Temperature(Decimal temperature, Decimal tolerance)
    {
        current = temperature;
        TemperatureInfo ti = new TemperatureInfo();
        ti.Temperature = temperature;
        tis.Add(ti);
        ti.Recorded = DateTimeOffset.UtcNow;
        this.tolerance = tolerance;
    }

    public Decimal CurrentTemperature
    {
        get { return current; }
        set {
            TemperatureInfo ti = new TemperatureInfo();
            ti.Temperature = value;
            ti.Recorded = DateTimeOffset.UtcNow;
            previous = current;
            current = value;
            if (Math.Abs(current - previous) >= tolerance)
                raise_TemperatureChanged(new TemperatureChangedEventArgs(previous, current, ti.Recorded));
        }
    }

    public void raise_TemperatureChanged(TemperatureChangedEventArgs eventArgs)
    {
        if (TemperatureChanged == null)
            return;

        foreach (TemperatureChanged d in TemperatureChanged.GetInvocationList()) {
            if (d.Method.Name.Contains("Duplicate"))
                Console.WriteLine("Duplicate event handler; event handler not executed.");
            else
                d.Invoke(this, eventArgs);
        }
    }
}

```

```

public class Example
{
    public Temperature temp;

    public static void Main()
    {
        Example ex = new Example();
    }

    public Example()
    {
        temp = new Temperature(65, 3);
        temp.TemperatureChanged += this.TemperatureNotification;
        RecordTemperatures();
        Example ex = new Example(temp);
        ex.RecordTemperatures();
    }

    public Example(Temperature t)
    {
        temp = t;
        RecordTemperatures();
    }

    public void RecordTemperatures()
    {
        temp.TemperatureChanged += this.DuplicateTemperatureNotification;
        temp.CurrentTemperature = 66;
        temp.CurrentTemperature = 63;
    }

    internal void TemperatureNotification(Object sender, TemperatureChangedEventArgs e)
    {
        Console.WriteLine("Notification 1: The temperature changed from {0} to {1}", e.OldTemperature,
e.CurrentTemperature);
    }

    public void DuplicateTemperatureNotification(Object sender, TemperatureChangedEventArgs e)
    {
        Console.WriteLine("Notification 2: The temperature changed from {0} to {1}", e.OldTemperature,
e.CurrentTemperature);
    }
}

```

```

Imports System.Collections
Imports System.Collections.Generic

<Assembly: CLSCompliant(True)>

Public Class TemperatureChangedEventArgs : Inherits EventArgs
    Private originalTemp As Decimal
    Private newTemp As Decimal
    Private [when] As DateTimeOffset

    Public Sub New(original As Decimal, [new] As Decimal, [time] As DateTimeOffset)
        originalTemp = original
        newTemp = [new]
        [when] = [time]
    End Sub

    Public ReadOnly Property OldTemperature As Decimal
        Get
            Return originalTemp
        End Get
    End Property

    Public ReadOnly Property CurrentTemperature As Decimal

```

```

    Get
        Return newTemp
    End Get
End Property

Public ReadOnly Property [Time] As DateTimeOffset
    Get
        Return [when]
    End Get
End Property
End Class

Public Delegate Sub TemperatureChanged(sender As Object, e As TemperatureChangedEventArgs)

Public Class Temperature
    Private Structure TemperatureInfo
        Dim Temperature As Decimal
        Dim Recorded As DateTimeOffset
    End Structure

    Public Event TemperatureChanged As TemperatureChanged

    Private previous As Decimal
    Private current As Decimal
    Private tolerance As Decimal
    Private tis As New List(Of TemperatureInfo)

    Public Sub New(temperature As Decimal, tolerance As Decimal)
        current = temperature
        Dim ti As New TemperatureInfo()
        ti.Temperature = temperature
        ti.Recorded = DateTimeOffset.UtcNow
        tis.Add(ti)
        Me.tolerance = tolerance
    End Sub

    Public Property CurrentTemperature As Decimal
        Get
            Return current
        End Get
        Set
            Dim ti As New TemperatureInfo()
            ti.Temperature = value
            ti.Recorded = DateTimeOffset.UtcNow
            previous = current
            current = value
            If Math.Abs(current - previous) >= tolerance Then
                raise_TemperatureChanged(New TemperatureChangedEventArgs(previous, current, ti.Recorded))
            End If
        End Set
    End Property

    Public Sub raise_TemperatureChanged(eventArgs As TemperatureChangedEventArgs)
        If TemperatureChangedEvent Is Nothing Then Exit Sub

        Dim ListenerList() As System.Delegate = TemperatureChangedEvent.GetInvocationList()
        For Each d As TemperatureChanged In TemperatureChangedEvent.GetInvocationList()
            If d.Method.Name.Contains("Duplicate") Then
                Console.WriteLine("Duplicate event handler; event handler not executed.")
            Else
                d.Invoke(Me, eventArgs)
            End If
        Next
    End Sub
End Class

Public Class Example
    Public WithEvents temp As Temperature

```

```

Public Shared Sub Main()
    Dim ex As New Example()
End Sub

Public Sub New()
    temp = New Temperature(65, 3)
    RecordTemperatures()
    Dim ex As New Example(temp)
    ex.RecordTemperatures()
End Sub

Public Sub New(t As Temperature)
    temp = t
    RecordTemperatures()
End Sub

Public Sub RecordTemperatures()
    temp.CurrentTemperature = 66
    temp.CurrentTemperature = 63
End Sub

Friend Shared Sub TemperatureNotification(sender As Object, e As TemperatureChangedEventArgs) _
    Handles temp.TemperatureChanged
    Console.WriteLine("Notification 1: The temperature changed from {0} to {1}", e.OldTemperature,
e.CurrentTemperature)
End Sub

Friend Shared Sub DuplicateTemperatureNotification(sender As Object, e As TemperatureChangedEventArgs) _
    Handles temp.TemperatureChanged
    Console.WriteLine("Notification 2: The temperature changed from {0} to {1}", e.OldTemperature,
e.CurrentTemperature)
End Sub
End Class

```

Overloads

The Common Language Specification imposes the following requirements on overloaded members:

- Members can be overloaded based on the number of parameters and the type of any parameter. Calling convention, return type, custom modifiers applied to the method or its parameter, and whether parameters are passed by value or by reference are not considered when differentiating between overloads. For an example, see the code for the requirement that names must be unique within a scope in the [Naming conventions](#) section.
- Only properties and methods can be overloaded. Fields and events cannot be overloaded.
- Generic methods can be overloaded based on the number of their generic parameters.

NOTE

The `op_Explicit` and `op_Implicit` operators are exceptions to the rule that return value is not considered part of a method signature for overload resolution. These two operators can be overloaded based on both their parameters and their return value.

Exceptions

Exception objects must derive from [System.Exception](#) or from another type derived from [System.Exception](#). The following example illustrates the compiler error that results when a custom class named `ErrorClass` is used for exception handling.

```
using System;

[assembly: CLSCompliant(true)]

public class ErrorClass
{
    string msg;

    public ErrorClass(string errorMessage)
    {
        msg = errorMessage;
    }

    public string Message
    {
        get { return msg; }
    }
}

public static class StringUtilities
{
    public static string[] SplitString(this string value, int index)
    {
        if (index < 0 | index > value.Length) {
            ErrorClass badIndex = new ErrorClass("The index is not within the string.");
            throw badIndex;
        }
        string[] retVal = { value.Substring(0, index - 1),
                           value.Substring(index) };
        return retVal;
    }
}
// Compilation produces a compiler error like the following:
//     Exceptions1.cs(26,16): error CS0155: The type caught or thrown must be derived from
//         System.Exception
```

```

Imports System.Runtime.CompilerServices

<Assembly: CLSCompliant(True)>

Public Class ErrorClass
    Dim msg As String

    Public Sub New(errorMessage As String)
        msg = errorMessage
    End Sub

    Public ReadOnly Property Message As String
        Get
            Return msg
        End Get
    End Property
End Class

Public Module StringUtilities
    <Extension()> Public Function SplitString(value As String, index As Integer) As String()
        If index < 0 Or index > value.Length Then
            Dim BadIndex As New ErrorClass("The index is not within the string.")
            Throw BadIndex
        End If
        Dim retVal() As String = {value.Substring(0, index - 1),
                                 value.Substring(index)}
        Return retVal
    End Function
End Module

' Compilation produces a compiler error like the following:
'   Exceptions1.vb(27) : error BC30665: 'Throw' operand must derive from 'System.Exception'.
'
'       Throw BadIndex
'
~~~~~
```

To correct this error, the `ErrorClass` class must inherit from `System.Exception`. In addition, the `Message` property must be overridden. The following example corrects these errors to define an `ErrorClass` class that is CLS-compliant.

```

using System;

[assembly: CLSCompliant(true)]

public class ErrorClass : Exception
{
    string msg;

    public ErrorClass(string errorMessage)
    {
        msg = errorMessage;
    }

    public override string Message
    {
        get { return msg; }
    }
}

public static class StringUtilities
{
    public static string[] SplitString(this string value, int index)
    {
        if (index < 0 | index > value.Length) {
            ErrorClass badIndex = new ErrorClass("The index is not within the string.");
            throw badIndex;
        }
        string[] retVal = { value.Substring(0, index - 1),
                           value.Substring(index) };
        return retVal;
    }
}

```

```

Imports System.Runtime.CompilerServices

<Assembly: CLSCompliant(True)>

Public Class ErrorClass : Inherits Exception
    Dim msg As String

    Public Sub New(errorMessage As String)
        msg = errorMessage
    End Sub

    Public Overrides ReadOnly Property Message As String
        Get
            Return msg
        End Get
    End Property
End Class

Public Module StringUtilities
    <Extension()> Public Function SplitString(value As String, index As Integer) As String()
        If index < 0 Or index > value.Length Then
            Dim BadIndex As New ErrorClass("The index is not within the string.")
            Throw BadIndex
        End If
        Dim retVal() As String = {value.Substring(0, index - 1),
                                 value.Substring(index)}
        Return retVal
    End Function
End Module

```

Attributes

In .NET assemblies, custom attributes provide an extensible mechanism for storing custom attributes and retrieving metadata about programming objects, such as assemblies, types, members, and method parameters. Custom attributes must derive from [System.Attribute](#) or from a type derived from [System.Attribute](#).

The following example violates this rule. It defines a [NumericAttribute](#) class that does not derive from [System.Attribute](#). A compiler error results only when the non-CLS-compliant attribute is applied, not when the class is defined.

```
using System;

[assembly: CLSCompliant(true)]

[AttributeUsageAttribute(AttributeTargets.Class | AttributeTargets.Struct)]
public class NumericAttribute
{
    private bool _isNumeric;

    public NumericAttribute(bool isNumeric)
    {
        _isNumeric = isNumeric;
    }

    public bool IsNumeric
    {
        get { return _isNumeric; }
    }
}

[Numeric(true)] public struct UDouble
{
    double Value;
}
// Compilation produces a compiler error like the following:
//     Attribute1.cs(22,2): error CS0616: 'NumericAttribute' is not an attribute class
//     Attribute1.cs(7,14): (Location of symbol related to previous error)
```

```
<Assembly: CLSCompliant(True)>

<AttributeUsageAttribute(AttributeTargets.Class Or AttributeTargets.Struct)> _
Public Class NumericAttribute
    Private _isNumeric As Boolean

    Public Sub New(isNumeric As Boolean)
        _isNumeric = isNumeric
    End Sub

    Public ReadOnly Property IsNumeric As Boolean
        Get
            Return _isNumeric
        End Get
    End Property
End Class

<Numeric(True)> Public Structure UDouble
    Dim Value As Double
End Structure
' Compilation produces a compiler error like the following:
'     error BC31504: 'NumericAttribute' cannot be used as an attribute because it
'     does not inherit from 'System.Attribute'.
'
'     <Numeric(True)> Public Structure UDouble
'     ~~~~~~
```

The constructor or the properties of a CLS-compliant attribute can expose only the following types:

- [Boolean](#)
- [Byte](#)
- [Char](#)
- [Double](#)
- [Int16](#)
- [Int32](#)
- [Int64](#)
- [Single](#)
- [String](#)
- [Type](#)
- Any enumeration type whose underlying type is [Byte](#), [Int16](#), [Int32](#), or [Int64](#).

The following example defines a `DescriptionAttribute` class that derives from [Attribute](#). The class constructor has a parameter of type `Descriptor`, so the class is not CLS-compliant. The C# compiler emits a warning but compiles successfully, whereas the Visual Basic compiler doesn't emit a warning or an error.

```
using System;

[assembly:CLSCompliantAttribute(true)]

public enum DescriptorType { type, member };

public class Descriptor
{
    public DescriptorType Type;
    public String Description;
}

[AttributeUsage(AttributeTargets.All)]
public class DescriptionAttribute : Attribute
{
    private Descriptor desc;

    public DescriptionAttribute(Descriptor d)
    {
        desc = d;
    }

    public Descriptor Descriptor
    { get { return desc; } }
}

// Attempting to compile the example displays output like the following:
//      warning CS3015: 'DescriptionAttribute' has no accessible
//                      constructors which use only CLS-compliant types
```

```

<Assembly: CLSCompliantAttribute(True)>

Public Enum DescriptorType As Integer
    Type = 0
    Member = 1
End Enum

Public Class Descriptor
    Public Type As DescriptorType
    Public Description As String
End Class

<AttributeUsage(AttributeTargets.All)> _
Public Class DescriptionAttribute : Inherits Attribute
    Private desc As Descriptor

    Public Sub New(d As Descriptor)
        desc = d
    End Sub

    Public ReadOnly Property Descriptor As Descriptor
        Get
            Return desc
        End Get
    End Property
End Class

```

The `CLSCompliantAttribute` attribute

The `CLSCompliantAttribute` attribute is used to indicate whether a program element complies with the Common Language Specification. The `CLSCompliantAttribute(Boolean)` constructor includes a single required parameter, `isCompliant`, that indicates whether the program element is CLS-compliant.

At compile time, the compiler detects non-compliant elements that are presumed to be CLS-compliant and emits a warning. The compiler does not emit warnings for types or members that are explicitly declared to be non-compliant.

Component developers can use the `CLSCompliantAttribute` attribute in two ways:

- To define the parts of the public interface exposed by a component that are CLS-compliant and the parts that are not CLS-compliant. When the attribute is used to mark particular program elements as CLS-compliant, its use guarantees that those elements are accessible from all languages and tools that target .NET.
- To ensure that the component library's public interface exposes only program elements that are CLS-compliant. If elements are not CLS-compliant, compilers will generally issue a warning.

WARNING

In some cases, language compilers enforce CLS-compliant rules regardless of whether the `CLSCompliantAttribute` attribute is used. For example, defining a static member in an interface violates a CLS rule. In this regard, if you define `static` (in C#) or `Shared` (in Visual Basic) member in an interface, both the C# and Visual Basic compilers display an error message and fail to compile the app.

The `CLSCompliantAttribute` attribute is marked with an `AttributeUsageAttribute` attribute that has a value of `AttributeTargets.All`. This value allows you to apply the `CLSCompliantAttribute` attribute to any program element, including assemblies, modules, types (classes, structures, enumerations, interfaces, and delegates), type members (constructors, methods, properties, fields, and events), parameters, generic parameters, and return

values. However, in practice, you should apply the attribute only to assemblies, types, and type members. Otherwise, compilers ignore the attribute and continue to generate compiler warnings whenever they encounter a non-compliant parameter, generic parameter, or return value in your library's public interface.

The value of the `CLSCCompliantAttribute` attribute is inherited by contained program elements. For example, if an assembly is marked as CLS-compliant, its types are also CLS-compliant. If a type is marked as CLS-compliant, its nested types and members are also CLS-compliant.

You can explicitly override the inherited compliance by applying the `CLSCCompliantAttribute` attribute to a contained program element. For example, you can use the `CLSCCompliantAttribute` attribute with an `isCompliant` value of `false` to define a non-compliant type in a compliant assembly, and you can use the attribute with an `isCompliant` value of `true` to define a compliant type in a non-compliant assembly. You can also define non-compliant members in a compliant type. However, a non-compliant type cannot have compliant members, so you cannot use the attribute with an `isCompliant` value of `true` to override inheritance from a non-compliant type.

When you are developing components, you should always use the `CLSCCompliantAttribute` attribute to indicate whether your assembly, its types, and its members are CLS-compliant.

To create CLS-compliant components:

1. Use the `CLSCCompliantAttribute` to mark your assembly as CLS-compliant.
2. Mark any publicly exposed types in the assembly that are not CLS-compliant as non-compliant.
3. Mark any publicly exposed members in CLS-compliant types as non-compliant.
4. Provide a CLS-compliant alternative for non-CLS-compliant members.

If you've successfully marked all your non-compliant types and members, your compiler should not emit any non-compliance warnings. However, you should indicate which members are not CLS-compliant and list their CLS-compliant alternatives in your product documentation.

The following example uses the `CLSCCompliantAttribute` attribute to define a CLS-compliant assembly and a type, `CharacterUtilities`, that has two non-CLS-compliant members. Because both members are tagged with the `CLSCCompliant(false)` attribute, the compiler produces no warnings. The class also provides a CLS-compliant alternative for both methods. Ordinarily, we would just add two overloads to the `ToUTF16` method to provide CLS-compliant alternatives. However, because methods cannot be overloaded based on return value, the names of the CLS-compliant methods are different from the names of the non-compliant methods.

```

using System;
using System.Text;

[assembly:CLSCompliant(true)]

public class CharacterUtilities
{
    [CLSCompliant(false)] public static ushort ToUTF16(String s)
    {
        s = s.Normalize(NormalizationForm.FormC);
        return Convert.ToInt16(s[0]);
    }

    [CLSCompliant(false)] public static ushort ToUTF16(Char ch)
    {
        return Convert.ToInt16(ch);
    }

    // CLS-compliant alternative for ToUTF16(String).
    public static int ToUTF16CodeUnit(String s)
    {
        s = s.Normalize(NormalizationForm.FormC);
        return (int) Convert.ToInt16(s[0]);
    }

    // CLS-compliant alternative for ToUTF16(Char).
    public static int ToUTF16CodeUnit(Char ch)
    {
        return Convert.ToInt32(ch);
    }

    public bool HasMultipleRepresentations(String s)
    {
        String s1 = s.Normalize(NormalizationForm.FormC);
        return s.Equals(s1);
    }

    public int GetUnicodeCodePoint(Char ch)
    {
        if (Char.IsSurrogate(ch))
            throw new ArgumentException("ch cannot be a high or low surrogate.");

        return Char.ConvertToUtf32(ch.ToString(), 0);
    }

    public int GetUnicodeCodePoint(Char[] chars)
    {
        if (chars.Length > 2)
            throw new ArgumentException("The array has too many characters.");

        if (chars.Length == 2) {
            if (! Char.IsSurrogatePair(chars[0], chars[1]))
                throw new ArgumentException("The array must contain a low and a high surrogate.");
            else
                return Char.ConvertToUtf32(chars[0], chars[1]);
        }
        else {
            return Char.ConvertToUtf32(chars.ToString(), 0);
        }
    }
}

```

```

Imports System.Text

<Assembly: CLSCompliant(True)>

Public Class CharacterUtilities
    <CLSCompliant(False)> Public Shared Function ToUTF16(s As String) As UShort
        s = s.Normalize(NormalizationForm.FormC)
        Return Convert.ToUInt16(s(0))
    End Function

    <CLSCompliant(False)> Public Shared Function ToUTF16(ch As Char) As UShort
        Return Convert.ToInt16(ch)
    End Function

    ' CLS-compliant alternative for ToUTF16(String).
    Public Shared Function ToUTF16CodeUnit(s As String) As Integer
        s = s.Normalize(NormalizationForm.FormC)
        Return CInt(Convert.ToInt16(s(0)))
    End Function

    ' CLS-compliant alternative for ToUTF16(Char).
    Public Shared Function ToUTF16CodeUnit(ch As Char) As Integer
        Return Convert.ToInt32(ch)
    End Function

    Public Function HasMultipleRepresentations(s As String) As Boolean
        Dim s1 As String = s.Normalize(NormalizationForm.FormC)
        Return s.Equals(s1)
    End Function

    Public Function GetUnicodeCodePoint(ch As Char) As Integer
        If Char.IsSurrogate(ch) Then
            Throw New ArgumentException("ch cannot be a high or low surrogate.")
        End If
        Return Char.ConvertToUtf32(ch.ToString(), 0)
    End Function

    Public Function GetUnicodeCodePoint(chars() As Char) As Integer
        If chars.Length > 2 Then
            Throw New ArgumentException("The array has too many characters.")
        End If
        If chars.Length = 2 Then
            If Not Char.IsSurrogatePair(chars(0), chars(1)) Then
                Throw New ArgumentException("The array must contain a low and a high surrogate.")
            Else
                Return Char.ConvertToUtf32(chars(0), chars(1))
            End If
        Else
            Return Char.ConvertToUtf32(chars.ToString(), 0)
        End If
    End Function
End Class

```

If you are developing an app rather than a library (that is, if you aren't exposing types or members that can be consumed by other app developers), the CLS compliance of the program elements that your app consumes are of interest only if your language does not support them. In that case, your language compiler will generate an error when you try to use a non-CLS-compliant element.

Cross-language interoperability

Language independence has a few possible meanings. One meaning involves seamlessly consuming types written in one language from an app written in another language. A second meaning, which is the focus of this article, involves combining code written in multiple languages into a single .NET assembly.

The following example illustrates cross-language interoperability by creating a class library named Utilities.dll that includes two classes, `NumericLib` and `StringLib`. The `NumericLib` class is written in C#, and the `StringLib` class is written in Visual Basic. Here's the source code for `StringUtil.vb`, which includes a single member, `ToTitleCase`, in its `StringLib` class.

```
Imports System.Collections.Generic
Imports System.Runtime.CompilerServices

Public Module StringLib
    Private exclusions As List(Of String)

    Sub New()
        Dim words() As String = {"a", "an", "and", "of", "the"}
        exclusions = New List(Of String)
        exclusions.AddRange(words)
    End Sub

    <Extension()> _
    Public Function ToTitleCase(title As String) As String
        Dim words() As String = title.Split()
        Dim result As String = String.Empty

        For ctr As Integer = 0 To words.Length - 1
            Dim word As String = words(ctr)
            If ctr = 0 OrElse Not exclusions.Contains(word.ToLower()) Then
                result += word.Substring(0, 1).ToUpper() + _
                          word.Substring(1).ToLower()
            Else
                result += word.ToLower()
            End If
            If ctr <= words.Length - 1 Then
                result += " "
            End If
        Next
        Return result
    End Function
End Module
```

Here's the source code for NumberUtil.cs, which defines a `NumericLib` class that has two members, `IsEven` and `NearZero`.

```
using System;

public static class NumericLib
{
    public static bool IsEven(this IConvertible number)
    {
        if (number is Byte ||
            number is SByte ||
            number is Int16 ||
            number is UInt16 ||
            number is Int32 ||
            number is UInt32 ||
            number is Int64)
            return Convert.ToInt64(number) % 2 == 0;
        else if (number is UInt64)
            return ((ulong) number) % 2 == 0;
        else
            throw new NotSupportedException("IsEven called for a non-integer value.");
    }

    public static bool NearZero(double number)
    {
        return Math.Abs(number) < .00001;
    }
}
```

To package the two classes in a single assembly, you must compile them into modules. To compile the Visual Basic source code file into a module, use this command:

```
vbc /t:module StringUtil.vb
```

For more information about the command-line syntax of the Visual Basic compiler, see [Building from the Command Line](#).

To compile the C# source code file into a module, use this command:

```
csc /t:module NumberUtil.cs
```

You then use the [Linker options](#) to compile the two modules into an assembly:

```
link numberutil.netmodule stringutil.netmodule /out:UtilityLib.dll /dl1
```

The following example then calls the `NumericLib.NearZero` and `StringLib.ToTitleCase` methods. Both the Visual Basic code and the C# code are able to access the methods in both classes.

```
using System;

public class Example
{
    public static void Main()
    {
        Double dbl = 0.0 - Double.Epsilon;
        Console.WriteLine(NumericLib.NearZero(dbl));

        string s = "war and peace";
        Console.WriteLine(s.ToTitleCase());
    }
}

// The example displays the following output:
//      True
//      War and Peace
```

```
Module Example
    Public Sub Main()
        Dim dbl As Double = 0.0 - Double.Epsilon
        Console.WriteLine(NumericLib.NearZero(dbl))

        Dim s As String = "war and peace"
        Console.WriteLine(s.ToTitleCase())
    End Sub
End Module

' The example displays the following output:
'      True
'      War and Peace
```

To compile the Visual Basic code, use this command:

```
vbc example.vb /r:UtilityLib.dll
```

To compile with C#, change the name of the compiler from `vbc` to `csc`, and change the file extension from `.vb` to `.cs`:

```
csc example.cs /r:UtilityLib.dll
```

Type conversion in .NET

9/20/2022 • 30 minutes to read • [Edit Online](#)

Every value has an associated type, which defines attributes such as the amount of space allocated to the value, the range of possible values it can have, and the members that it makes available. Many values can be expressed as more than one type. For example, the value 4 can be expressed as an integer or a floating-point value. Type conversion creates a value in a new type that is equivalent to the value of an old type, but does not necessarily preserve the identity (or exact value) of the original object.

.NET automatically supports the following conversions:

- Conversion from a derived class to a base class. This means, for example, that an instance of any class or structure can be converted to an [Object](#) instance. This conversion does not require a casting or conversion operator.
- Conversion from a base class back to the original derived class. In C#, this conversion requires a casting operator. In Visual Basic, it requires the `CType` operator if `Option Strict` is on.
- Conversion from a type that implements an interface to an interface object that represents that interface. This conversion does not require a casting or conversion operator.
- Conversion from an interface object back to the original type that implements that interface. In C#, this conversion requires a casting operator. In Visual Basic, it requires the `CType` operator if `Option Strict` is on.

In addition to these automatic conversions, .NET provides several features that support custom type conversion. These include the following:

- The `Implicit` operator, which defines the available widening conversions between types. For more information, see the [Implicit Conversion with the Implicit Operator](#) section.
- The `Explicit` operator, which defines the available narrowing conversions between types. For more information, see the [Explicit Conversion with the Explicit Operator](#) section.
- The [IConvertible](#) interface, which defines conversions to each of the base .NET data types. For more information, see [The IConvertible Interface](#) section.
- The [Convert](#) class, which provides a set of methods that implement the methods in the [IConvertible](#) interface. For more information, see [The Convert Class](#) section.
- The [TypeConverter](#) class, which is a base class that can be extended to support the conversion of a specified type to any other type. For more information, see [The TypeConverter Class](#) section.

Implicit conversion with the implicit operator

Widening conversions involve the creation of a new value from the value of an existing type that has either a more restrictive range or a more restricted member list than the target type. Widening conversions cannot result in data loss (although they may result in a loss of precision). Because data cannot be lost, compilers can handle the conversion implicitly or transparently, without requiring the use of an explicit conversion method or a casting operator.

NOTE

Although code that performs an implicit conversion can call a conversion method or use a casting operator, their use is not required by compilers that support implicit conversions.

For example, the [Decimal](#) type supports implicit conversions from [Byte](#), [Char](#), [Int16](#), [Int32](#), [Int64](#), [SByte](#), [UInt16](#), [UInt32](#), and [UInt64](#) values. The following example illustrates some of these implicit conversions in assigning values to a [Decimal](#) variable.

```
byte byteValue = 16;
short shortValue = -1024;
int intValue = -1034000;
long longValue = 1152921504606846976;
ulong ulongValue = UInt64.MaxValue;

decimal decimalValue;

decimalValue = byteValue;
Console.WriteLine("After assigning a {0} value, the Decimal value is {1}.",
    byteValue.GetType().Name, decimalValue);

decimalValue = shortValue;
Console.WriteLine("After assigning a {0} value, the Decimal value is {1}.",
    shortValue.GetType().Name, decimalValue);

decimalValue = intValue;
Console.WriteLine("After assigning a {0} value, the Decimal value is {1}.",
    intValue.GetType().Name, decimalValue);

decimalValue = longValue;
Console.WriteLine("After assigning a {0} value, the Decimal value is {1}.",
    longValue.GetType().Name, decimalValue);

decimalValue = ulongValue;
Console.WriteLine("After assigning a {0} value, the Decimal value is {1}.",
    ulongValue.GetType().Name, decimalValue);
// The example displays the following output:
//   After assigning a Byte value, the Decimal value is 16.
//   After assigning a Int16 value, the Decimal value is -1024.
//   After assigning a Int32 value, the Decimal value is -1034000.
//   After assigning a Int64 value, the Decimal value is 1152921504606846976.
//   After assigning a UInt64 value, the Decimal value is 18446744073709551615.
```

```

Dim byteValue As Byte = 16
Dim shortValue As Short = -1024
Dim intValue As Integer = -1034000
Dim longValue As Long = CLng(1024 ^ 6)
Dim ulongValue As ULong = ULong.MaxValue

Dim decimalValue As Decimal

decimalValue = byteValue
Console.WriteLine("After assigning a {0} value, the Decimal value is {1}.",
    byteValue.GetType().Name, decimalValue)

decimalValue = shortValue
Console.WriteLine("After assigning a {0} value, the Decimal value is {1}.",
    shortValue.GetType().Name, decimalValue)

decimalValue = intValue
Console.WriteLine("After assigning a {0} value, the Decimal value is {1}.",
    intValue.GetType().Name, decimalValue)

decimalValue = longValue
Console.WriteLine("After assigning a {0} value, the Decimal value is {1}.",
    longValue.GetType().Name, decimalValue)

decimalValue = ulongValue
Console.WriteLine("After assigning a {0} value, the Decimal value is {1}.",
    ulongValue.GetType().Name, decimalValue)

' The example displays the following output:
'   After assigning a Byte value, the Decimal value is 16.
'   After assigning a Int16 value, the Decimal value is -1024.
'   After assigning a Int32 value, the Decimal value is -1034000.
'   After assigning a Int64 value, the Decimal value is 1152921504606846976.
'   After assigning a UInt64 value, the Decimal value is 18446744073709551615.

```

If a particular language compiler supports custom operators, you can also define implicit conversions in your own custom types. The following example provides a partial implementation of a signed byte data type named `ByteWithSign` that uses sign-and-magnitude representation. It supports implicit conversion of `Byte` and `SByte` values to `ByteWithSign` values.

```

public struct ByteWithSign
{
    private SByte signValue;
    private Byte value;

    public static implicit operator ByteWithSign(SByte value)
    {
        ByteWithSign newValue;
        newValue.signValue = (SByte)Math.Sign(value);
        newValue.value = (byte)Math.Abs(value);
        return newValue;
    }

    public static implicit operator ByteWithSign(Byte value)
    {
        ByteWithSign newValue;
        newValue.signValue = 1;
        newValue.value = value;
        return newValue;
    }

    public override string ToString()
    {
        return (signValue * value).ToString();
    }
}

```

```

Public Structure ImplicitByteWithSign
    Private signValue As SByte
    Private value As Byte

    Public Overloads Shared Widening Operator CType(value As SByte) As ImplicitByteWithSign
        Dim newValue As ImplicitByteWithSign
        newValue.signValue = CSByte(Math.Sign(value))
        newValue.value = CByte(Math.Abs(value))
        Return newValue
    End Operator

    Public Overloads Shared Widening Operator CType(value As Byte) As ImplicitByteWithSign
        Dim newValue As ImplicitByteWithSign
        newValue.signValue = 1
        newValue.value = value
        Return newValue
    End Operator

    Public Overrides Function ToString() As String
        Return (signValue * value).ToString()
    End Function
End Structure

```

Client code can then declare a `ByteWithSign` variable and assign it `Byte` and `SByte` values without performing any explicit conversions or using any casting operators, as the following example shows.

```

SByte sbyteValue = -120;
ByteWithSign value = sbyteValue;
Console.WriteLine(value);
value = Byte.MaxValue;
Console.WriteLine(value);
// The example displays the following output:
//      -120
//      255

```

```

Dim sbyteValue As SByte = -120
Dim value As ImplicitByteWithSign = sbyteValue
Console.WriteLine(value.ToString())
value = Byte.MaxValue
Console.WriteLine(value.ToString())
' The example displays the following output:
'      -120
'      255

```

Explicit conversion with the explicit operator

Narrowing conversions involve the creation of a new value from the value of an existing type that has either a greater range or a larger member list than the target type. Because a narrowing conversion can result in a loss of data, compilers often require that the conversion be made explicit through a call to a conversion method or a casting operator. That is, the conversion must be handled explicitly in developer code.

NOTE

The major purpose of requiring a conversion method or casting operator for narrowing conversions is to make the developer aware of the possibility of data loss or an [OverflowException](#) so that it can be handled in code. However, some compilers can relax this requirement. For example, in Visual Basic, if `Option Strict` is off (its default setting), the Visual Basic compiler tries to perform narrowing conversions implicitly.

For example, the `UInt32`, `Int64`, and `UInt64` data types have ranges that exceed that the `Int32` data type, as the following table shows.

TYPE	COMPARISON WITH RANGE OF INT32
<code>Int64</code>	<code>Int64.MaxValue</code> is greater than <code>Int32.MaxValue</code> , and <code>Int64.MinValue</code> is less than (has a greater negative range than) <code>Int32.MinValue</code> .
<code>UInt32</code>	<code>UInt32.MaxValue</code> is greater than <code>Int32.MaxValue</code> .
<code>UInt64</code>	<code>UInt64.MaxValue</code> is greater than <code>Int32.MaxValue</code> .

To handle such narrowing conversions, .NET allows types to define an `Explicit` operator. Individual language compilers can then implement this operator using their own syntax, or a member of the `Convert` class can be called to perform the conversion. (For more information about the `Convert` class, see [The Convert Class](#) later in this topic.) The following example illustrates the use of language features to handle the explicit conversion of these potentially out-of-range integer values to `Int32` values.

```
long number1 = int.MaxValue + 20L;
uint number2 = int.MaxValue - 1000;
ulong number3 = int.MaxValue;

int intNumber;

try
{
    intNumber = checked((int)number1);
    Console.WriteLine("After assigning a {0} value, the Integer value is {1}.",
                      number1.GetType().Name, intNumber);
}
catch (OverflowException)
{
    if (number1 > int.MaxValue)
        Console.WriteLine("Conversion failed: {0} exceeds {1}.",
                          number1, int.MaxValue);
    else
        Console.WriteLine("Conversion failed: {0} is less than {1}.",
                          number1, int.MinValue);
}

try
{
    intNumber = checked((int)number2);
    Console.WriteLine("After assigning a {0} value, the Integer value is {1}.",
                      number2.GetType().Name, intNumber);
}
catch (OverflowException)
{
    Console.WriteLine("Conversion failed: {0} exceeds {1}.",
                      number2, int.MaxValue);
}

try
{
    intNumber = checked((int)number3);
    Console.WriteLine("After assigning a {0} value, the Integer value is {1}.",
                      number3.GetType().Name, intNumber);
}
catch (OverflowException)
{
    Console.WriteLine("Conversion failed: {0} exceeds {1}.",
                      number1, int.MaxValue);

}

// The example displays the following output:
//      Conversion failed: 2147483667 exceeds 2147483647.
//      After assigning a UInt32 value, the Integer value is 2147482647.
//      After assigning a UInt64 value, the Integer value is 2147483647.
```

```

Dim number1 As Long = Integer.MaxValue + 20L
Dim number2 As UInteger = Integer.MaxValue - 1000
Dim number3 As ULong = Integer.MaxValue

Dim intNumber As Integer

Try
    intNumber = CInt(number1)
    Console.WriteLine("After assigning a {0} value, the Integer value is {1}.",
        number1.GetType().Name, intNumber)
Catch e As OverflowException
    If number1 > Integer.MaxValue Then
        Console.WriteLine("Conversion failed: {0} exceeds {1}.",
            number1, Integer.MaxValue)
    Else
        Console.WriteLine("Conversion failed: {0} is less than {1}.\\n",
            number1, Integer.MinValue)
    End If
End Try

Try
    intNumber = CInt(number2)
    Console.WriteLine("After assigning a {0} value, the Integer value is {1}.",
        number2.GetType().Name, intNumber)
Catch e As OverflowException
    Console.WriteLine("Conversion failed: {0} exceeds {1}.",
        number2, Integer.MaxValue)
End Try

Try
    intNumber = CInt(number3)
    Console.WriteLine("After assigning a {0} value, the Integer value is {1}.",
        number3.GetType().Name, intNumber)
Catch e As OverflowException
    Console.WriteLine("Conversion failed: {0} exceeds {1}.",
        number1, Integer.MaxValue)
End Try

' The example displays the following output:
'   Conversion failed: 2147483667 exceeds 2147483647.
'   After assigning a UInt32 value, the Integer value is 2147482647.
'   After assigning a UInt64 value, the Integer value is 2147483647.

```

Explicit conversions can produce different results in different languages, and these results can differ from the value returned by the corresponding [Convert](#) method. For example, if the [Double](#) value 12.63251 is converted to an [Int32](#), both the Visual Basic `cint` method and the .NET `Convert.ToInt32(Double)` method round the [Double](#) to return a value of 13, but the C# `(int)` operator truncates the [Double](#) to return a value of 12. Similarly, the C# `(int)` operator does not support Boolean-to-integer conversion, but the Visual Basic `cint` method converts a value of `true` to -1. On the other hand, the `Convert.ToInt32(Boolean)` method converts a value of `true` to 1.

Most compilers allow explicit conversions to be performed in a checked or unchecked manner. When a checked conversion is performed, an [OverflowException](#) is thrown when the value of the type to be converted is outside the range of the target type. When an unchecked conversion is performed under the same conditions, the conversion might not throw an exception, but the exact behavior becomes undefined and an incorrect value might result.

NOTE

In C#, checked conversions can be performed by using the `checked` keyword together with a casting operator, or by specifying the `/checked+` compiler option. Conversely, unchecked conversions can be performed by using the `unchecked` keyword together with the casting operator, or by specifying the `/checked-` compiler option. By default, explicit conversions are unchecked. In Visual Basic, checked conversions can be performed by clearing the **Remove integer overflow checks** check box in the project's **Advanced Compiler Settings** dialog box, or by specifying the `/removeintchecks-` compiler option. Conversely, unchecked conversions can be performed by selecting the **Remove integer overflow checks** check box in the project's **Advanced Compiler Settings** dialog box or by specifying the `/removeintchecks+` compiler option. By default, explicit conversions are checked.

The following C# example uses the `checked` and `unchecked` keywords to illustrate the difference in behavior when a value outside the range of a `Byte` is converted to a `Byte`. The checked conversion throws an exception, but the unchecked conversion assigns `Byte.MaxValue` to the `Byte` variable.

```
int largeValue = Int32.MaxValue;
byte newValue;

try
{
    newValue = unchecked((byte)largeValue);
    Console.WriteLine("Converted the {0} value {1} to the {2} value {3}.",
                      largeValue.GetType().Name, largeValue,
                      newValue.GetType().Name, newValue);
}
catch (OverflowException)
{
    Console.WriteLine("{0} is outside the range of the Byte data type.",
                      largeValue);
}

try
{
    newValue = checked((byte)largeValue);
    Console.WriteLine("Converted the {0} value {1} to the {2} value {3}.",
                      largeValue.GetType().Name, largeValue,
                      newValue.GetType().Name, newValue);
}
catch (OverflowException)
{
    Console.WriteLine("{0} is outside the range of the Byte data type.",
                      largeValue);
}

// The example displays the following output:
//     Converted the Int32 value 2147483647 to the Byte value 255.
//     2147483647 is outside the range of the Byte data type.
```

If a particular language compiler supports custom overloaded operators, you can also define explicit conversions in your own custom types. The following example provides a partial implementation of a signed byte data type named `ByteWithSign` that uses sign-and-magnitude representation. It supports explicit conversion of `Int32` and `UInt32` values to `ByteWithSign` values.

```
public struct ByteWithSignE
{
    private SByte signValue;
    private Byte value;

    private const byte.MaxValue = byte.MaxValue;
    private const int.MinValue = -1 * byte.MaxValue;

    public static explicit operator ByteWithSignE(int value)
    {
        // Check for overflow.
        if (value > ByteWithSignE.MaxValue || value < ByteWithSignE.MinValue)
            throw new OverflowException(String.Format("{0}' is out of range of the ByteWithSignE data
type.",

        ByteWithSignE newValue;
        newValue.signValue = (SByte)Math.Sign(value);
        newValue.value = (byte)Math.Abs(value);
        return newValue;
    }

    public static explicit operator ByteWithSignE(uint value)
    {
        if (value > ByteWithSignE.MaxValue)
            throw new OverflowException(String.Format("{0}' is out of range of the ByteWithSignE data
type.",

        ByteWithSignE newValue;
        newValue.signValue = 1;
        newValue.value = (byte)value;
        return newValue;
    }

    public override string ToString()
    {
        return (signValue * value).ToString();
    }
}
```

```

Public Structure ByteWithSign
    Private signValue As SByte
    Private value As Byte

    Private Const MaxValue As Byte = Byte.MaxValue
    Private Const MinValue As Integer = -1 * Byte.MaxValue

    Public Overloads Shared Narrowing Operator CType(value As Integer) As ByteWithSign
        ' Check for overflow.
        If value > ByteWithSign.MaxValue Or value < ByteWithSign.MinValue Then
            Throw New OverflowException(String.Format("{0}' is out of range of the ByteWithSign data type.", value))
        End If

        Dim newValue As ByteWithSign

        newValue.signValue = CSByte(Math.Sign(value))
        newValue.value = CByte(Math.Abs(value))
        Return newValue
    End Operator

    Public Overloads Shared Narrowing Operator CType(value As UInteger) As ByteWithSign
        If value > ByteWithSign.MaxValue Then
            Throw New OverflowException(String.Format("{0}' is out of range of the ByteWithSign data type.", value))
        End If

        Dim newValue As ByteWithSign

        newValue.signValue = 1
        newValue.value = CByte(value)
        Return newValue
    End Operator

    Public Overrides Function ToString() As String
        Return (signValue * value).ToString()
    End Function
End Structure

```

Client code can then declare a `ByteWithSign` variable and assign it `Int32` and `UInt32` values if the assignments include a casting operator or a conversion method, as the following example shows.

```

ByteWithSignE value;

try
{
    int intValue = -120;
    value = (ByteWithSignE)intValue;
    Console.WriteLine(value);
}
catch (OverflowException e)
{
    Console.WriteLine(e.Message);
}

try
{
    uint uintValue = 1024;
    value = (ByteWithSignE)uintValue;
    Console.WriteLine(value);
}
catch (OverflowException e)
{
    Console.WriteLine(e.Message);
}

// The example displays the following output:
//      -120
//      '1024' is out of range of the ByteWithSignE data type.

```

```

Dim value As ByteWithSign

Try
    Dim intValue As Integer = -120
    value = CType(intValue, ByteWithSign)
    Console.WriteLine(value)
Catch e As OverflowException
    Console.WriteLine(e.Message)
End Try

Try
    Dim uintValue As UInteger = 1024
    value = CType(uintValue, ByteWithSign)
    Console.WriteLine(value)
Catch e As OverflowException
    Console.WriteLine(e.Message)
End Try

' The example displays the following output:
'      -120
'      '1024' is out of range of the ByteWithSign data type.

```

The IConvertible interface

To support the conversion of any type to a common language runtime base type, .NET provides the [IConvertible](#) interface. The implementing type is required to provide the following:

- A method that returns the [TypeCode](#) of the implementing type.
- Methods to convert the implementing type to each common language runtime base type ([Boolean](#), [Byte](#), [DateTime](#), [Decimal](#), [Double](#), and so on).
- A generalized conversion method to convert an instance of the implementing type to another specified type. Conversions that are not supported should throw an [InvalidCastException](#).

Each common language runtime base type (that is, the [Boolean](#), [Byte](#), [Char](#), [DateTime](#), [Decimal](#), [Double](#), [Int16](#),

`Int32`, `Int64`, `SByte`, `Single`, `String`, `UInt16`, `UInt32`, and `UInt64`, as well as the `DBNull` and `Enum` types, implement the `IConvertible` interface. However, these are explicit interface implementations; the conversion method can be called only through an `IConvertible` interface variable, as the following example shows. This example converts an `Int32` value to its equivalent `Char` value.

```
int codePoint = 1067;
IConvertible iConv = codePoint;
char ch = iConv.ToChar(null);
Console.WriteLine("Converted {0} to {1}.", codePoint, ch);
```

```
Dim codePoint As Integer = 1067
Dim iConv As IConvertible = codePoint
Dim ch As Char = iConv.ToChar(Nothing)
Console.WriteLine("Converted {0} to {1}.", codePoint, ch)
```

The requirement to call the conversion method on its interface rather than on the implementing type makes explicit interface implementations relatively expensive. Instead, we recommend that you call the appropriate member of the `Convert` class to convert between common language runtime base types. For more information, see the next section, [The Convert Class](#).

NOTE

In addition to the `IConvertible` interface and the `Convert` class provided by .NET, individual languages may also provide ways to perform conversions. For example, C# uses casting operators; Visual Basic uses compiler-implemented conversion functions such as `CType`, `CInt`, and `DirectCast`.

For the most part, the `IConvertible` interface is designed to support conversion between the base types in .NET. However, the interface can also be implemented by a custom type to support conversion of that type to other custom types. For more information, see the section [Custom Conversions with the ChangeType Method](#) later in this topic.

The Convert class

Although each base type's `IConvertible` interface implementation can be called to perform a type conversion, calling the methods of the `System.Convert` class is the recommended language-neutral way to convert from one base type to another. In addition, the `Convert.ChangeType(Object, Type, IFormatProvider)` method can be used to convert from a specified custom type to another type.

Conversions between base types

The `Convert` class provides a language-neutral way to perform conversions between base types and is available to all languages that target the common language runtime. It provides a complete set of methods for both widening and narrowing conversions, and throws an `InvalidOperationException` for conversions that are not supported (such as the conversion of a `DateTime` value to an integer value). Narrowing conversions are performed in a checked context, and an `OverflowException` is thrown if the conversion fails.

IMPORTANT

Because the `Convert` class includes methods to convert to and from each base type, it eliminates the need to call each base type's `IConvertible` explicit interface implementation.

The following example illustrates the use of the `System.Convert` class to perform several widening and narrowing conversions between .NET base types.

```

// Convert an Int32 value to a Decimal (a widening conversion).
int intValue = 12534;
decimal decimalValue = Convert.ToDecimal(intValue);
Console.WriteLine("Converted the {0} value {1} to " +
    "the {2} value {3:N2}.",
    intValue.GetType().Name,
    intValue,
    decimalValue.GetType().Name,
    decimalValue);

// Convert a Byte value to an Int32 value (a widening conversion).
byte byteValue = Byte.MaxValue;
int intValue2 = Convert.ToInt32(byteValue);
Console.WriteLine("Converted the {0} value {1} to " +
    "the {2} value {3:G}.",
    byteValue.GetType().Name,
    byteValue,
    intValue2.GetType().Name,
    intValue2);

// Convert a Double value to an Int32 value (a narrowing conversion).
double doubleValue = 16.32513e12;
try
{
    long longValue = Convert.ToInt64(doubleValue);
    Console.WriteLine("Converted the {0} value {1:E} to " +
        "the {2} value {3:N0}.",
        doubleValue.GetType().Name,
        doubleValue,
        longValue.GetType().Name,
        longValue);
}
catch (OverflowException)
{
    Console.WriteLine("Unable to convert the {0:E} value {1}.",
        doubleValue.GetType().Name, doubleValue);
}

// Convert a signed byte to a byte (a narrowing conversion).
sbyte sbyteValue = -16;
try
{
    byte byteValue2 = Convert.ToByte(sbyteValue);
    Console.WriteLine("Converted the {0} value {1} to " +
        "the {2} value {3:G}.",
        sbyteValue.GetType().Name,
        sbyteValue,
        byteValue2.GetType().Name,
        byteValue2);
}
catch (OverflowException)
{
    Console.WriteLine("Unable to convert the {0} value {1}.",
        sbyteValue.GetType().Name, sbyteValue);
}

// The example displays the following output:
//      Converted the Int32 value 12534 to the Decimal value 12,534.00.
//      Converted the Byte value 255 to the Int32 value 255.
//      Converted the Double value 1.632513E+013 to the Int64 value 16,325,130,000,000.
//      Unable to convert the SByte value -16.

```

```

' Convert an Int32 value to a Decimal (a widening conversion).
Dim intValue As Integer = 12534
Dim decimalValue As Decimal = Convert.ToDecimal(intintValue)
Console.WriteLine("Converted the {0} value {1} to the {2} value {3:N2}.",
    intValue.GetType().Name,
    intValue,
    decimalValue.GetType().Name,
    decimalValue)

' Convert a Byte value to an Int32 value (a widening conversion).
Dim byteValue As Byte = Byte.MaxValue
Dim intValue2 As Integer = Convert.ToInt32(byteValue)
Console.WriteLine("Converted the {0} value {1} to " +
    "the {2} value {3:G}.",
    byteValue.GetType().Name,
    byteValue,
    intValue2.GetType().Name,
    intValue2)

' Convert a Double value to an Int32 value (a narrowing conversion).
Dim doubleValue As Double = 16.32513e12
Try
    Dim longValue As Long = Convert.ToInt64(doubleValue)
    Console.WriteLine("Converted the {0} value {1:E} to " +
        "the {2} value {3:N0}.",
        doubleValue.GetType().Name,
        doubleValue,
        longValue.GetType().Name,
        longValue)
Catch e As OverflowException
    Console.WriteLine("Unable to convert the {0:E} value {1}.",
        doubleValue.GetType().Name, doubleValue)
End Try

' Convert a signed byte to a byte (a narrowing conversion).
Dim sbyteValue As SByte = -16
Try
    Dim byteValue2 As Byte = Convert.ToByte(sbyteValue)
    Console.WriteLine("Converted the {0} value {1} to " +
        "the {2} value {3:G}.",
        sbyteValue.GetType().Name,
        sbyteValue,
        byteValue2.GetType().Name,
        byteValue2)
Catch e As OverflowException
    Console.WriteLine("Unable to convert the {0} value {1}.",
        sbyteValue.GetType().Name, sbyteValue)
End Try
' The example displays the following output:
'     Converted the Int32 value 12534 to the Decimal value 12,534.00.
'     Converted the Byte value 255 to the Int32 value 255.
'     Converted the Double value 1.632513E+013 to the Int64 value 16,325,130,000,000.
'     Unable to convert the SByte value -16.

```

In some cases, particularly when converting to and from floating-point values, a conversion may involve a loss of precision, even though it does not throw an [OverflowException](#). The following example illustrates this loss of precision. In the first case, a [Decimal](#) value has less precision (fewer significant digits) when it is converted to a [Double](#). In the second case, a [Double](#) value is rounded from 42.72 to 43 in order to complete the conversion.

```

double doubleValue;

// Convert a Double to a Decimal.
decimal decimalValue = 13956810.96702888123451471211m;
doubleValue = Convert.ToDouble(decimalValue);
Console.WriteLine("{0} converted to {1}.", decimalValue, doubleValue);

doubleValue = 42.72;
try
{
    int integerValue = Convert.ToInt32(doubleValue);
    Console.WriteLine("{0} converted to {1}.",
                      doubleValue, integerValue);
}
catch (OverflowException)
{
    Console.WriteLine("Unable to convert {0} to an integer.",
                      doubleValue);
}
// The example displays the following output:
//      13956810.96702888123451471211 converted to 13956810.9670289.
//      42.72 converted to 43.

```

```

Dim doubleValue As Double

' Convert a Double to a Decimal.
Dim decimalValue As Decimal = 13956810.96702888123451471211d
doubleValue = Convert.ToDouble(decimalValue)
Console.WriteLine("{0} converted to {1}.", decimalValue, doubleValue)

doubleValue = 42.72
Try
    Dim integerValue As Integer = Convert.ToInt32(doubleValue)
    Console.WriteLine("{0} converted to {1}.",
                      doubleValue, integerValue)
Catch e As OverflowException
    Console.WriteLine("Unable to convert {0} to an integer.",
                      doubleValue)
End Try
' The example displays the following output:
'      13956810.96702888123451471211 converted to 13956810.9670289.
'      42.72 converted to 43.

```

For a table that lists both the widening and narrowing conversions supported by the [Convert](#) class, see [Type Conversion Tables](#).

Custom conversions with the `ChangeType` method

In addition to supporting conversions to each of the base types, the [Convert](#) class can be used to convert a custom type to one or more predefined types. This conversion is performed by the [Convert.ChangeType\(Object, Type\)](#), [Convert.ChangeType\(Object, Type, IFormatProvider\)](#) method, which in turn wraps a call to the [IConvertible.ToType](#) method of the `value` parameter. This means that the object represented by the `value` parameter must provide an implementation of the [IConvertible](#) interface.

NOTE

Because the [Convert.ChangeType\(Object, Type\)](#) and [Convert.ChangeType\(Object, Type, IFormatProvider\)](#) methods use a `Type` object to specify the target type to which `value` is converted, they can be used to perform a dynamic conversion to an object whose type is not known at compile time. However, note that the [IConvertible](#) implementation of `value` must still support this conversion.

The following example illustrates a possible implementation of the [IConvertible](#) interface that allows a `TemperatureCelsius` object to be converted to a `TemperatureFahrenheit` object and vice versa. The example defines a base class, `Temperature`, that implements the [IConvertible](#) interface and overrides the `Object.ToString` method. The derived `TemperatureCelsius` and `TemperatureFahrenheit` classes each override the `ToType` and the `ToString` methods of the base class.

```
using System;

public abstract class Temperature : IConvertible
{
    protected decimal temp;

    public Temperature(decimal temperature)
    {
        this.temp = temperature;
    }

    public decimal Value
    {
        get { return this.temp; }
        set { this.temp = value; }
    }

    public override string ToString()
    {
        return temp.ToString(null as IFormatProvider) + "°";
    }

    // IConvertible implementations.
    public TypeCode GetTypeCode()
    {
        return TypeCode.Object;
    }

    public bool ToBoolean(IFormatProvider provider)
    {
        throw new InvalidCastException(String.Format("Temperature-to-Boolean conversion is not supported."));
    }

    public byte ToByte(IFormatProvider provider)
    {
        if (temp < Byte.MinValue || temp > Byte.MaxValue)
            throw new OverflowException(String.Format("{0} is out of range of the Byte data type.", temp));
        else
            return (byte)temp;
    }

    public char ToChar(IFormatProvider provider)
    {
        throw new InvalidCastException("Temperature-to-Char conversion is not supported.");
    }

    public DateTime ToDateTime(IFormatProvider provider)
    {
        throw new InvalidCastException("Temperature-to-DateTime conversion is not supported.");
    }

    public decimal ToDecimal(IFormatProvider provider)
    {
        return temp;
    }

    public double ToDouble(IFormatProvider provider)
    {
        return (double)temp;
    }
}
```

```
}

public shortToInt16(IFormatProvider provider)
{
    if (temp < Int16.MinValue || temp > Int16.MaxValue)
        throw new OverflowException(String.Format("{0} is out of range of the Int16 data type.", temp));
    else
        return (short)Math.Round(temp);
}

public intToInt32(IFormatProvider provider)
{
    if (temp < Int32.MinValue || temp > Int32.MaxValue)
        throw new OverflowException(String.Format("{0} is out of range of the Int32 data type.", temp));
    else
        return (int)Math.Round(temp);
}

public longToInt64(IFormatProvider provider)
{
    if (temp < Int64.MinValue || temp > Int64.MaxValue)
        throw new OverflowException(String.Format("{0} is out of range of the Int64 data type.", temp));
    else
        return (long)Math.Round(temp);
}

public sbyteToSByte(IFormatProvider provider)
{
    if (temp < SByte.MinValue || temp > SByte.MaxValue)
        throw new OverflowException(String.Format("{0} is out of range of the SByte data type.", temp));
    else
        return (sbyte)temp;
}

public floatToSingle(IFormatProvider provider)
{
    return (float)temp;
}

public virtual stringToString(IFormatProvider provider)
{
    return temp.ToString(provider) + "°";
}

// If conversionType is implemented by another IConvertible method, call it.
public virtual objectToType(Type conversionType, IFormatProvider provider)
{
    switch (Type.GetTypeCode(conversionType))
    {
        case TypeCode.Boolean:
            return this.ToBoolean(provider);
        case TypeCode.Byte:
            return this.ToByte(provider);
        case TypeCode.Char:
            return this.ToChar(provider);
        case TypeCode.DateTime:
            return this.ToDateTime(provider);
        case TypeCode.Decimal:
            return this.ToDecimal(provider);
        case TypeCode.Double:
            return this.ToDouble(provider);
        case TypeCode.Empty:
            throw new NullReferenceException("The target type is null.");
        case TypeCode.Int16:
            return this.ToInt16(provider);
        case TypeCode.Int32:
            return this.ToInt32(provider);
        case TypeCode.Int64:
            return this.ToInt64(provider);
```

```

        case TypeCode.Object:
            // Leave conversion of non-base types to derived classes.
            throw new InvalidCastException(String.Format("Cannot convert from Temperature to {0}.",
                conversionType.Name));
        case TypeCode.SByte:
            return this.ToSByte(provider);
        case TypeCode.Single:
            return this.ToSingle(provider);
        case TypeCode.String:
            IConvertible iconv = this;
            return iconv.ToString(provider);
        case TypeCode.UInt16:
            return this.ToInt16(provider);
        case TypeCode.UInt32:
            return this.ToInt32(provider);
        case TypeCode.UInt64:
            return this.ToInt64(provider);
        default:
            throw new InvalidCastException("Conversion not supported.");
    }
}

public ushort ToUInt16(IFormatProvider provider)
{
    if (temp < UInt16.MinValue || temp > UInt16.MaxValue)
        throw new OverflowException(String.Format("{0} is out of range of the UInt16 data type.", temp));
    else
        return (ushort)Math.Round(temp);
}

public uint ToUInt32(IFormatProvider provider)
{
    if (temp < UInt32.MinValue || temp > UInt32.MaxValue)
        throw new OverflowException(String.Format("{0} is out of range of the UInt32 data type.", temp));
    else
        return (uint)Math.Round(temp);
}

public ulong ToUInt64(IFormatProvider provider)
{
    if (temp < UInt64.MinValue || temp > UInt64.MaxValue)
        throw new OverflowException(String.Format("{0} is out of range of the UInt64 data type.", temp));
    else
        return (ulong)Math.Round(temp);
}

public class TemperatureCelsius : Temperature, IConvertible
{
    public TemperatureCelsius(decimal value) : base(value)
    {
    }

    // Override ToString methods.
    public override string ToString()
    {
        return this.ToString(null);
    }

    public override string ToString(IFormatProvider provider)
    {
        return temp.ToString(provider) + "°C";
    }

    // If conversionType is a implemented by another IConvertible method, call it.
    public override object ToType(Type conversionType, IFormatProvider provider)

```

```

    {
        // For non-objects, call base method.
        if (Type.GetTypeCode(conversionType) != TypeCode.Object)
        {
            return base.ToType(conversionType, provider);
        }
        else
        {
            if (conversionType.Equals(typeof(TemperatureCelsius)))
                return this;
            else if (conversionType.Equals(typeof(TemperatureFahrenheit)))
                return new TemperatureFahrenheit((decimal)this.temp * 9 / 5 + 32);
            else
                throw new InvalidCastException(String.Format("Cannot convert from Temperature to {0}.",
                                                               conversionType.Name));
        }
    }
}

public class TemperatureFahrenheit : Temperature, IConvertible
{
    public TemperatureFahrenheit(decimal value) : base(value)
    {
    }

    // Override ToString methods.
    public override string ToString()
    {
        return this.ToString(null);
    }

    public override string ToString(IFormatProvider provider)
    {
        return temp.ToString(provider) + "°F";
    }

    public override object ToType(Type conversionType, IFormatProvider provider)
    {
        // For non-objects, call base method.
        if (Type.GetTypeCode(conversionType) != TypeCode.Object)
        {
            return base.ToType(conversionType, provider);
        }
        else
        {
            // Handle conversion between derived classes.
            if (conversionType.Equals(typeof(TemperatureFahrenheit)))
                return this;
            else if (conversionType.Equals(typeof(TemperatureCelsius)))
                return new TemperatureCelsius((decimal)(this.temp - 32) * 5 / 9);
            // Unspecified object type: throw an InvalidCastException.
            else
                throw new InvalidCastException(String.Format("Cannot convert from Temperature to {0}.",
                                                               conversionType.Name));
        }
    }
}

```

```

Public MustInherit Class Temperature
    Implements IConvertible

    Protected temp As Decimal

    Public Sub New(temperature As Decimal)
        Me.temp = temperature
    End Sub

```

```

Public Property Value As Decimal
    Get
        Return Me.temp
    End Get
    Set
        Me.temp = Value
    End Set
End Property

Public Overrides Function ToString() As String
    Return temp.ToString() & "°"
End Function

' IConvertible implementations.
Public Function GetTypeCode() As TypeCode Implements IConvertible.GetTypeCode
    Return TypeCode.Object
End Function

Public Function ToBoolean(provider As IFormatProvider) As Boolean Implements IConvertible.ToBoolean
    Throw New InvalidCastException(String.Format("Temperature-to-Boolean conversion is not supported."))
End Function

Public Function ToByte(provider As IFormatProvider) As Byte Implements IConvertible.ToByte
    If temp < Byte.MinValue Or temp > Byte.MaxValue Then
        Throw New OverflowException(String.Format("{0} is out of range of the Byte data type.", temp))
    Else
        Return CByte(temp)
    End If
End Function

Public Function ToChar(provider As IFormatProvider) As Char Implements IConvertible.ToChar
    Throw New InvalidCastException("Temperature-to-Char conversion is not supported.")
End Function

Public Function ToDateTime(provider As IFormatProvider) As DateTime Implements IConvertible.ToDateTime
    Throw New InvalidCastException("Temperature-to-DateTime conversion is not supported.")
End Function

Public Function ToDecimal(provider As IFormatProvider) As Decimal Implements IConvertible.ToDecimal
    Return temp
End Function

Public Function ToDouble(provider As IFormatProvider) As Double Implements IConvertible.ToDouble
    Return CDbl(temp)
End Function

Public FunctionToInt16(provider As IFormatProvider) As Int16 Implements IConvertible.ToInt16
    If temp < Int16.MinValue Or temp > Int16.MaxValue Then
        Throw New OverflowException(String.Format("{0} is out of range of the Int16 data type.", temp))
    End If
    Return CShort(Math.Round(temp))
End Function

Public FunctionToInt32(provider As IFormatProvider) As Int32 Implements IConvertible.ToInt32
    If temp < Int32.MinValue Or temp > Int32.MaxValue Then
        Throw New OverflowException(String.Format("{0} is out of range of the Int32 data type.", temp))
    End If
    Return CInt(Math.Round(temp))
End Function

Public FunctionToInt64(provider As IFormatProvider) As Int64 Implements IConvertible.ToInt64
    If temp < Int64.MinValue Or temp > Int64.MaxValue Then
        Throw New OverflowException(String.Format("{0} is out of range of the Int64 data type.", temp))
    End If
    Return CLng(Math.Round(temp))
End Function

Public Function ToSByte(provider As IFormatProvider) As SByte Implements IConvertible.ToSByte
    If temp < SByte.MinValue Or temp > SByte.MaxValue Then

```

```

        Throw New OverflowException(String.Format("{0} is out of range of the SByte data type.", temp))
    Else
        Return CSByte(temp)
    End If
End Function

Public Function ToSingle(provider As IFormatProvider) As Single Implements IConvertible.ToSingle
    Return CSng(temp)
End Function

Public Overridable Overloads Function ToString(provider As IFormatProvider) As String Implements IConvertible.ToString
    Return temp.ToString(provider) & " °C"
End Function

' If conversionType is a implemented by another IConvertible method, call it.
Public Overridable Function ToType(conversionType As Type, provider As IFormatProvider) As Object
Implements IConvertible.ToType
    Select Case Type.GetTypeCode(conversionType)
        Case TypeCode.Boolean
            Return Me.ToBoolean(provider)
        Case TypeCode.Byte
            Return Me.ToByte(provider)
        Case TypeCode.Char
            Return Me.ToChar(provider)
        Case TypeCode.DateTime
            Return Me.ToDateTime(provider)
        Case TypeCode.Decimal
            Return Me.ToDecimal(provider)
        Case TypeCode.Double
            Return Me.ToDouble(provider)
        Case TypeCode.Empty
            Throw New NullReferenceException("The target type is null.")
        Case TypeCode.Int16
            Return Me.ToInt16(provider)
        Case TypeCode.Int32
            Return Me.ToInt32(provider)
        Case TypeCode.Int64
            Return Me.ToInt64(provider)
        Case TypeCode.Object
            ' Leave conversion of non-base types to derived classes.
            Throw New InvalidCastException(String.Format("Cannot convert from Temperature to {0}.", _
                conversionType.Name))
        Case TypeCode.SByte
            Return Me.ToSByte(provider)
        Case TypeCode.Single
            Return Me.ToSingle(provider)
        Case TypeCode.String
            Return Me.ToString(provider)
        Case TypeCode.UInt16
            Return Me.ToUInt16(provider)
        Case TypeCode.UInt32
            Return Me.ToUInt32(provider)
        Case TypeCode.UInt64
            Return Me.ToUInt64(provider)
        Case Else
            Throw New InvalidCastException("Conversion not supported.")
    End Select
End Function

Public Function ToUInt16(provider As IFormatProvider) As UInt16 Implements IConvertible.ToUInt16
    If temp < UInt16.MinValue Or temp > UInt16.MaxValue Then
        Throw New OverflowException(String.Format("{0} is out of range of the UInt16 data type.", temp))
    End If
    Return CUShort(Math.Round(temp))
End Function

Public Function ToUInt32(provider As IFormatProvider) As UInt32 Implements IConvertible.ToUInt32
    If temp < UInt32.MinValue Or temp > UInt32.MaxValue Then

```

```

        Throw New OverflowException(String.Format("{0} is out of range of the UInt32 data type.", temp))
    End If
    Return CUInt(Math.Round(temp))
End Function

Public Function ToUInt64(provider As IFormatProvider) As UInt64 Implements IConvertible.ToUInt64
    If temp < UInt64.MinValue Or temp > UInt64.MaxValue Then
        Throw New OverflowException(String.Format("{0} is out of range of the UInt64 data type.", temp))
    End If
    Return CULng(Math.Round(temp))
End Function
End Class

Public Class TemperatureCelsius : Inherits Temperature : Implements IConvertible
    Public Sub New(value As Decimal)
        MyBase.New(value)
    End Sub

    ' Override ToString methods.
    Public Overrides Function ToString() As String
        Return Me.ToString(Nothing)
    End Function

    Public Overrides Function ToString(provider As IFormatProvider) As String
        Return temp.ToString(provider) + "°C"
    End Function

    ' If conversionType is implemented by another IConvertible method, call it.
    Public Overrides Function ToType(conversionType As Type, provider As IFormatProvider) As Object
        ' For non-objects, call base method.
        If Type.GetTypeCode(conversionType) <> TypeCode.Object Then
            Return MyBase.ToType(conversionType, provider)
        Else
            If conversionType.Equals(GetType(TemperatureCelsius)) Then
                Return Me
            ElseIf conversionType.Equals(GetType(TemperatureFahrenheit))
                Return New TemperatureFahrenheit(CDec(Me.temp * 9 / 5 + 32))
            ' Unspecified object type: throw an InvalidCastException.
            Else
                Throw New InvalidCastException(String.Format("Cannot convert from Temperature to {0}.", _
                    conversionType.Name))
            End If
        End If
    End Function
End Class

Public Class TemperatureFahrenheit : Inherits Temperature : Implements IConvertible
    Public Sub New(value As Decimal)
        MyBase.New(value)
    End Sub

    ' Override ToString methods.
    Public Overrides Function ToString() As String
        Return Me.ToString(Nothing)
    End Function

    Public Overrides Function ToString(provider As IFormatProvider) As String
        Return temp.ToString(provider) + "°F"
    End Function

    Public Overrides Function ToType(conversionType As Type, provider As IFormatProvider) As Object
        ' For non-objects, call base method.
        If Type.GetTypeCode(conversionType) <> TypeCode.Object Then
            Return MyBase.ToType(conversionType, provider)
        Else
            ' Handle conversion between derived classes.
            If conversionType.Equals(GetType(TemperatureFahrenheit)) Then
                Return Me
            ElseIf conversionType.Equals(GetType(TemperatureCelsius))

```

```

        ElseIf conversionType.Equals(DataType.TemperatureCelsius))
            Return New TemperatureCelsius(CDec((MyBase.temp - 32) * 5 / 9))
        ' Unspecified object type: throw an InvalidCastException.
    Else
        Throw New InvalidCastException(String.Format("Cannot convert from {0} to {1}.", _
                                                    conversionType.Name))
    End If
End If
End Function
End Class

```

The following example illustrates several calls to these [IConvertible](#) implementations to convert `TemperatureCelsius` objects to `TemperatureFahrenheit` objects and vice versa.

```

TemperatureCelsius tempC1 = new TemperatureCelsius(0);
TemperatureFahrenheit tempF1 = (TemperatureFahrenheit)Convert.ChangeType(tempC1,
typeof(TemperatureFahrenheit), null);
Console.WriteLine("{0} equals {1}.", tempC1, tempF1);
TemperatureCelsius tempC2 = (TemperatureCelsius)Convert.ChangeType(tempC1, typeof(TemperatureCelsius),
null);
Console.WriteLine("{0} equals {1}.", tempC1, tempC2);
TemperatureFahrenheit tempF2 = new TemperatureFahrenheit(212);
TemperatureCelsius tempC3 = (TemperatureCelsius)Convert.ChangeType(tempF2, typeof(TemperatureCelsius),
null);
Console.WriteLine("{0} equals {1}.", tempF2, tempC3);
TemperatureFahrenheit tempF3 = (TemperatureFahrenheit)Convert.ChangeType(tempF2,
typeof(TemperatureFahrenheit), null);
Console.WriteLine("{0} equals {1}.", tempF2, tempF3);
// The example displays the following output:
//      0°C equals 32°F.
//      0°C equals 0°C.
//      212°F equals 100°C.
//      212°F equals 212°F.

```

```

Dim tempC1 As New TemperatureCelsius(0)
Dim tempF1 As TemperatureFahrenheit = CType(Convert.ChangeType(tempC1, GetType(TemperatureFahrenheit),
Nothing), TemperatureFahrenheit)
Console.WriteLine("{0} equals {1}.", tempC1, tempF1)
Dim tempC2 As TemperatureCelsius = CType(Convert.ChangeType(tempC1, GetType(TemperatureCelsius), Nothing),
TemperatureCelsius)
Console.WriteLine("{0} equals {1}.", tempC1, tempC2)
Dim tempF2 As New TemperatureFahrenheit(212)
Dim tempC3 As TemperatureCelsius = CType(Convert.ChangeType(tempF2, GetType(TemperatureCelsius), Nothing),
TemperatureCelsius)
Console.WriteLine("{0} equals {1}.", tempF2, tempC3)
Dim tempF3 As TemperatureFahrenheit = CType(Convert.ChangeType(tempF2, GetType(TemperatureFahrenheit),
Nothing), TemperatureFahrenheit)
Console.WriteLine("{0} equals {1}.", tempF2, tempF3)
' The example displays the following output:
'      0°C equals 32°F.
'      0°C equals 0°C.
'      212°F equals 100°C.
'      212°F equals 212°F.

```

The TypeConverter class

.NET also allows you to define a type converter for a custom type by extending the [System.ComponentModel.TypeConverter](#) class and associating the type converter with the type through a [System.ComponentModel.TypeConverterAttribute](#) attribute. The following table highlights the differences between this approach and implementing the [IConvertible](#) interface for a custom type.

NOTE

Design-time support can be provided for a custom type only if it has a type converter defined for it.

CONVERSION USING TYPECONVERTER	CONVERSION USING IConvertible
Is implemented for a custom type by deriving a separate class from TypeConverter . This derived class is associated with the custom type by applying a TypeConverterAttribute attribute.	Is implemented by a custom type to perform conversion. A user of the type invokes an IConvertible conversion method on the type.
Can be used both at design time and at run time.	Can be used only at run time.
Uses reflection; therefore, is slower than conversion enabled by IConvertible .	Does not use reflection.
Allows two-way type conversions from the custom type to other data types, and from other data types to the custom type. For example, a TypeConverter defined for <code>MyType</code> allows conversions from <code>MyType</code> to <code>String</code> , and from <code>String</code> to <code>MyType</code> .	Allows conversion from a custom type to other data types, but not from other data types to the custom type.

For more information about using type converters to perform conversions, see [System.ComponentModel.TypeConverter](#).

See also

- [System.Convert](#)
- [IConvertible](#)
- [Type Conversion Tables](#)

Type Conversion Tables in .NET

9/20/2022 • 2 minutes to read • [Edit Online](#)

Widening conversion occurs when a value of one type is converted to another type that is of equal or greater size. A narrowing conversion occurs when a value of one type is converted to a value of another type that is of a smaller size. The tables in this topic illustrate the behaviors exhibited by both types of conversions.

Widening Conversions

The following table describes the widening conversions that can be performed without the loss of information.

TYPE	CAN BE CONVERTED WITHOUT DATA LOSS TO
Byte	UInt16, Int16, UInt32, Int32, UInt64, Int64, Single, Double, Decimal
SByte	Int16, Int32, Int64, Single, Double, Decimal
Int16	Int32, Int64, Single, Double, Decimal
UInt16	UInt32, Int32, UInt64, Int64, Single, Double, Decimal
Char	UInt16, UInt32, Int32, UInt64, Int64, Single, Double, Decimal
Int32	Int64, Double, Decimal
UInt32	Int64, UInt64, Double, Decimal
Int64	Decimal
UInt64	Decimal
Single	Double

Some widening conversions to [Single](#) or [Double](#) can cause a loss of precision. The following table describes the widening conversions that sometimes result in a loss of information.

TYPE	CAN BE CONVERTED TO
Int32	Single
UInt32	Single
Int64	Single, Double
UInt64	Single, Double
Decimal	Single, Double

Narrowing Conversions

A narrowing conversion to [Single](#) or [Double](#) can cause a loss of information. If the target type cannot properly express the magnitude of the source, the resulting type is set to the constant `PositiveInfinity` or `NegativeInfinity`. `PositiveInfinity` results from dividing a positive number by zero and is also returned when the value of a [Single](#) or [Double](#) exceeds the value of the `.MaxValue` field. `NegativeInfinity` results from dividing a negative number by zero and is also returned when the value of a [Single](#) or [Double](#) falls below the value of the `.MinValue` field. A conversion from a [Double](#) to a [Single](#) might result in `PositiveInfinity` or `NegativeInfinity`.

A narrowing conversion can also result in a loss of information for other data types. However, an [OverflowException](#) is thrown if the value of a type that is being converted falls outside of the range specified by the target type's `.MaxValue` and `.MinValue` fields, and the conversion is checked by the runtime to ensure that the value of the target type does not exceed its `.MaxValue` or `.MinValue`. Conversions that are performed with the [System.Convert](#) class are always checked in this manner.

The following table lists conversions that throw an [OverflowException](#) using [System.Convert](#) or any checked conversion if the value of the type being converted is outside the defined range of the resulting type.

TYPE	CAN BE CONVERTED TO
Byte	SByte
SByte	Byte , UInt16 , UInt32 , UInt64
Int16	Byte , SByte , UInt16
UInt16	Byte , SByte , Int16
Int32	Byte , SByte , Int16 , UInt16 , UInt32
UInt32	Byte , SByte , Int16 , UInt16 , Int32
Int64	Byte , SByte , Int16 , UInt16 , Int32 , UInt32 , UInt64
UInt64	Byte , SByte , Int16 , UInt16 , Int32 , UInt32 , Int64
Decimal	Byte , SByte , Int16 , UInt16 , Int32 , UInt32 , Int64 , UInt64
Single	Byte , SByte , Int16 , UInt16 , Int32 , UInt32 , Int64 , UInt64
Double	Byte , SByte , Int16 , UInt16 , Int32 , UInt32 , Int64 , UInt64

See also

- [System.Convert](#)
- [Type Conversion in .NET](#)

Choosing between anonymous and tuple types

9/20/2022 • 3 minutes to read • [Edit Online](#)

Choosing the appropriate type involves considering its usability, performance, and tradeoffs compared to other types. Anonymous types have been available since C# 3.0, while generic `System.Tuple<T1,T2>` types were introduced with .NET Framework 4.0. Since then new options have been introduced with language level support, such as `System.ValueTuple<T1,T2>` - which as the name implies, provide a value type with the flexibility of anonymous types. In this article, you'll learn when it's appropriate to choose one type over the other.

Usability and functionality

Anonymous types were introduced in C# 3.0 with Language-Integrated Query (LINQ) expressions. With LINQ, developers often project results from queries into anonymous types that hold a few select properties from the objects they're working with. Consider the following example, that instantiates an array of `DateTime` objects, and iterates through them projecting into an anonymous type with two properties.

```
var dates = new[]
{
    DateTime.UtcNow.AddHours(-1),
    DateTime.UtcNow,
    DateTime.UtcNow.AddHours(1),
};

foreach (var anonymous in
    dates.Select(
        date => new { Formatted = $"{date:MMM dd, yyyy hh:mm zzz}", date.Ticks }))
{
    Console.WriteLine($"Ticks: {anonymous.Ticks}, formatted: {anonymous.Formatted}");
}
```

Anonymous types are instantiated by using the `new` operator, and the property names and types are inferred from the declaration. If two or more anonymous object initializers in the same assembly specify a sequence of properties that are in the same order and that have the same names and types, the compiler treats the objects as instances of the same type. They share the same compiler-generated type information.

The previous C# snippet projects an anonymous type with two properties, much like the following compiler-generated C# class:

```
internal sealed class f__AnonymousType0
{
    public string Formatted { get; }
    public long Ticks { get; }

    public f__AnonymousType0(string formatted, long ticks)
    {
        Formatted = formatted;
        Ticks = ticks;
    }
}
```

For more information, see [anonymous types](#). The same functionality exists with tuples when projecting into LINQ queries, you can select properties into tuples. These tuples flow through the query, just as anonymous types would. Now consider the following example using the `System.Tuple<string, long>`.

```

var dates = new[]
{
    DateTime.UtcNow.AddHours(-1),
    DateTime.UtcNow,
    DateTime.UtcNow.AddHours(1),
};

foreach (var tuple in
    dates.Select(
        date => new Tuple<string, long>($"date:{date:MMM dd, yyyy hh:mm zzz}", date.Ticks)))
{
    Console.WriteLine($"Ticks: {tuple.Item2}, formatted: {tuple.Item1}");
}

```

With the [System.Tuple<T1,T2>](#), the instance exposes numbered item properties, such as `Item1` and `Item2`.

These property names can make it more difficult to understand the intent of the property values, as the property name only provides the ordinal. Furthermore, the [System.Tuple](#) types are reference [class](#) types. The [System.ValueTuple<T1,T2>](#) however, is a value [struct](#) type. The following C# snippet, uses

`ValueTuple<string, long>` to project into. In doing so, it assigns using a literal syntax.

```

var dates = new[]
{
    DateTime.UtcNow.AddHours(-1),
    DateTime.UtcNow,
    DateTime.UtcNow.AddHours(1),
};

foreach (var (formatted, ticks) in
    dates.Select(
        date => (Formatted: $"date:{date:MMM dd, yyyy at hh:mm zzz}", date.Ticks)))
{
    Console.WriteLine($"Ticks: {ticks}, formatted: {formatted}");
}

```

For more information about tuples, see [Tuple types \(C# reference\)](#) or [Tuples \(Visual Basic\)](#).

The previous examples are all functionally equivalent, however, there are slight differences in their usability and their underlying implementations.

Tradeoffs

You might want to always use [ValueTuple](#) over [Tuple](#), and anonymous types, but there are tradeoffs you should consider. The [ValueTuple](#) types are mutable, whereas [Tuple](#) are read-only. Anonymous types can be used in expression trees, while tuples cannot. The following table is an overview of some of the key differences.

Key differences

NAME	ACCESS MODIFIER	TYPE	CUSTOM MEMBER NAME	DECONSTRUCTION SUPPORT	EXPRESSION TREE SUPPORT
Anonymous types	<code>internal</code>	<code>class</code>	✓	✗	✓
Tuple	<code>public</code>	<code>class</code>	✗	✗	✓
ValueTuple	<code>public</code>	<code>struct</code>	✓	✓	✗

Serialization

One important consideration when choosing a type, is whether or not it will need to be serialized. Serialization is the process of converting the state of an object into a form that can be persisted or transported. For more information, see [serialization](#). When serialization is important, creating a `class` or `struct` is preferred over anonymous types or tuple types.

Performance

Performance between these types depends on the scenario. The major impact involves the tradeoff between allocations and copying. In most scenarios, the impact is small. When major impacts could arise, measurements should be taken to inform the decision.

Conclusion

As a developer choosing between tuples and anonymous types, there are several factors to consider. Generally speaking, if you're not working with [expression trees](#), and you're comfortable with tuple syntax then choose [ValueTuple](#) as they provide a value type with the flexibility to name properties. If you're working with expression trees, and you'd prefer to name properties, choose anonymous types. Otherwise, use [Tuple](#).

See also

- [Anonymous types](#)
- [Expression trees](#)
- [Tuple types \(C# reference\)](#)
- [Tuples \(Visual Basic\)](#)
- [Type design guidelines](#)

Framework libraries

9/20/2022 • 3 minutes to read • [Edit Online](#)

.NET has an expansive standard set of class libraries, referred to as either the base class libraries (core set) or framework class libraries (complete set). These libraries provide implementations for many general and app-specific types, algorithms, and utility functionality. Both commercial and community libraries build on top of the framework class libraries, providing easy to use, off-the-shelf libraries for a wide set of computing tasks.

A subset of these libraries are provided with each .NET implementation. Base class library (BCL) APIs are expected with any .NET implementation, both because developers will want them and because popular libraries will need them to run. App-specific libraries above the BCL, such as ASP.NET, will not be available on all .NET implementations.

Base class library

The BCL provides the most foundational types and utility functionality and is the base of all other .NET class libraries. The BCL aims to provide general implementations without bias to any workload. Performance is an important consideration, since apps might prefer a particular policy, such as low-latency to high-throughput or low-memory to low-CPU usage. The BCL is intended to be high-performance generally, and take a middle-ground approach according to these various performance concerns. For most apps, this approach has been quite successful.

Primitives and other basic types

.NET includes a set of basic types that are used (to varying degrees) in all programs. These types contain data, such as numbers, strings, bytes, and arbitrary objects. The C# language includes keywords for these types. A sample set of these types is listed below, with the matching C# keywords.

TYPE	C# KEYWORD	DESCRIPTION
System.Object	<code>object</code>	The ultimate base class in the CLR type system. It is the root of the type hierarchy.
System.Int16	<code>short</code>	A 16-bit signed integer type. The unsigned UInt16 also exists.
System.Int32	<code>int</code>	A 32-bit signed integer type. The unsigned UInt32 also exists.
System.Single	<code>float</code>	A 32-bit floating-point type.
System.Decimal	<code>decimal</code>	A 128-bit decimal type.
System.Byte	<code>byte</code>	An unsigned 8-bit integer that represents a byte of memory.
System.Boolean	<code>bool</code>	A Boolean type that represents <code>true</code> or <code>false</code> .

TYPE	C# KEYWORD	DESCRIPTION
System.Char	<code>char</code>	A 16-bit numeric type that represents a Unicode character.
System.String	<code>string</code>	Represents a series of characters. Different than a <code>char[]</code> , but enables indexing into each individual <code>char</code> in the <code>string</code> .

Data structures

.NET includes a set of data structures that are the workhorses of many .NET apps. These are mostly collections, but also include other types.

- [Array](#) - Represents an array of strongly typed objects that can be accessed by index. Has a fixed size, per its construction.
- [List<T>](#) - Represents a strongly typed list of objects that can be accessed by index. Is automatically resized as needed.
- [Dictionary< TKey, TValue >](#) - Represents a collection of values that are indexed by a key. Values can be accessed via key. Is automatically resized as needed.
- [Uri](#) - Provides an object representation of a uniform resource identifier (URI) and easy access to the parts of the URI.
- [DateTime](#) - Represents an instant in time, typically expressed as a date and time of day.

Utility APIs

.NET includes a set of utility APIs that provide functionality for many important tasks.

- [HttpClient](#) - An API for sending HTTP requests and receiving HTTP responses from a resource identified by a URI.
- [XDocument](#) - An API for loading and querying XML documents with LINQ.
- [StreamReader](#) - An API for reading files.
- [StreamWriter](#) - An API for writing files.

App-Model APIs

There are many app-models that can be used with .NET, for example:

- [ASP.NET](#) - A web framework for building Web sites and services. Supported on Windows, Linux, and macOS (depends on ASP.NET version).
- [Xamarin](#) - An app platform for building Android and iOS apps with .NET and C#. Supported on Windows and macOS.
- [Windows Desktop](#) - Includes Windows Presentation Foundation (WPF) and Windows Forms.

.NET class library overview

9/20/2022 • 4 minutes to read • [Edit Online](#)

.NET APIs include classes, interfaces, delegates, and value types that expedite and optimize the development process and provide access to system functionality. To facilitate interoperability between languages, most .NET types are CLS-compliant and can therefore be used from any programming language whose compiler conforms to the common language specification (CLS).

.NET types are the foundation on which .NET applications, components, and controls are built. .NET includes types that perform the following functions:

- Represent base data types and exceptions.
- Encapsulate data structures.
- Perform I/O.
- Access information about loaded types.
- Invoke .NET security checks.
- Provide data access, rich client-side GUI, and server-controlled, client-side GUI.

.NET provides a rich set of interfaces, as well as abstract and concrete (non-abstract) classes. You can use the concrete classes as-is or, in many cases, derive your own classes from them. To use the functionality of an interface, you can either create a class that implements the interface or derive a class from one of the .NET classes that implements the interface.

Naming conventions

.NET types use a dot syntax naming scheme that connotes a hierarchy. This technique groups related types into namespaces so they can be searched and referenced more easily. The first part of the full name—up to the rightmost dot—is the namespace name. The last part of the name is the type name. For example,

`System.Collections.Generic.List<T>` represents the `List<T>` type, which belongs to the `System.Collections.Generic` namespace. The types in `System.Collections.Generic` can be used to work with generic collections.

This naming scheme makes it easy for library developers extending .NET to create hierarchical groups of types and name them in a consistent, informative manner. It also allows types to be unambiguously identified by their full name (that is, by their namespace and type name), which prevents type name collisions. Library developers are expected to use the following convention when creating names for their namespaces:

CompanyName.TechnologyName

For example, the namespace `Microsoft.Word` conforms to this guideline.

The use of naming patterns to group related types into namespaces is a useful way to build and document class libraries. However, this naming scheme has no effect on visibility, member access, inheritance, security, or binding. A namespace can be partitioned across multiple assemblies and a single assembly can contain types from multiple namespaces. The assembly provides the formal structure for versioning, deployment, security, loading, and visibility in the common language runtime.

For more information on namespaces and type names, see [Common Type System](#).

System namespace

The [System](#) namespace is the root namespace for fundamental types in .NET. This namespace includes classes that represent the base data types used by all applications, for example, [Object](#) (the root of the inheritance hierarchy), [Byte](#), [Char](#), [Array](#), [Int32](#), and [String](#). Many of these types correspond to the primitive data types that your programming language uses. When you write code using .NET types, you can use your language's corresponding keyword when a .NET base data type is expected.

The following table lists the base types that .NET supplies, briefly describes each type, and indicates the corresponding type in Visual Basic, C#, C++, and F#.

CATEGORY	CLASS NAME	DESCRIPTION	VISUAL BASIC DATA TYPE	C# DATA TYPE	C++/CLI DATA TYPE	F# DATA TYPE
Integer	Byte	An 8-bit unsigned integer.	<code>Byte</code>	<code>byte</code>	<code>unsigned char</code>	<code>byte</code>
	SByte	An 8-bit signed integer. Not CLS-compliant.	<code>SByte</code>	<code>sbyte</code>	<code>char</code> or <code>signed char</code>	<code>sbyte</code>
	Int16	A 16-bit signed integer.	<code>Short</code>	<code>short</code>	<code>short</code>	<code>int16</code>
	Int32	A 32-bit signed integer.	<code>Integer</code>	<code>int</code>	<code>int</code> or <code>long</code>	<code>int</code>
	Int64	A 64-bit signed integer.	<code>Long</code>	<code>long</code>	<code>__int64</code>	<code>int64</code>
	UInt16	A 16-bit unsigned integer. Not CLS-compliant.	<code>UShort</code>	<code>ushort</code>	<code>unsigned short</code>	<code>uint16</code>
	UInt32	A 32-bit unsigned integer. Not CLS-compliant.	<code>UInteger</code>	<code>uint</code>	<code>unsigned int</code> or <code>unsigned long</code>	<code>uint32</code>
	UInt64	A 64-bit unsigned integer. Not CLS-compliant.	<code>ULong</code>	<code>ulong</code>	<code>unsigned __int64</code>	<code>uint64</code>

Category	Class Name	Description	Visual Basic Data Type	C# Data Type	C++/CLI Data Type	F# Data Type
Floating point	Half	A half-precision (16-bit) floating-point number.				
	Single	A single-precision (32-bit) floating-point number.	Single	float	float	float32 or single
	Double	A double-precision (64-bit) floating-point number.	Double	double	double	float or double
Logical	Boolean	A Boolean value (true or false).	Boolean	bool	bool	bool
Other	Char	A Unicode (16-bit) character.	Char	char	wchar_t	char
	Decimal	A decimal (128-bit) value.	Decimal	decimal	Decimal	decimal
	IntPtr	A signed integer whose size depends on the underlying platform (a 32-bit value on a 32-bit platform and a 64-bit value on a 64-bit platform).		nint		unativeint
	UIntPtr	An unsigned integer whose size depends on the underlying platform (a 32-bit value on a 32-bit platform and a 64-bit value on a 64-bit platform). Not CLS-compliant.		nuint		unativeint

Category	Class Name	Description	Visual Basic Data Type	C# Data Type	C++/CLI Data Type	F# Data Type
	Object	The root of the object hierarchy.	Object	object	Object^	obj
	String	An immutable, fixed-length string of Unicode characters.	String	string	String^	string

In addition to the base data types, the [System](#) namespace contains over 100 classes, ranging from classes that handle exceptions to classes that deal with core runtime concepts, such as application domains and the garbage collector. The [System](#) namespace also contains many second-level namespaces.

For more information about namespaces, use the [.NET API Browser](#) to browse the .NET Class Library. The API reference documentation provides documentation on each namespace, its types, and each of their members.

See also

- [Common Type System](#)
- [.NET API Browser](#)
- [Overview](#)

Generics in .NET

9/20/2022 • 9 minutes to read • [Edit Online](#)

Generics let you tailor a method, class, structure, or interface to the precise data type it acts upon. For example, instead of using the [Hashtable](#) class, which allows keys and values to be of any type, you can use the [Dictionary< TKey, TValue >](#) generic class and specify the types allowed for the key and the value. Among the benefits of generics are increased code reusability and type safety.

Define and use generics

Generics are classes, structures, interfaces, and methods that have placeholders (type parameters) for one or more of the types that they store or use. A generic collection class might use a type parameter as a placeholder for the type of objects that it stores. The type parameters appear as the types of its fields and the parameter types of its methods. A generic method might use its type parameter as the type of its return value or as the type of one of its formal parameters. The following code illustrates a simple generic class definition.

```
generic<typename T>
public ref class Generics
{
public:
    T Field;
};
```

```
public class Generic<T>
{
    public T Field;
}
```

```
Public Class Generic(Of T)
    Public Field As T

End Class
```

When you create an instance of a generic class, you specify the actual types to substitute for the type parameters. This establishes a new generic class, referred to as a constructed generic class, with your chosen types substituted everywhere that the type parameters appear. The result is a type-safe class that is tailored to your choice of types, as the following code illustrates.

```
static void Main()
{
    Generics<String^>^ g = gcnew Generics<String^>();
    g->Field = "A string";
    //...
    Console::WriteLine("Generics.Field           = \"{0}\\"", g->Field);
    Console::WriteLine("Generics.Field.GetType() = {0}", g->Field->GetType()->FullName);
}
```

```

public static void Main()
{
    Generic<string> g = new Generic<string>();
    g.Field = "A string";
    //...
    Console.WriteLine("Generic.Field      = \'{0}\'", g.Field);
    Console.WriteLine("Generic.Field.GetType() = {0}", g.Field.GetType().FullName);
}

```

```

Public Shared Sub Main()
    Dim g As New Generic(Of String)
    g.Field = "A string"
    '...
    Console.WriteLine("Generic.Field      = \"{0}\"", g.Field)
    Console.WriteLine("Generic.Field.GetType() = {0}", g.Field.GetType().FullName)
End Sub

```

Terminology

The following terms are used to discuss generics in .NET:

- A *generic type definition* is a class, structure, or interface declaration that functions as a template, with placeholders for the types that it can contain or use. For example, the [System.Collections.Generic.Dictionary<TKey,TValue>](#) class can contain two types: keys and values. Because a generic type definition is only a template, you cannot create instances of a class, structure, or interface that is a generic type definition.
- *Generic type parameters*, or *type parameters*, are the placeholders in a generic type or method definition. The [System.Collections.Generic.Dictionary<TKey,TValue>](#) generic type has two type parameters, `TKey` and `TValue`, that represent the types of its keys and values.
- A *constructed generic type*, or *constructed type*, is the result of specifying types for the generic type parameters of a generic type definition.
- A *generic type argument* is any type that is substituted for a generic type parameter.
- The general term *generic type* includes both constructed types and generic type definitions.
- *Covariance* and *contravariance* of generic type parameters enable you to use constructed generic types whose type arguments are more derived (covariance) or less derived (contravariance) than a target constructed type. Covariance and contravariance are collectively referred to as *variance*. For more information, see [Covariance and contravariance](#).
- *Constraints* are limits placed on generic type parameters. For example, you might limit a type parameter to types that implement the [System.Collections.Generic.IComparer<T>](#) generic interface, to ensure that instances of the type can be ordered. You can also constrain type parameters to types that have a particular base class, that have a parameterless constructor, or that are reference types or value types. Users of the generic type cannot substitute type arguments that do not satisfy the constraints.
- A *generic method definition* is a method with two parameter lists: a list of generic type parameters and a list of formal parameters. Type parameters can appear as the return type or as the types of the formal parameters, as the following code shows.

```
generic<typename T>
T Generic(T arg)
{
    T temp = arg;
    //...
    return temp;
}
```

```
T Generic<T>(T arg)
{
    T temp = arg;
    //...
    return temp;
}
```

```
Function Generic(Of T)(ByVal arg As T) As T
    Dim temp As T = arg
    '...
    Return temp
End Function
```

Generic methods can appear on generic or nongeneric types. It's important to note that a method is not generic just because it belongs to a generic type, or even because it has formal parameters whose types are the generic parameters of the enclosing type. A method is generic only if it has its own list of type parameters. In the following code, only method `G` is generic.

```
ref class A
{
    generic<typename T>
    T G(T arg)
    {
        T temp = arg;
        //...
        return temp;
    }
};

generic<typename T>
ref class Generic
{
    T M(T arg)
    {
        T temp = arg;
        //...
        return temp;
    }
};
```

```

class A
{
    T G<T>(T arg)
    {
        T temp = arg;
        //...
        return temp;
    }
}
class Generic<T>
{
    T M(T arg)
    {
        T temp = arg;
        //...
        return temp;
    }
}

```

```

Class A
    Function G(Of T)(ByVal arg As T) As T
        Dim temp As T = arg
        '...
        Return temp
    End Function
End Class
Class Generic(Of T)
    Function M(ByVal arg As T) As T
        Dim temp As T = arg
        '...
        Return temp
    End Function
End Class

```

Advantages and disadvantages of generics

There are many advantages to using generic collections and delegates:

- Type safety. Generics shift the burden of type safety from you to the compiler. There is no need to write code to test for the correct data type because it is enforced at compile time. The need for type casting and the possibility of run-time errors are reduced.
- Less code and code is more easily reused. There is no need to inherit from a base type and override members. For example, the `LinkedList<T>` is ready for immediate use. For example, you can create a linked list of strings with the following variable declaration:

```
LinkedList<String^>^ llist = gcnew LinkedList<String^>();
```

```
LinkedList<string> llist = new LinkedList<string>();
```

```
Dim llist As New LinkedList(Of String)()
```

- Better performance. Generic collection types generally perform better for storing and manipulating value types because there is no need to box the value types.
- Generic delegates enable type-safe callbacks without the need to create multiple delegate classes. For

example, the [Predicate<T>](#) generic delegate allows you to create a method that implements your own search criteria for a particular type and to use your method with methods of the [Array](#) type such as [Find](#), [FindLast](#), and [FindAll](#).

- Generics streamline dynamically generated code. When you use generics with dynamically generated code you do not need to generate the type. This increases the number of scenarios in which you can use lightweight dynamic methods instead of generating entire assemblies. For more information, see [How to: Define and Execute Dynamic Methods](#) and [DynamicMethod](#).

The following are some limitations of generics:

- Generic types can be derived from most base classes, such as [MarshalByRefObject](#) (and constraints can be used to require that generic type parameters derive from base classes like [MarshalByRefObject](#)). However, .NET does not support context-bound generic types. A generic type can be derived from [ContextBoundObject](#), but trying to create an instance of that type causes a [TypeLoadException](#).
- Enumerations cannot have generic type parameters. An enumeration can be generic only incidentally (for example, because it is nested in a generic type that is defined using Visual Basic, C#, or C++). For more information, see "Enumerations" in [Common Type System](#).
- Lightweight dynamic methods cannot be generic.
- In Visual Basic, C#, and C++, a nested type that is enclosed in a generic type cannot be instantiated unless types have been assigned to the type parameters of all enclosing types. Another way of saying this is that in reflection, a nested type that is defined using these languages includes the type parameters of all its enclosing types. This allows the type parameters of enclosing types to be used in the member definitions of a nested type. For more information, see "Nested Types" in [MakeGenericType](#).

NOTE

A nested type that is defined by emitting code in a dynamic assembly or by using the [Ilasm.exe \(IL Assembler\)](#) is not required to include the type parameters of its enclosing types; however, if it does not include them, the type parameters are not in scope in the nested class.

For more information, see "Nested Types" in [MakeGenericType](#).

Class library and language support

.NET provides a number of generic collection classes in the following namespaces:

- The [System.Collections.Generic](#) namespace contains most of the generic collection types provided by .NET, such as the [List<T>](#) and [Dictionary< TKey, TValue >](#) generic classes.
- The [System.Collections.ObjectModel](#) namespace contains additional generic collection types, such as the [ReadOnlyCollection<T>](#) generic class, that are useful for exposing object models to users of your classes.

Generic interfaces for implementing sort and equality comparisons are provided in the [System](#) namespace, along with generic delegate types for event handlers, conversions, and search predicates.

The [System.Numerics](#) namespace provides generic interfaces for mathematical functionality (available in .NET 7 and later versions). For more information, see [Generic math](#).

Support for generics has been added to the [System.Reflection](#) namespace for examining generic types and generic methods, to [System.Reflection.Emit](#) for emitting dynamic assemblies that contain generic types and methods, and to [System.CodeDom](#) for generating source graphs that include generics.

The common language runtime provides new opcodes and prefixes to support generic types in Microsoft

intermediate language (MSIL), including [Stelem](#), [Ldelem](#), [Unbox_Any](#), [Constrained](#), and [Readonly](#).

Visual C++, C#, and Visual Basic all provide full support for defining and using generics. For more information about language support, see [Generic Types in Visual Basic](#), [Introduction to Generics](#), and [Overview of Generics in Visual C++](#).

Nested types and generics

A type that is nested in a generic type can depend on the type parameters of the enclosing generic type. The common language runtime considers nested types to be generic, even if they do not have generic type parameters of their own. When you create an instance of a nested type, you must specify type arguments for all enclosing generic types.

Related articles

TITLE	DESCRIPTION
Generic Collections in .NET	Describes generic collection classes and other generic types in .NET.
Generic Delegates for Manipulating Arrays and Lists	Describes generic delegates for conversions, search predicates, and actions to be taken on elements of an array or collection.
Generic math	Describes how you can perform mathematical operations generically.
Generic Interfaces	Describes generic interfaces that provide common functionality across families of generic types.
Covariance and Contravariance	Describes covariance and contravariance in generic type parameters.
Commonly Used Collection Types	Provides summary information about the characteristics and usage scenarios of the collection types in .NET, including generic types.
When to Use Generic Collections	Describes general rules for determining when to use generic collection types.
How to: Define a Generic Type with Reflection Emit	Explains how to generate dynamic assemblies that include generic types and methods.
Generic Types in Visual Basic	Describes the generics feature for Visual Basic users, including how-to topics for using and defining generic types.
Introduction to Generics	Provides an overview of defining and using generic types for C# users.
Overview of Generics in Visual C++	Describes the generics feature for C++ users, including the differences between generics and templates.

Reference

- [System.Collections.Generic](#)

- [System.Collections.ObjectModel](#)
- [System.Reflection.Emit.OpCodes](#)

Generic types overview

9/20/2022 • 2 minutes to read • [Edit Online](#)

Developers use generics all the time in .NET, whether implicitly or explicitly. When you use LINQ in .NET, did you ever notice that you're working with `IEnumerable<T>`? Or if you ever saw an online sample of a "generic repository" for talking to databases using Entity Framework, did you see that most methods return `IQueryable<T>`? You may have wondered what the `T` is in these examples and why it's in there.

First introduced in .NET Framework 2.0, generics are essentially a "code template" that allows developers to define **type-safe** data structures without committing to an actual data type. For example, `List<T>` is a **generic collection** that can be declared and used with any type, such as `List<int>`, `List<string>`, or `List<Person>`.

To understand why generics are useful, let's take a look at a specific class before and after adding generics: `ArrayList`. In .NET Framework 1.0, the `ArrayList` elements were of type `Object`. Any element added to the collection was silently converted into an `Object`. The same would happen when reading elements from the list. This process is known as **boxing and unboxing**, and it impacts performance. Aside from performance, however, there's no way to determine the type of data in the list at compile time, which makes for some fragile code. Generics solve this problem by defining the type of data each instance of list will contain. For example, you can only add integers to `List<int>` and only add Persons to `List<Person>`.

Generics are also available at run time. The runtime knows what type of data structure you're using and can store it in memory more efficiently.

The following example is a small program that illustrates the efficiency of knowing the data structure type at run time:

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Diagnostics;

namespace GenericsExample {
    class Program {
        static void Main(string[] args) {
            //generic list
            List<int> ListGeneric = new List<int> { 5, 9, 1, 4 };
            //non-generic list
            ArrayList ListNonGeneric = new ArrayList { 5, 9, 1, 4 };
            // timer for generic list sort
            Stopwatch s = Stopwatch.StartNew();
            ListGeneric.Sort();
            s.Stop();
            Console.WriteLine($"Generic Sort: {ListGeneric} \n Time taken: {s.Elapsed.TotalMilliseconds}ms");

            //timer for non-generic list sort
            Stopwatch s2 = Stopwatch.StartNew();
            ListNonGeneric.Sort();
            s2.Stop();
            Console.WriteLine($"Non-Generic Sort: {ListNonGeneric} \n Time taken:
{s2.Elapsed.TotalMilliseconds}ms");
            Console.ReadLine();
        }
    }
}
```

This program produces output similar to the following:

```
Generic Sort: System.Collections.Generic.List`1[System.Int32]
Time taken: 0.0034ms
Non-Generic Sort: System.Collections.ArrayList
Time taken: 0.2592ms
```

The first thing you can notice here is that sorting the generic list is significantly faster than sorting the non-generic list. You might also notice that the type for the generic list is distinct (`[System.Int32]`), whereas the type for the non-generic list is generalized. Because the runtime knows the generic `List<int>` is of type `Int32`, it can store the list elements in an underlying integer array in memory, while the non-generic `ArrayList` has to cast each list element to an object. As this example shows, the extra casts take up time and slow down the list sort.

An additional advantage of the runtime knowing the type of your generic is a better debugging experience. When you're debugging a generic in C#, you know what type each element is in your data structure. Without generics, you would have no idea what type each element was.

See also

- [C# Programming Guide - Generics](#)

Generic collections in .NET

9/20/2022 • 2 minutes to read • [Edit Online](#)

The .NET class library provides a number of generic collection classes in the [System.Collections.Generic](#) and [System.Collections.ObjectModel](#) namespaces. For more detailed information about these classes, see [Commonly Used Collection Types](#).

System.Collections.Generic

Many of the generic collection types are direct analogs of nongeneric types. [Dictionary<TKey,TValue>](#) is a generic version of [Hashtable](#); it uses the generic structure [KeyValuePair<TKey,TValue>](#) for enumeration instead of [DictionaryEntry](#).

[List<T>](#) is a generic version of [ArrayList](#). There are generic [Queue<T>](#) and [Stack<T>](#) classes that correspond to the nongeneric versions.

There are generic and nongeneric versions of [SortedList<TKey,TValue>](#). Both versions are hybrids of a dictionary and a list. The [SortedDictionary<TKey,TValue>](#) generic class is a pure dictionary and has no nongeneric counterpart.

The [LinkedList<T>](#) generic class is a true linked list. It has no nongeneric counterpart.

System.Collections.ObjectModel

The [Collection<T>](#) generic class provides a base class for deriving your own generic collection types. The [ReadOnlyCollection<T>](#) class provides an easy way to produce a read-only collection from any type that implements the [IList<T>](#) generic interface. The [KeyedCollection<TKey,TItem>](#) generic class provides a way to store objects that contain their own keys.

Other generic types

The [Nullable<T>](#) generic structure allows you to use value types as if they could be assigned `null`. This can be useful when working with database queries, where fields that contain value types can be missing. The generic type parameter can be any value type.

NOTE

In C# and Visual Basic, it is not necessary to use [Nullable<T>](#) explicitly because the language has syntax for nullable types. See [Nullable value types \(C# reference\)](#) and [Nullable value types \(Visual Basic\)](#).

The [ArraySegment<T>](#) generic structure provides a way to delimit a range of elements within a one-dimensional, zero-based array of any type. The generic type parameter is the type of the array's elements.

The [EventHandler<TEventArgs>](#) generic delegate eliminates the need to declare a delegate type to handle events, if your event follows the event-handling pattern used by .NET. For example, suppose you have created a `MyEventArgs` class, derived from [EventArgs](#), to hold the data for your event. You can then declare the event as follows:

```
public:  
    event EventHandler<MyEventArgs^>^ MyEvent;
```

```
public event EventHandler<MyEventArgs> MyEvent;
```

```
Public Event MyEvent As EventHandler(Of MyEventArgs)
```

See also

- [System.Collections.Generic](#)
- [System.Collections.ObjectModel](#)
- [Generics](#)
- [Generic Delegates for Manipulating Arrays and Lists](#)
- [Generic Interfaces](#)

Generic Delegates for Manipulating Arrays and Lists

9/20/2022 • 2 minutes to read • [Edit Online](#)

This topic provides an overview of generic delegates for conversions, search predicates, and actions to be taken on elements of an array or collection.

Generic Delegates for Manipulating Arrays and Lists

The [Action<T>](#) generic delegate represents a method that performs some action on an element of the specified type. You can create a method that performs the desired action on the element, create an instance of the [Action<T>](#) delegate to represent that method, and then pass the array and the delegate to the [Array.ForEach](#) static generic method. The method is called for every element of the array.

The [List<T>](#) generic class also provides a [ForEach](#) method that uses the [Action<T>](#) delegate. This method is not generic.

NOTE

This makes an interesting point about generic types and methods. The [Array.ForEach](#) method must be static (`Shared` in Visual Basic) and generic because [Array](#) is not a generic type; the only reason you can specify a type for [Array.ForEach](#) to operate on is that the method has its own type parameter list. By contrast, the nongeneric [List<T>.ForEach](#) method belongs to the generic class [List<T>](#), so it simply uses the type parameter of its class. The class is strongly typed, so the method can be an instance method.

The [Predicate<T>](#) generic delegate represents a method that determines whether a particular element meets criteria you define. You can use it with the following static generic methods of [Array](#) to search for an element or a set of elements: [Exists](#), [Find](#), [FindAll](#), [FindIndex](#), [FindLast](#), [FindLastIndex](#), and [TrueForAll](#).

[Predicate<T>](#) also works with the corresponding nongeneric instance methods of the [List<T>](#) generic class.

The [Comparison<T>](#) generic delegate allows you to provide a sort order for array or list elements that do not have a native sort order, or to override the native sort order. Create a method that performs the comparison, create an instance of the [Comparison<T>](#) delegate to represent your method, and then pass the array and the delegate to the [Array.Sort<T>\(T\[\], Comparison<T>\)](#) static generic method. The [List<T>](#) generic class provides a corresponding instance method overload, [List<T>.Sort\(Comparison<T>\)](#).

The [Converter<TInput,TOutput>](#) generic delegate allows you to define a conversion between two types, and to convert an array of one type into an array of the other, or to convert a list of one type to a list of the other. Create a method that converts the elements of the existing list to a new type, create a delegate instance to represent the method, and use the [Array.ConvertAll](#) generic static method to produce an array of the new type from the original array, or the [List<T>.ConvertAll<TOutput>\(Converter<T,TOutput>\)](#) generic instance method to produce a list of the new type from the original list.

Chaining Delegates

Many of the methods that use these delegates return an array or list, which can be passed to another method. For example, if you want to select certain elements of an array, convert those elements to a new type, and save them in a new array, you can pass the array returned by the [FindAll](#) generic method to the [ConvertAll](#) generic method. If the new element type lacks a natural sort order, you can pass the array returned by the [ConvertAll](#) generic method to the [Sort<T>\(T\[\], Comparison<T>\)](#) generic method.

See also

- [System.Collections.Generic](#)
- [System.Collections.ObjectModel](#)
- [Generics](#)
- [Generic Collections in the .NET](#)
- [Generic Interfaces](#)
- [Covariance and Contravariance](#)

Generic math

9/20/2022 • 10 minutes to read • [Edit Online](#)

.NET 7 introduces new math-related generic interfaces to the base class library. The availability of these interfaces means you can constrain a type parameter of a generic type or method to be "number-like". In addition, C# 11 and later lets you define `static virtual interface members`. Because operators must be declared as `static`, this new C# feature lets operators be declared in the new interfaces for number-like types.

Together, these innovations allow you to perform mathematical operations generically—that is, without having to know the exact type you're working with. For example, if you wanted to write a method that adds two numbers, previously you had to add an overload of the method for each type (for example,

`static int Add(int first, int second)` and `static float Add(float first, float second)`). Now you can write a single, generic method, where the type parameter is constrained to be a number-like type. For example:

```
static T Add<T>(T left, T right)
    where T : INumber<TSelf>
{
    return left + right;
}
```

In this method, the type parameter `T` is constrained to be a type that implements the new `INumber<TSelf>` interface. `INumber<TSelf>` implements the `IAdditionOperators<TSelf,TOther,TResult>` interface, which contains the `+` operator. That allows the method to generically add the two numbers. The method can be used with any of .NET's built-in numeric types, because they've all been updated to implement `INumber<TSelf>` in .NET 7.

Library authors will benefit most from the generic math interfaces, because they can simplify their code base by removing "redundant" overloads. Other developers will benefit indirectly, because the APIs they consume may start supporting more types.

The interfaces

When designing the interfaces, they needed to be both fine-grained enough that users can define their own interfaces on top, while also being granular enough that they're easy to consume. To that extent, there are a few core numeric interfaces that most users will interact with, such as `INumber<TSelf>` and `IBinaryInteger<TSelf>`. The more fine-grained interfaces, such as `IAdditionOperators<TSelf,TOther,TResult>` and `ITrigonometricFunctions<TSelf>`, support these types and are available for developers who define their own domain-specific numeric interfaces.

- [Numeric interfaces](#)
- [Operator interfaces](#)
- [Function interfaces](#)
- [Parsing and formatting interfaces](#)

Numeric interfaces

This section describes the interfaces in `System.Numerics` that describe number-like types and the functionality available to them.

INTERFACE NAME	DESCRIPTION
----------------	-------------

INTERFACE NAME	DESCRIPTION
IBinaryFloatingPointeee754<TSelf>	Exposes APIs common to <i>binary</i> floating-point types ¹ that implement the IEEE 754 standard.
IBinaryInteger<TSelf>	Exposes APIs common to binary integers ² .
IBinaryNumber<TSelf>	Exposes APIs common to binary numbers.
IFloatingPoint<TSelf>	Exposes APIs common to floating-point types.
IFloatingPointeee754<TSelf>	Exposes APIs common to floating-point types that implement the IEEE 754 standard.
INumber<TSelf>	Exposes APIs common to comparable number types (effectively the "real" number domain).
INumberBase<TSelf>	Exposes APIs common to all number types (effectively the "complex" number domain).
ISignedNumber<TSelf>	Exposes APIs common to all signed number types (such as the concept of <code>NegativeOne</code>).
IUnsignedNumber<TSelf>	Exposes APIs common to all unsigned number types.
IAdditiveIdentity<TSelf,TResult>	Exposes the concept of <code>(x + T.AdditiveIdentity) == x</code> .
IMinMaxValue<TSelf>	Exposes the concept of <code>T.MinValue</code> and <code>T.MaxValue</code> .
IMultiplicativeIdentity<TSelf,TResult>	Exposes the concept of <code>(x * T.MultiplicativeIdentity) == x</code> .

¹The binary **floating-point types** are **Double** (`double`), **Half**, and **Single** (`float`).

²The binary **integer types** are **Byte** (`byte`), **Int16** (`short`), **Int32** (`int`), **Int64** (`long`), **Int128**, **IntPtr** (`nint`), **SByte** (`sbyte`), **UInt16** (`ushort`), **UInt32** (`uint`), **UInt64** (`ulong`), **UInt128**, and **UIntPtr** (`nuint`).

The interface you're most likely to use directly is [INumber<TSelf>](#), which roughly corresponds to a *real*/number. If a type implements this interface, it means that a value has a sign (this includes `unsigned` types, which are considered positive) and can be compared to other values of the same type. [INumberBase<TSelf>](#) confers more advanced concepts, such as *complex* and *imaginary* numbers, for example, the square root of a negative number. Other interfaces, such as [IFloatingPointeee754<TSelf>](#), were created because not all operations make sense for all number types—for example, calculating the floor of a number only makes sense for floating-point types. In the .NET base class library, the floating-point type **Double** implements [IFloatingPointeee754<TSelf>](#) but **Int32** doesn't.

Several of the interfaces are also implemented by various other types, including [Char](#), [DateOnly](#), [DateTime](#), [DateTimeOffset](#), [Decimal](#), [Guid](#), [TimeOnly](#), and [TimeSpan](#).

The following table shows some of the core APIs exposed by each interface.

INTERFACE	API NAME	DESCRIPTION
-----------	----------	-------------

INTERFACE	API NAME	DESCRIPTION
IBinaryInteger<TSelf>	DivRem	Computes the quotient and remainder simultaneously.
	LeadingZeroCount	Counts the number of leading zero bits in the binary representation.
	PopCount	Counts the number of set bits in the binary representation.
	RotateLeft	Rotates bits left, sometimes also called a circular left shift.
	RotateRight	Rotates bits right, sometimes also called a circular right shift.
	TrailingZeroCount	Counts the number of trailing zero bits in the binary representation.
IFloatingPoint<TSelf>	Ceiling	Rounds the value towards positive infinity. +4.5 becomes +5, and -4.5 becomes -4.
	Floor	Rounds the value towards negative infinity. +4.5 becomes +4, and -4.5 becomes -5.
	Round	Rounds the value using the specified rounding mode.
	Truncate	Rounds the value towards zero. +4.5 becomes +4, and -4.5 becomes -4.
IFloatingPointeee754<TSelf>	E	Gets a value representing Euler's number for the type.
	Epsilon	Gets the smallest representable value that's greater than zero for the type.
	NaN	Gets a value representing NaN for the type.
	NegativeInfinity	Gets a value representing -Infinity for the type.
	NegativeZero	Gets a value representing -Zero for the type.
	Pi	Gets a value representing Pi for the type.
	PositiveInfinity	Gets a value representing +Infinity for the type.

INTERFACE	API NAME	DESCRIPTION
	Tau	Gets a value representing <code>Tau</code> ($2 * \pi$) for the type.
	(Other)	(Implements the full set of interfaces listed under Function interfaces .)
INumber<TSelf>	Clamp	Restricts a value to no more and no less than the specified min and max value.
	CopySign	Sets the sign of a specified value to the same as another specified value.
	Max	Returns the greater of two values, returning <code>NaN</code> if either input is <code>NaN</code> .
	MaxNumber	Returns the greater of two values, returning the number if one input is <code>NaN</code> .
	Min	Returns the lesser of two values, returning <code>NaN</code> if either input is <code>NaN</code> .
	MinNumber	Returns the lesser of two values, returning the number if one input is <code>NaN</code> .
	Sign	Returns -1 for negative values, 0 for zero, and +1 for positive values.
INumberBase<TSelf>	One	Gets the value 1 for the type.
	Radix	Gets the radix, or base, for the type. <code>Int32</code> returns 2. <code>Decimal</code> returns 10.
	Zero	Gets the value 0 for the type.
	CreateChecked	Creates a value, throwing an OverflowException if the input can't fit. ¹
	CreateSaturating	Creates a value, clamping to <code>T.MinValue</code> or <code>T.MaxValue</code> if the input can't fit. ¹
	CreateTruncating	Creates a value from another value, wrapping around if the input can't fit.
	IsComplexNumber	Returns true if the value has a non-zero real part and a non-zero imaginary part.

INTERFACE	API NAME	DESCRIPTION
	<code>IsEvenInteger</code>	Returns true if the value is an even integer. 2.0 returns <code>true</code> , and 2.2 returns <code>false</code> .
	<code>IsFinite</code>	Returns true if the value is not infinite and not <code>NaN</code> .
	<code>IsImaginaryNumber</code>	Returns true if the value has a zero real part. This means <code>0</code> is imaginary and <code>1 + 1i</code> isn't.
	<code>IsInfinity</code>	Returns true if the value represents infinity.
	<code>IsInteger</code>	Returns true if the value is an integer. 2.0 and 3.0 return <code>true</code> , and 2.2 and 3.1 return <code>false</code> .
	<code>IsNaN</code>	Returns true if the value represents <code>NaN</code> .
	<code>IsNegative</code>	Returns true if the value is negative. This includes -0.0.
	<code>IsPositive</code>	Returns true if the value is positive. This includes 0 and +0.0.
	<code>IsRealNumber</code>	Returns true if the value has a zero imaginary part. This means 0 is real as are all <code>INumber<T></code> types.
	<code>IsZero</code>	Returns true if the value represents zero. This includes 0, +0.0, and -0.0.
	<code>MaxMagnitude</code>	Returns the value with a greater absolute value, returning <code>NaN</code> if either input is <code>NaN</code> .
	<code>MaxMagnitudeNumber</code>	Returns the value with a greater absolute value, returning the number if one input is <code>NaN</code> .
	<code>MinMagnitude</code>	Returns the value with a lesser absolute value, returning <code>NaN</code> if either input is <code>NaN</code> .
	<code>MinMagnitudeNumber</code>	Returns the value with a lesser absolute value, returning the number if one input is <code>NaN</code> .
<code>ISignedNumber<TSelf></code>	<code>NegativeOne</code>	Gets the value -1 for the type.

¹To help understand the behavior of the three `Create*` methods, consider the following examples.

Example when given a value that's too large:

- `byte.CreateChecked(384)` will throw an [OverflowException](#).
- `byte.CreateSaturating(384)` returns 255 because 384 is greater than `Byte.MaxValue` (which is 255).
- `byte.CreateTruncating(384)` returns 128 because it takes the lowest eight bits (384 has a hex representation of `0x0180`, and the lowest eight bits is `0x80`, which is 128).

Example when given a value that's too small:

- `byte.CreateChecked(-384)` will throw an [OverflowException](#).
- `byte.CreateSaturating(-384)` returns 0 because -384 is smaller than `Byte.MinValue` (which is 0).
- `byte.CreateTruncating(-384)` returns 128 because it takes the lowest 8 bits (384 has a hex representation of `0xFE80`, and the lowest 8 bits is `0x80`, which is 128).

The `Create*` methods also have some special considerations for IEEE 754 floating-point types, like `float` and `double`, as they have the special values `PositiveInfinity`, `NegativeInfinity`, and `NaN`. All three `Create*` APIs behave as `CreateSaturating`. Also, while `MinValue` and `MaxValue` represent the largest negative/positive "normal" number, the actual minimum and maximum values are `NegativeInfinity` and `PositiveInfinity`, so they clamp to these values instead.

Operator interfaces

The operator interfaces correspond to the various operators available to the C# language.

- They explicitly don't pair operations such as multiplication and division since that isn't correct for all types. For example, `Matrix4x4 * Matrix4x4` is valid, but `Matrix4x4 / Matrix4x4` isn't valid.
- They typically allow the input and result types to differ to support scenarios such as dividing two integers to obtain a `double`, for example, `3 / 2 = 1.5`, or calculating the average of a set of integers.

INTERFACE NAME	DEFINED OPERATORS
<code>IAdditionOperators<TSelf,TOther,TResult></code>	<code>x + y</code>
<code>IBitwiseOperators<TSelf,TOther,TResult></code>	<code>x & y</code> , <code>x y</code> , <code>x ^ y</code> , and <code>~x</code>
<code>IComparisonOperators<TSelf,TOther,TResult></code>	<code>x < y</code> , <code>x > y</code> , <code>x <= y</code> , and <code>x >= y</code>
<code>IDecrementOperators<TSelf></code>	<code>--x</code> and <code>x--</code>
<code>IDivisionOperators<TSelf,TOther,TResult></code>	<code>x / y</code>
<code>IEqualityOperators<TSelf,TOther,TResult></code>	<code>x == y</code> and <code>x != y</code>
<code>IIncrementOperators<TSelf></code>	<code>++x</code> and <code>x++</code>
<code>IModulusOperators<TSelf,TOther,TResult></code>	<code>x % y</code>
<code>IMultiplyOperators<TSelf,TOther,TResult></code>	<code>x * y</code>
<code>IShiftOperators<TSelf,TOther,TResult></code>	<code>x << y</code> and <code>x >> y</code>
<code>ISubtractionOperators<TSelf,TOther,TResult></code>	<code>x - y</code>

INTERFACE NAME	DEFINED OPERATORS
IUnaryNegationOperators<TSelf,TResult>	-x
IUnaryPlusOperators<TSelf,TResult>	+x

NOTE

Some of the interfaces define a checked operator in addition to a regular unchecked operator. Checked operators are called in checked contexts and allow a user-defined type to define overflow behavior. If you implement a checked operator, for example, [CheckedSubtraction\(TSelf, TOther\)](#), you must also implement the unchecked operator, for example, [Subtraction\(TSelf, TOther\)](#).

Function interfaces

The function interfaces define common mathematical APIs that apply more broadly than to a specific [numeric interface](#). These interfaces are all implemented by [IFloatingPointee754<TSelf>](#), and may get implemented by other relevant types in the future.

INTERFACE NAME	DESCRIPTION
IExponentialFunctions<TSelf>	Exposes exponential functions supporting <code>e^x</code> , <code>e^x - 1</code> , <code>2^x</code> , <code>2^x - 1</code> , <code>10^x</code> , and <code>10^x - 1</code> .
IHyperbolicFunctions<TSelf>	Exposes hyperbolic functions supporting <code>acosh(x)</code> , <code>asinh(x)</code> , <code>atanh(x)</code> , <code>cosh(x)</code> , <code>sinh(x)</code> , and <code>tanh(x)</code> .
ILogarithmicFunctions<TSelf>	Exposes logarithmic functions supporting <code>ln(x)</code> , <code>ln(x + 1)</code> , <code>log2(x)</code> , <code>log2(x + 1)</code> , <code>log10(x)</code> , and <code>log10(x + 1)</code> .
IPowerFunctions<TSelf>	Exposes power functions supporting <code>x^y</code> .
IRootFunctions<TSelf>	Exposes root functions supporting <code>cbrt(x)</code> and <code>sqrt(x)</code> .
ITrigonometricFunctions<TSelf>	Exposes trigonometric functions supporting <code>acos(x)</code> , <code>asin(x)</code> , <code>atan(x)</code> , <code>cos(x)</code> , <code>sin(x)</code> , and <code>tan(x)</code> .

Parsing and formatting interfaces

Parsing and formatting are core concepts in programming. They're commonly used when converting user input to a given type or displaying a type to the user. These interfaces are in the [System](#) namespace.

INTERFACE NAME	DESCRIPTION
IParsable<TSelf>	Exposes support for <code>T.Parse(string, IFormatProvider)</code> and <code>T.TryParse(string, IFormatProvider, out Tself)</code> .
ISpanParsable<TSelf>	Exposes support for <code>T.Parse(ReadOnlySpan<char>, IFormatProvider)</code> and <code>T.TryParse(ReadOnlySpan<char>, IFormatProvider, out Tself)</code> .

INTERFACE NAME	DESCRIPTION
IFormattable ¹	Exposes support for <code>value.ToString(string, IFormatProvider)</code> .
ISpanFormattable ¹	Exposes support for <code>value.TryFormat(Span<char>, out int, ReadOnlySpan<char>, IFormatProvider)</code>

¹This interface is not new, nor is it generic. However, it's implemented by all number types and represents the inverse operation of [IParsable](#).

For example, the following program takes two numbers as input, reading them from the console using a generic method where the type parameter is constrained to be [IParsable<TSelf>](#). It calculates the average using a generic method where the type parameters for the input and result values are constrained to be [INumber<TSelf>](#), and then displays the result to the console.

```
using System.Globalization;
using System.Numerics;

static TResult Average<T, TResult>(T first, T second)
    where T : INumber<T>
    where TResult : INumber<TResult>
{
    return TResult.CreateChecked( (first + second) / T.CreateChecked(2) );
}

static T ParseInvariant<T>(string s)
    where T : IParsable<T>
{
    return T.Parse(s, CultureInfo.InvariantCulture);
}

Console.WriteLine("First number: ");
var left = ParseInvariant<float>(Console.ReadLine());

Console.WriteLine("Second number: ");
var right = ParseInvariant<float>(Console.ReadLine());

Console.WriteLine($"Result: {Average<float, float>(left, right)}");

/* This code displays output similar to:

First number: 5.0
Second number: 6
Result: 5.5
*/
```

See also

- [Generic math in .NET 7 \(blog post\)](#)

Generic interfaces in .NET

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article provides an overview of .NET's generic interfaces that provide common functionality across families of generic types.

Generic interfaces provide type-safe counterparts to nongeneric interfaces for ordering and equality comparisons, and for functionality that's shared by generic collection types. .NET 7 introduces generic interfaces for number-like types, for example, [System.Numerics.INumber<TSelf>](#). These interfaces let you define generic methods that provide mathematical functionality, where the generic type parameter is constrained to be a type that implements a generic, numeric interface.

NOTE

The type parameters of several generic interfaces are marked covariant or contravariant, providing greater flexibility in assigning and using types that implement these interfaces. For more information, see [Covariance and Contravariance](#).

Equality and ordering comparisons

- In the [System](#) namespace, the [System.IComparable<T>](#) and [System.IEquatable<T>](#) generic interfaces, like their nongeneric counterparts, define methods for ordering comparisons and equality comparisons, respectively. Types implement these interfaces to provide the ability to perform such comparisons.
- In the [System.Collections.Generic](#) namespace, the [IComparer<T>](#) and [IEqualityComparer<T>](#) generic interfaces offer a way to define an ordering or equality comparison for types that don't implement the [System.IComparable<T>](#) or [System.IEquatable<T>](#) interface. They also provide a way to redefine those relationships for types that do.

These interfaces are used by methods and constructors of many of the generic collection classes. For example, you can pass a generic [IComparer<T>](#) object to the constructor of the [SortedDictionary< TKey, TValue >](#) class to specify a sort order for a type that does not implement generic [System.IComparable<T>](#). There are overloads of the [Array.Sort](#) generic static method and the [List<T>.Sort](#) instance method for sorting arrays and lists using generic [IComparer<T>](#) implementations.

The [Comparer<T>](#) and [EqualityComparer<T>](#) generic classes provide base classes for implementations of the [IComparer<T>](#) and [IEqualityComparer<T>](#) generic interfaces, and also provide default ordering and equality comparisons through their respective [Comparer<T>.Default](#) and [EqualityComparer<T>.Default](#) properties.

Collection functionality

- The [ICollection<T>](#) generic interface is the basic interface for generic collection types. It provides basic functionality for adding, removing, copying, and enumerating elements. [ICollection<T>](#) inherits from both generic [IEnumerable<T>](#) and nongeneric [IEnumerable](#).
- The [IList<T>](#) generic interface extends the [ICollection<T>](#) generic interface with methods for indexed retrieval.
- The [IDictionary< TKey, TValue >](#) generic interface extends the [ICollection<T>](#) generic interface with methods for keyed retrieval. Generic dictionary types in the .NET base class library also implement the

nongeneric [IDictionary](#) interface.

- The [IEnumerable<T>](#) generic interface provides a generic enumerator structure. The [IEnumerator<T>](#) generic interface implemented by generic enumerators inherits the nongeneric [IEnumerator](#) interface; the [MoveNext](#) and [Reset](#) members, which do not depend on the type parameter `T`, appear only on the nongeneric interface. This means that any consumer of the nongeneric interface can also consume the generic interface.

Mathematical functionality

.NET 7 introduces generic interfaces in the [System.Numerics](#) namespace that describe number-like types and the functionality available to them. The 20 numeric types that the .NET base class library provides, for example, [Int32](#) and [Double](#), have been updated to implement these interfaces. The most prominent of these interfaces is [INumber<TSelf>](#), which roughly corresponds to a "real" number.

For more information about these interfaces, see [Generic math](#).

See also

- [System.Collections.Generic](#)
- [System.Collections.ObjectModel](#)
- [Generics](#)
- [Generic collections in .NET](#)
- [Generic delegates for manipulating arrays and lists](#)
- [Covariance and contravariance](#)

Covariance and contravariance in generics

9/20/2022 • 14 minutes to read • [Edit Online](#)

Covariance and *contravariance* are terms that refer to the ability to use a more derived type (more specific) or a less derived type (less specific) than originally specified. Generic type parameters support covariance and contravariance to provide greater flexibility in assigning and using generic types.

When you're referring to a type system, covariance, contravariance, and invariance have the following definitions. The examples assume a base class named `Base` and a derived class named `Derived`.

- **Covariance**

Enables you to use a more derived type than originally specified.

You can assign an instance of `IEnumerable<Derived>` to a variable of type `IEnumerable<Base>`.

- **Contravariance**

Enables you to use a more generic (less derived) type than originally specified.

You can assign an instance of `Action<Base>` to a variable of type `Action<Derived>`.

- **Invariance**

Means that you can use only the type originally specified. An invariant generic type parameter is neither covariant nor contravariant.

You cannot assign an instance of `List<Base>` to a variable of type `List<Derived>` or vice versa.

Covariant type parameters enable you to make assignments that look much like ordinary [Polymorphism](#), as shown in the following code.

```
IEnumerable<Derived> d = new List<Derived>();
IEnumerable<Base> b = d;
```

```
Dim d As IEnumerable(Of Derived) = New List(Of Derived)
Dim b As IEnumerable(Of Base) = d
```

The `List<T>` class implements the `IEnumerable<T>` interface, so `List<Derived>` (`List(Of Derived)`) in Visual Basic implements `IEnumerable<Derived>`. The covariant type parameter does the rest.

Contravariance, on the other hand, seems counterintuitive. The following example creates a delegate of type `Action<Base>` (`Action(Of Base)` in Visual Basic), and then assigns that delegate to a variable of type `Action<Derived>`.

```
Action<Base> b = (target) => { Console.WriteLine(target.GetType().Name); };
Action<Derived> d = b;
d(new Derived());
```

```

Dim b As Action(Of Base) = Sub(target As Base)
    Console.WriteLine(target.GetType().Name)
End Sub
Dim d As Action(Of Derived) = b
d(New Derived())

```

This seems backward, but it is type-safe code that compiles and runs. The lambda expression matches the delegate it's assigned to, so it defines a method that takes one parameter of type `Base` and that has no return value. The resulting delegate can be assigned to a variable of type `Action<Derived>` because the type parameter `T` of the `Action<T>` delegate is contravariant. The code is type-safe because `T` specifies a parameter type. When the delegate of type `Action<Base>` is invoked as if it were a delegate of type `Action<Derived>`, its argument must be of type `Derived`. This argument can always be passed safely to the underlying method, because the method's parameter is of type `Base`.

In general, a covariant type parameter can be used as the return type of a delegate, and contravariant type parameters can be used as parameter types. For an interface, covariant type parameters can be used as the return types of the interface's methods, and contravariant type parameters can be used as the parameter types of the interface's methods.

Covariance and contravariance are collectively referred to as *variance*. A generic type parameter that is not marked covariant or contravariant is referred to as *invariant*. A brief summary of facts about variance in the common language runtime:

- Variant type parameters are restricted to generic interface and generic delegate types.
- A generic interface or generic delegate type can have both covariant and contravariant type parameters.
- Variance applies only to reference types; if you specify a value type for a variant type parameter, that type parameter is invariant for the resulting constructed type.
- Variance does not apply to delegate combination. That is, given two delegates of types `Action<Derived>` and `Action<Base>` (`Action(Of Derived)` and `Action(Of Base)` in Visual Basic), you cannot combine the second delegate with the first although the result would be type safe. Variance allows the second delegate to be assigned to a variable of type `Action<Derived>`, but delegates can combine only if their types match exactly.
- Starting in C# 9, covariant return types are supported. An overriding method can declare a more derived return type than the method it overrides, and an overriding, read-only property can declare a more derived type.

Generic interfaces with covariant type parameters

Several generic interfaces have covariant type parameters, for example, `IEnumerable<T>`, `IEnumerator<T>`, `IQueryable<T>`, and `IGrouping< TKey, TElement >`. All the type parameters of these interfaces are covariant, so the type parameters are used only for the return types of the members.

The following example illustrates covariant type parameters. The example defines two types: `Base` has a static method named `PrintBases` that takes an `IEnumerable<Base>` (`IEnumerable(Of Base)` in Visual Basic) and prints the elements. `Derived` inherits from `Base`. The example creates an empty `List<Derived>` (`List(Of Derived)` in Visual Basic) and demonstrates that this type can be passed to `PrintBases` and assigned to a variable of type `IEnumerable<Base>` without casting. `List<T>` implements `IEnumerable<T>`, which has a single covariant type parameter. The covariant type parameter is the reason why an instance of `IEnumerable<Derived>` can be used instead of `IEnumerable<Base>`.

```

using System;
using System.Collections.Generic;

class Base
{
    public static void PrintBases(IEnumerable<Base> bases)
    {
        foreach(Base b in bases)
        {
            Console.WriteLine(b);
        }
    }
}

class Derived : Base
{
    public static void Main()
    {
        List<Derived> dlist = new List<Derived>();

        Derived.PrintBases(dlist);
        IEnumerable<Base> bIEnum = dlist;
    }
}

```

```

Imports System.Collections.Generic

Class Base
    Public Shared Sub PrintBases(ByVal bases As IEnumerable(Of Base))
        For Each b As Base In bases
            Console.WriteLine(b)
        Next
    End Sub
End Class

Class Derived
    Inherits Base

    Shared Sub Main()
        Dim dlist As New List(Of Derived)()

        Derived.PrintBases(dlist)
        Dim bIEnum As IEnumerable(Of Base) = dlist
    End Sub
End Class

```

Generic interfaces with contravariant type parameters

Several generic interfaces have contravariant type parameters; for example: [IComparer<T>](#), [IComparable<T>](#), and [IEqualityComparer<T>](#). These interfaces have only contravariant type parameters, so the type parameters are used only as parameter types in the members of the interfaces.

The following example illustrates contravariant type parameters. The example defines an abstract ([MustInherit](#) in Visual Basic) `Shape` class with an `Area` property. The example also defines a `ShapeAreaComparer` class that implements `IComparer<Shape>` (`IComparer(Of Shape)` in Visual Basic). The implementation of the `IComparer<T>.Compare` method is based on the value of the `Area` property, so `ShapeAreaComparer` can be used to sort `Shape` objects by area.

The `circle` class inherits `Shape` and overrides `Area`. The example creates a `SortedSet<T>` of `circle` objects, using a constructor that takes an `IComparer<Circle>` (`IComparer(Of Circle)` in Visual Basic). However, instead of

passing an `IComparer<Circle>`, the example passes a `ShapeAreaComparer` object, which implements `IComparer<Shape>`. The example can pass a comparer of a less derived type (`Shape`) when the code calls for a comparer of a more derived type (`circle`), because the type parameter of the `IComparer<T>` generic interface is contravariant.

When a new `Circle` object is added to the `SortedSet<Circle>`, the `IComparer<Shape>.Compare` method (`IComparer(Of Shape).Compare` method in Visual Basic) of the `ShapeAreaComparer` object is called each time the new element is compared to an existing element. The parameter type of the method (`shape`) is less derived than the type that is being passed (`Circle`), so the call is type safe. Contravariance enables `ShapeAreaComparer` to sort a collection of any single type, as well as a mixed collection of types, that derive from `Shape`.

```
using System;
using System.Collections.Generic;

abstract class Shape
{
    public virtual double Area { get { return 0; } }

}

class Circle : Shape
{
    private double r;
    public Circle(double radius) { r = radius; }
    public double Radius { get { return r; } }
    public override double Area { get { return Math.PI * r * r; } }
}

class ShapeAreaComparer : System.Collections.Generic.IComparer<Shape>
{
    int IComparer<Shape>.Compare(Shape a, Shape b)
    {
        if (a == null) return b == null ? 0 : -1;
        return b == null ? 1 : a.Area.CompareTo(b.Area);
    }
}

class Program
{
    static void Main()
    {
        // You can pass ShapeAreaComparer, which implements IComparer<Shape>,
        // even though the constructor for SortedSet<Circle> expects
        // IComparer<Circle>, because type parameter T of IComparer<T> is
        // contravariant.
        SortedSet<Circle> circlesByArea =
            new SortedSet<Circle>(new ShapeAreaComparer())
            { new Circle(7.2), new Circle(100), null, new Circle(.01) };

        foreach (Circle c in circlesByArea)
        {
            Console.WriteLine(c == null ? "null" : "Circle with area " + c.Area);
        }
    }
}

/* This code example produces the following output:

null
Circle with area 0.000314159265358979
Circle with area 162.860163162095
Circle with area 31415.9265358979
*/
```

```

Imports System.Collections.Generic

MustInherit Class Shape
    Public MustOverride ReadOnly Property Area As Double
End Class

Class Circle
    Inherits Shape

    Private r As Double
    Public Sub New(ByVal radius As Double)
        r = radius
    End Sub
    Public ReadOnly Property Radius As Double
        Get
            Return r
        End Get
    End Property
    Public Overrides ReadOnly Property Area As Double
        Get
            Return Math.Pi * r * r
        End Get
    End Property
End Class

Class ShapeAreaComparer
    Implements System.Collections.Generic.IComparer(Of Shape)

    Private Function AreaComparer(ByVal a As Shape, ByVal b As Shape) As Integer _
        Implements System.Collections.Generic.IComparer(Of Shape).Compare
        If a Is Nothing Then Return If(b Is Nothing, 0, -1)
        Return If(b Is Nothing, 1, a.Area.CompareTo(b.Area))
    End Function
End Class

Class Program
    Shared Sub Main()
        ' You can pass ShapeAreaComparer, which implements IComparer(Of Shape),
        ' even though the constructor for SortedSet(Of Circle) expects
        ' IComparer(Of Circle), because type parameter T of IComparer(Of T)
        ' is contravariant.
        Dim circlesByArea As New SortedSet(Of Circle)(New ShapeAreaComparer()) _
            From {New Circle(7.2), New Circle(100), Nothing, New Circle(.01)}

        For Each c As Circle In circlesByArea
            Console.WriteLine>If(c Is Nothing, "Nothing", "Circle with area " & c.Area))
        Next
    End Sub
End Class

' This code example produces the following output:
'
'Nothing
'Circle with area 0.000314159265358979
'Circle with area 162.860163162095
'Circle with area 31415.9265358979

```

Generic delegates with variant type parameters

The `Func` generic delegates, such as `Func<T,TResult>`, have covariant return types and contravariant parameter types. The `Action` generic delegates, such as `Action<T1,T2>`, have contravariant parameter types. This means that the delegates can be assigned to variables that have more derived parameter types and (in the case of the `Func` generic delegates) less derived return types.

NOTE

The last generic type parameter of the `Func` generic delegates specifies the type of the return value in the delegate signature. It is covariant (`out` keyword), whereas the other generic type parameters are contravariant (`in` keyword).

The following code illustrates this. The first piece of code defines a class named `Base`, a class named `Derived` that inherits `Base`, and another class with a `static` method (`Shared` in Visual Basic) named `MyMethod`. The method takes an instance of `Base` and returns an instance of `Derived`. (If the argument is an instance of `Derived`, `MyMethod` returns it; if the argument is an instance of `Base`, `MyMethod` returns a new instance of `Derived`.) In `Main()`, the example creates an instance of `Func<Base, Derived>` (`Func(Of Base, Derived)` in Visual Basic) that represents `MyMethod`, and stores it in the variable `f1`.

```
public class Base {}
public class Derived : Base {}

public class Program
{
    public static Derived MyMethod(Base b)
    {
        return b as Derived ?? new Derived();
    }

    static void Main()
    {
        Func<Base, Derived> f1 = MyMethod;
```

```
Public Class Base
End Class
Public Class Derived
    Inherits Base
End Class

Public Class Program
    Public Shared Function MyMethod(ByVal b As Base) As Derived
        Return If(TypeOf b Is Derived, b, New Derived())
    End Function

    Shared Sub Main()
        Dim f1 As Func(Of Base, Derived) = AddressOf MyMethod
```

The second piece of code shows that the delegate can be assigned to a variable of type `Func<Base, Base>` (`Func(Of Base, Base)` in Visual Basic), because the return type is covariant.

```
// Covariant return type.
Func<Base, Base> f2 = f1;
Base b2 = f2(new Base());
```

```
' Covariant return type.
Dim f2 As Func(Of Base, Base) = f1
Dim b2 As Base = f2(New Base())
```

The third piece of code shows that the delegate can be assigned to a variable of type `Func<Derived, Derived>` (`Func(Of Derived, Derived)` in Visual Basic), because the parameter type is contravariant.

```
// Contravariant parameter type.  
Func<Derived, Derived> f3 = f1;  
Derived d3 = f3(new Derived());
```

```
' Contravariant parameter type.  
Dim f3 As Func(Of Derived, Derived) = f1  
Dim d3 As Derived = f3(New Derived())
```

The final piece of code shows that the delegate can be assigned to a variable of type `Func<Derived, Base>` (`Func(Of Derived, Base)` in Visual Basic), combining the effects of the contravariant parameter type and the covariant return type.

```
// Covariant return type and contravariant parameter type.  
Func<Derived, Base> f4 = f1;  
Base b4 = f4(new Derived());
```

```
' Covariant return type and contravariant parameter type.  
Dim f4 As Func(Of Derived, Base) = f1  
Dim b4 As Base = f4(New Derived())
```

Variance in non-generic delegates

In the preceding code, the signature of `MyMethod` exactly matches the signature of the constructed generic delegate: `Func<Base, Derived>` (`Func(Of Base, Derived)` in Visual Basic). The example shows that this generic delegate can be stored in variables or method parameters that have more derived parameter types and less derived return types, as long as all the delegate types are constructed from the generic delegate type `Func<T,TResult>`.

This is an important point. The effects of covariance and contravariance in the type parameters of generic delegates are similar to the effects of covariance and contravariance in ordinary delegate binding (see [Variance in Delegates \(C#\)](#) and [Variance in Delegates \(Visual Basic\)](#)). However, variance in delegate binding works with all delegate types, not just with generic delegate types that have variant type parameters. Furthermore, variance in delegate binding enables a method to be bound to any delegate that has more restrictive parameter types and a less restrictive return type, whereas the assignment of generic delegates works only if both delegate types are constructed from the same generic type definition.

The following example shows the combined effects of variance in delegate binding and variance in generic type parameters. The example defines a type hierarchy that includes three types, from least derived (`Type1`) to most derived (`Type3`). Variance in ordinary delegate binding is used to bind a method with a parameter type of `Type1` and a return type of `Type3` to a generic delegate with a parameter type of `Type2` and a return type of `Type2`. The resulting generic delegate is then assigned to another variable whose generic delegate type has a parameter of type `Type3` and a return type of `Type1`, using the covariance and contravariance of generic type parameters. The second assignment requires both the variable type and the delegate type to be constructed from the same generic type definition, in this case, `Func<T,TResult>`.

```

using System;

public class Type1 {}
public class Type2 : Type1 {}
public class Type3 : Type2 {}

public class Program
{
    public static Type3 MyMethod(Type1 t)
    {
        return t as Type3 ?? new Type3();
    }

    static void Main()
    {
        Func<Type2, Type2> f1 = MyMethod;

        // Covariant return type and contravariant parameter type.
        Func<Type3, Type1> f2 = f1;
        Type1 t1 = f2(new Type3());
    }
}

```

```

Public Class Type1
End Class
Public Class Type2
    Inherits Type1
End Class
Public Class Type3
    Inherits Type2
End Class

Public Class Program
    Public Shared Function MyMethod(ByVal t As Type1) As Type3
        Return If(TypeOf t Is Type3, t, New Type3())
    End Function

    Shared Sub Main()
        Dim f1 As Func(Of Type2, Type2) = AddressOf MyMethod

        ' Covariant return type and contravariant parameter type.
        Dim f2 As Func(Of Type3, Type1) = f1
        Dim t1 As Type1 = f2(New Type3())
    End Sub
End Class

```

Define variant generic interfaces and delegates

Visual Basic and C# have keywords that enable you to mark the generic type parameters of interfaces and delegates as covariant or contravariant.

A covariant type parameter is marked with the `out` keyword (`out` keyword in Visual Basic). You can use a covariant type parameter as the return value of a method that belongs to an interface, or as the return type of a delegate. You cannot use a covariant type parameter as a generic type constraint for interface methods.

NOTE

If a method of an interface has a parameter that is a generic delegate type, a covariant type parameter of the interface type can be used to specify a contravariant type parameter of the delegate type.

A contravariant type parameter is marked with the `in` keyword (`In` keyword in Visual Basic). You can use a contravariant type parameter as the type of a parameter of a method that belongs to an interface, or as the type of a parameter of a delegate. You can use a contravariant type parameter as a generic type constraint for an interface method.

Only interface types and delegate types can have variant type parameters. An interface or delegate type can have both covariant and contravariant type parameters.

Visual Basic and C# do not allow you to violate the rules for using covariant and contravariant type parameters, or to add covariance and contravariance annotations to the type parameters of types other than interfaces and delegates.

For information and example code, see [Variance in Generic Interfaces \(C#\)](#) and [Variance in Generic Interfaces \(Visual Basic\)](#).

List of types

The following interface and delegate types have covariant and/or contravariant type parameters.

TYPE	COVARIANT TYPE PARAMETERS	CONTRAVARIANT TYPE PARAMETERS
<code>Action<T></code> to <code>Action<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16></code>		Yes
<code>Comparison<T></code>		Yes
<code>Converter<TInput,TOOutput></code>	Yes	Yes
<code>Func<TResult></code>	Yes	
<code>Func<T,TResult></code> to <code>Func<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16,TResult></code>	Yes	Yes
<code>IComparable<T></code>		Yes
<code>Predicate<T></code>		Yes
<code>IComparer<T></code>		Yes
<code>IEnumerable<T></code>	Yes	
<code>IEnumerator<T></code>	Yes	
<code>IEqualityComparer<T></code>		Yes
<code>IGrouping< TKey,TElement ></code>	Yes	
<code>IOrderedEnumerable< TElement ></code>	Yes	
<code>IOrderedQueryable< T ></code>	Yes	
<code>IQueryable<T></code>	Yes	

See also

- [Covariance and Contravariance \(C#\)](#)
- [Covariance and Contravariance \(Visual Basic\)](#)
- [Variance in Delegates \(C#\)](#)
- [Variance in Delegates \(Visual Basic\)](#)

Collections and Data Structures

9/20/2022 • 6 minutes to read • [Edit Online](#)

Similar data can often be handled more efficiently when stored and manipulated as a collection. You can use the [System.Array](#) class or the classes in the [System.Collections](#), [System.Collections.Generic](#), [System.Collections.Concurrent](#), and [System.Collections.Immutable](#) namespaces to add, remove, and modify either individual elements or a range of elements in a collection.

There are two main types of collections; generic collections and non-generic collections. Generic collections are type-safe at compile time. Because of this, generic collections typically offer better performance. Generic collections accept a type parameter when they're constructed. They don't require that you cast to and from the [Object](#) type when you add or remove items from the collection. In addition, most generic collections are supported in Windows Store apps. Non-generic collections store items as [Object](#), require casting, and most aren't supported for Windows Store app development. However, you might see non-generic collections in older code.

In .NET Framework 4 and later versions, the collections in the [System.Collections.Concurrent](#) namespace provide efficient thread-safe operations for accessing collection items from multiple threads. The immutable collection classes in the [System.Collections.Immutable](#) namespace ([NuGet package](#)) are inherently thread-safe because operations are performed on a copy of the original collection, and the original collection can't be modified.

Common collection features

All collections provide methods for adding, removing, or finding items in the collection. In addition, all collections that directly or indirectly implement the [ICollection](#) interface or the [ICollection<T>](#) interface share these features:

- **The ability to enumerate the collection**

.NET collections either implement [System.Collections.IEnumerable](#) or [System.Collections.Generic.IEnumerable<T>](#) to enable the collection to be iterated through. An enumerator can be thought of as a movable pointer to any element in the collection. The [foreach](#), [in](#) statement and the [For Each...Next Statement](#) use the enumerator exposed by the [GetEnumerator](#) method and hide the complexity of manipulating the enumerator. In addition, any collection that implements [System.Collections.Generic.IEnumerable<T>](#) is considered a *queryable type* and can be queried with LINQ. LINQ queries provide a common pattern for accessing data. They're typically more concise and readable than standard [foreach](#) loops and provide filtering, ordering, and grouping capabilities. LINQ queries can also improve performance. For more information, see [LINQ to Objects \(C#\)](#), [LINQ to Objects \(Visual Basic\)](#), [Parallel LINQ \(PLINQ\)](#), [Introduction to LINQ Queries \(C#\)](#), and [Basic Query Operations \(Visual Basic\)](#).

- **The ability to copy the collection contents to an array**

All collections can be copied to an array using the [CopyTo](#) method. However, the order of the elements in the new array is based on the sequence in which the enumerator returns them. The resulting array is always one-dimensional with a lower bound of zero.

In addition, many collection classes contain the following features:

- **Capacity and Count properties**

The capacity of a collection is the number of elements it can contain. The count of a collection is the

number of elements it actually contains. Some collections hide the capacity or the count or both.

Most collections automatically expand in capacity when the current capacity is reached. The memory is reallocated, and the elements are copied from the old collection to the new one. This design reduces the code required to use the collection. However, the performance of the collection might be negatively affected. For example, for [List<T>](#), if [Count](#) is less than [Capacity](#), adding an item is an O(1) operation. If the capacity needs to be increased to accommodate the new element, adding an item becomes an O(n) operation, where n is [Count](#). The best way to avoid poor performance caused by multiple reallocations is to set the initial capacity to be the estimated size of the collection.

A [BitArray](#) is a special case; its capacity is the same as its length, which is the same as its count.

- **A consistent lower bound**

The lower bound of a collection is the index of its first element. All indexed collections in the [System.Collections](#) namespaces have a lower bound of zero, meaning they're 0-indexed. [Array](#) has a lower bound of zero by default, but a different lower bound can be defined when creating an instance of the [Array](#) class using [Array.CreateInstance](#).

- **Synchronization for access from multiple threads** ([System.Collections](#) classes only).

Non-generic collection types in the [System.Collections](#) namespace provide some thread safety with synchronization; typically exposed through the [SyncRoot](#) and [IsSynchronized](#) members. These collections aren't thread-safe by default. If you require scalable and efficient multi-threaded access to a collection, use one of the classes in the [System.Collections.Concurrent](#) namespace or consider using an immutable collection. For more information, see [Thread-Safe Collections](#).

Choose a collection

In general, you should use generic collections. The following table describes some common collection scenarios and the collection classes you can use for those scenarios. If you're new to generic collections, the following table will help you choose the generic collection that works best for your task:

I WANT TO...	GENERIC COLLECTION OPTIONS	NON-GENERIC COLLECTION OPTIONS	THREAD-SAFE OR IMMUTABLE COLLECTION OPTIONS
Store items as key/value pairs for quick look-up by key	Dictionary<TKey,TValue>	Hashtable (A collection of key/value pairs that are organized based on the hash code of the key.)	ConcurrentDictionary<TKey,TValue> ReadOnlyDictionary<TKey,TValue> ImmutableDictionary<TKey,TValue>
Access items by index	List<T>	Array ArrayList	ImmutableList<T> ImmutableArray
Use items first-in-first-out (FIFO)	Queue<T>	Queue	ConcurrentQueue<T> ImmutableQueue<T>
Use data Last-In-First-Out (LIFO)	Stack<T>	Stack	ConcurrentStack<T> ImmutableStack<T>

I WANT TO...	GENERIC COLLECTION OPTIONS	NON-GENERIC COLLECTION OPTIONS	THREAD-SAFE OR IMMUTABLE COLLECTION OPTIONS
Access items sequentially	<code>LinkedList<T></code>	No recommendation	No recommendation
Receive notifications when items are removed or added to the collection. (implements <code>INotifyPropertyChanged</code> and <code>INotifyCollectionChanged</code>)	<code>ObservableCollection<T></code>	No recommendation	No recommendation
A sorted collection	<code>SortedList< TKey, TValue ></code>	<code>SortedList</code>	<code>ImmutableSortedDictionary< TKey, TValue ></code> <code>ImmutableSortedSet<T></code>
A set for mathematical functions	<code>HashSet<T></code> <code>SortedSet<T></code>	No recommendation	<code>ImmutableHashSet<T></code> <code>ImmutableSortedSet<T></code>

Algorithmic complexity of collections

When choosing a [collection class](#), it's worth considering potential tradeoffs in performance. Use the following table to reference how various mutable collection types compare in algorithmic complexity to their corresponding immutable counterparts. Often immutable collection types are less performant but provide immutability - which is often a valid comparative benefit.

MUTABLE	AMORTIZED	WORST CASE	IMMUTABLE	COMPLEXITY
<code>Stack<T>.Push</code>	$O(1)$	$O(n)$	<code>ImmutableStack<T>.Push</code>	$O(1)$
<code>Queue<T>.Enqueue</code>	$O(1)$	$O(n)$	<code>ImmutableQueue<T>.Enqueue</code>	$O(1)$
<code>List<T>.Add</code>	$O(1)$	$O(n)$	<code>ImmutableList<T>.Add</code>	$O(\log n)$
<code>List<T>.Item[Int32]</code>	$O(1)$	$O(1)$	<code>ImmutableList<T>.Item</code>	$O(\log n)$
<code>List<T>.Enumerator</code>	$O(n)$	$O(n)$	<code>ImmutableList<T>.Enumerator</code>	$O(n)$
<code>HashSet<T>.Add</code> , lookup	$O(1)$	$O(n)$	<code>ImmutableHashSet<T>.Add</code>	$O(\log n)$
<code>SortedSet<T>.Add</code>	$O(\log n)$	$O(n)$	<code>ImmutableSortedSet<T>.Add</code>	$O(\log n)$
<code>Dictionary<T>.Add</code>	$O(1)$	$O(n)$	<code>ImmutableDictionary<T>.Add</code>	$O(\log n)$
<code>Dictionary<T></code> lookup	$O(1)$	$O(1) - \text{or strictly } O(n)$	<code>ImmutableDictionary<T>.Lookup</code>	$O(\log n)$
<code>SortedDictionary<T>.Add</code>	$O(\log n)$	$O(n \log n)$	<code>ImmutableSortedDictionary.Add</code>	$O(\log n)$

A `List<T>` can be efficiently enumerated using either a `for` loop or a `foreach` loop. An `ImmutableList<T>`, however, does a poor job inside a `for` loop, due to the $O(\log n)$ time for its indexer. Enumerating an

`ImmutableList<T>` using a `foreach` loop is efficient because `ImmutableList<T>` uses a binary tree to store its data instead of an array like `List<T>` uses. An array can be quickly indexed into, whereas a binary tree must be walked down until the node with the desired index is found.

Additionally, `SortedSet<T>` has the same complexity as `ImmutableSortedSet<T>` because they both use binary trees. The significant difference is that `ImmutableSortedSet<T>` uses an immutable binary tree. Since `ImmutableSortedSet<T>` also offers a `System.Collections.Immutable.ImmutableSortedSet<T>.Builder` class that allows mutation, you can have both immutability and performance.

Related articles

TITLE	DESCRIPTION
Selecting a Collection Class	Describes the different collections and helps you select one for your scenario.
Commonly Used Collection Types	Describes commonly used generic and non-generic collection types such as <code>System.Array</code> , <code>System.Collections.Generic.List<T></code> , and <code>System.Collections.Generic.Dictionary<TKey,TValue></code> .
When to Use Generic Collections	Discusses the use of generic collection types.
Comparisons and Sorts Within Collections	Discusses the use of equality comparisons and sorting comparisons in collections.
Sorted Collection Types	Describes sorted collections performance and characteristics.
Hashtable and Dictionary Collection Types	Describes the features of generic and non-generic hash-based dictionary types.
Thread-Safe Collections	Describes collection types such as <code>System.Collections.Concurrent.BlockingCollection<T></code> and <code>System.Collections.Concurrent.ConcurrentBag<T></code> that support safe and efficient concurrent access from multiple threads.
System.Collections.Immutable	Introduces the immutable collections and provides links to the collection types.

Reference

- [System.Array](#)
- [System.Collections](#)
- [System.Collections.Concurrent](#)
- [System.Collections.Generic](#)
- [System.Collections.Specialized](#)
- [System.Linq](#)
- [System.Collections.Immutable](#)

Selecting a Collection Class

9/20/2022 • 3 minutes to read • [Edit Online](#)

Be sure to choose your collection class carefully. Using the wrong type can restrict your use of the collection.

IMPORTANT

Avoid using the types in the [System.Collections](#) namespace. The generic and concurrent versions of the collections are recommended because of their greater type safety and other improvements.

Consider the following questions:

- Do you need a sequential list where the element is typically discarded after its value is retrieved?
 - If yes, consider using the [Queue](#) class or the [Queue<T>](#) generic class if you need first-in, first-out (FIFO) behavior. Consider using the [Stack](#) class or the [Stack<T>](#) generic class if you need last-in, first-out (LIFO) behavior. For safe access from multiple threads, use the concurrent versions, [ConcurrentQueue<T>](#) and [ConcurrentStack<T>](#). For immutability, consider the immutable versions, [ImmutableQueue<T>](#) and [ImmutableStack<T>](#).
 - If not, consider using the other collections.
- Do you need to access the elements in a certain order, such as FIFO, LIFO, or random?
 - The [Queue](#) class, as well as the [Queue<T>](#), [ConcurrentQueue<T>](#), and [ImmutableQueue<T>](#) generic classes all offer FIFO access. For more information, see [When to Use a Thread-Safe Collection](#).
 - The [Stack](#) class, as well as the [Stack<T>](#), [ConcurrentStack<T>](#), and [ImmutableStack<T>](#) generic classes all offer LIFO access. For more information, see [When to Use a Thread-Safe Collection](#).
 - The [LinkedList<T>](#) generic class allows sequential access either from the head to the tail, or from the tail to the head.
- Do you need to access each element by index?
 - The [ArrayList](#) and [StringCollection](#) classes and the [List<T>](#) generic class offer access to their elements by the zero-based index of the element. For immutability, consider the immutable generic versions, [ImmutableArray<T>](#) and [ImmutableList<T>](#).
 - The [Hashtable](#), [SortedList](#), [ListDictionary](#), and [StringDictionary](#) classes, and the [Dictionary< TKey, TValue >](#) and [SortedDictionary< TKey, TValue >](#) generic classes offer access to their elements by the key of the element. Additionally, there are immutable versions of several corresponding types: [ImmutableHashSet<T>](#), [ImmutableDictionary< TKey, TValue >](#), [ImmutableSortedSet<T>](#), and [ImmutableSortedDictionary< TKey, TValue >](#).
 - The [NameObjectCollectionBase](#) and [NameValuePairCollection](#) classes, and the [KeyedCollection< TKey, TItem >](#) and [SortedList< TKey, TValue >](#) generic classes offer access to their elements by either the zero-based index or the key of the element.
- Will each element contain one value, a combination of one key and one value, or a combination of one key and multiple values?
 - One value: Use any of the collections based on the [IList](#) interface or the [IList<T>](#) generic interface.

For an immutable option, consider the [IImmutableList<T>](#) generic interface.

- One key and one value: Use any of the collections based on the [IDictionary](#) interface or the [IDictionary< TKey, TValue >](#) generic interface. For an immutable option, consider the [IImmutableSet<T>](#) or [IImmutableDictionary< TKey, TValue >](#) generic interfaces.
- One value with embedded key: Use the [KeyedCollection< TKey, TItem >](#) generic class.
- One key and multiple values: Use the [NameValueCollection](#) class.
- Do you need to sort the elements differently from how they were entered?
 - The [Hashtable](#) class sorts its elements by their hash codes.
 - The [SortedList](#) class, and the [SortedList< TKey, TValue >](#) and [SortedDictionary< TKey, TValue >](#) generic classes sort their elements by the key. The sort order is based on the implementation of the [IComparer](#) interface for the [SortedList](#) class and on the implementation of the [IComparer< T >](#) generic interface for the [SortedList< TKey, TValue >](#) and [SortedDictionary< TKey, TValue >](#) generic classes. Of the two generic types, [SortedDictionary< TKey, TValue >](#) offers better performance than [SortedList< TKey, TValue >](#), while [SortedList< TKey, TValue >](#) consumes less memory.
 - [ArrayList](#) provides a [Sort](#) method that takes an [IComparer](#) implementation as a parameter. Its generic counterpart, the [List< T >](#) generic class, provides a [Sort](#) method that takes an implementation of the [IComparer< T >](#) generic interface as a parameter.
- Do you need fast searches and retrieval of information?
 - [ListDictionary](#) is faster than [Hashtable](#) for small collections (10 items or fewer). The [Dictionary< TKey, TValue >](#) generic class provides faster lookup than the [SortedDictionary< TKey, TValue >](#) generic class. The multi-threaded implementation is [ConcurrentDictionary< TKey, TValue >](#). [ConcurrentBag< T >](#) provides fast multi-threaded insertion for unordered data. For more information about both multi-threaded types, see [When to Use a Thread-Safe Collection](#).
- Do you need collections that accept only strings?
 - [StringCollection](#) (based on [IList](#)) and [StringDictionary](#) (based on [IDictionary](#)) are in the [System.Collections.Specialized](#) namespace.
 - In addition, you can use any of the generic collection classes in the [System.Collections.Generic](#) namespace as strongly typed string collections by specifying the [String](#) class for their generic type arguments. For example, you can declare a variable to be of type [List< String >](#) or [Dictionary< String, String >](#).

LINQ to Objects and PLINQ

LINQ to Objects enables developers to use LINQ queries to access in-memory objects as long as the object type implements [IEnumerable](#) or [IEnumerable< T >](#). LINQ queries provide a common pattern for accessing data, are typically more concise and readable than standard `foreach` loops, and provide filtering, ordering, and grouping capabilities. For more information, see [LINQ to Objects \(C#\)](#) and [LINQ to Objects \(Visual Basic\)](#).

PLINQ provides a parallel implementation of LINQ to Objects that can offer faster query execution in many scenarios, through more efficient use of multi-core computers. For more information, see [Parallel LINQ \(PLINQ\)](#).

See also

- [System.Collections](#)
- [System.Collections.Specialized](#)

- [System.Collections.Generic](#)
- [Thread-Safe Collections](#)

Commonly used collection types

9/20/2022 • 2 minutes to read • [Edit Online](#)

Collection types represent different ways to collect data, such as hash tables, queues, stacks, bags, dictionaries, and lists.

All collections are based on the [ICollection](#) or [ICollection<T>](#) interfaces, either directly or indirectly. [IList](#) and [IDictionary](#) and their generic counterparts all derive from these two interfaces.

In collections based on [IList](#) or directly on [ICollection](#), every element contains only a value. These types include:

- [Array](#)
- [ArrayList](#)
- [List<T>](#)
- [Queue](#)
- [ConcurrentQueue<T>](#)
- [Stack](#)
- [ConcurrentStack<T>](#)
- [LinkedList<T>](#)

In collections based on the [IDictionary](#) interface, every element contains both a key and a value. These types include:

- [Hashtable](#)
- [SortedList](#)
- [SortedList< TKey, TValue >](#)
- [Dictionary< TKey, TValue >](#)
- [ConcurrentDictionary< TKey, TValue >](#)

The [KeyedCollection< TKey, TItem >](#) class is unique because it is a list of values with keys embedded within the values. As a result, it behaves both like a list and like a dictionary.

When you need efficient multi-threaded collection access, use the generic collections in the [System.Collections.Concurrent](#) namespace.

The [Queue](#) and [Queue<T>](#) classes provide first-in-first-out lists. The [Stack](#) and [Stack<T>](#) classes provide last-in-first-out lists.

Strong typing

Generic collections are the best solution to strong typing. For example, adding an element of any type other than an [Int32](#) to a [List< Int32 >](#) collection causes a compile-time error. However, if your language does not support generics, the [System.Collections](#) namespace includes abstract base classes that you can extend to create collection classes that are strongly typed. These base classes include:

- [CollectionBase](#)
- [ReadOnlyCollectionBase](#)
- [DictionaryBase](#)

How collections vary

Collections vary in how they store, sort, and compare elements, and how they perform searches.

The [SortedList](#) class and the [SortedList<TKey,TValue>](#) generic class provide sorted versions of the [Hashtable](#) class and the [Dictionary<TKey,TValue>](#) generic class.

All collections use zero-based indexes except [Array](#), which allows arrays that are not zero-based.

You can access the elements of a [SortedList](#) or a [KeyedCollection<TKey,TItem>](#) by either the key or the element's index. You can only access the elements of a [Hashtable](#) or a [Dictionary<TKey,TValue>](#) by the element's key.

Use LINQ with collection types

The LINQ to Objects feature provides a common pattern for accessing in-memory objects of any type that implements [IEnumerable](#) or [IEnumerable<T>](#). LINQ queries have several benefits over standard constructs like `foreach` loops:

- They are concise and easier to understand.
- They can filter, order, and group data.
- They can improve performance.

For more information, see [LINQ to Objects \(C#\)](#), [LINQ to Objects \(Visual Basic\)](#), and [Parallel LINQ \(PLINQ\)](#).

Related topics

TITLE	DESCRIPTION
Collections and Data Structures	Discusses the various collection types available in .NET, including stacks, queues, lists, arrays, and dictionaries.
Hashtable and Dictionary Collection Types	Describes the features of generic and nongeneric hash-based dictionary types.
Sorted Collection Types	Describes classes that provide sorting functionality for lists and sets.
Generics	Describes the generics feature, including the generic collections, delegates, and interfaces provided by .NET. Provides links to feature documentation for C#, Visual Basic, and Visual C++, and to supporting technologies such as reflection.

Reference

[System.Collections](#)

[System.Collections.Generic](#)

[System.Collections.ICollection](#)

[System.Collections.Generic.ICollection<T>](#)

[System.Collections_IList](#)

[System.Collections.Generic.IList<T>](#)

[System.Collections.IDictionary](#)

[System.Collections.Generic.IDictionary<TKey,TValue>](#)

When to use generic collections

9/20/2022 • 3 minutes to read • [Edit Online](#)

Using generic collections gives you the automatic benefit of type safety without having to derive from a base collection type and implement type-specific members. Generic collection types also generally perform better than the corresponding nongeneric collection types (and better than types that are derived from nongeneric base collection types) when the collection elements are value types, because with generics, there's no need to box the elements.

For programs that target .NET Standard 1.0 or later, use the generic collection classes in the [System.Collections.Concurrent](#) namespace when multiple threads might be adding or removing items from the collection concurrently. Additionally, when immutability is desired, consider the generic collection classes in the [System.Collections.Immutable](#) namespace.

The following generic types correspond to existing collection types:

- [List<T>](#) is the generic class that corresponds to [ArrayList](#).
- [Dictionary< TKey, TValue >](#) and [ConcurrentDictionary< TKey, TValue >](#) are the generic classes that correspond to [Hashtable](#).
- [Collection<T>](#) is the generic class that corresponds to [CollectionBase](#). [Collection<T>](#) can be used as a base class, but unlike [CollectionBase](#), it is not abstract, which makes it much easier to use.
- [ReadOnlyCollection<T>](#) is the generic class that corresponds to [ReadOnlyCollectionBase](#). [ReadOnlyCollection<T>](#) is not abstract and has a constructor that makes it easy to expose an existing [List<T>](#) as a read-only collection.
- The [Queue<T>](#), [ConcurrentQueue<T>](#), [ImmutableQueue<T>](#), [ImmutableArray<T>](#), [SortedList< TKey, TValue >](#), and [ImmutableSortedSet< T >](#) generic classes correspond to the respective nongeneric classes with the same names.

Additional Types

Several generic collection types do not have nongeneric counterparts. They include the following:

- [LinkedList<T>](#) is a general-purpose linked list that provides O(1) insertion and removal operations.
- [SortedDictionary< TKey, TValue >](#) is a sorted dictionary with O(log n) insertion and retrieval operations, which makes it a useful alternative to [SortedList< TKey, TValue >](#).
- [KeyedCollection< TKey, TItem >](#) is a hybrid between a list and a dictionary, which provides a way to store objects that contain their own keys.
- [BlockingCollection<T>](#) implements a collection class with bounding and blocking functionality.
- [ConcurrentBag<T>](#) provides fast insertion and removal of unordered elements.

Immutable builders

When you desire immutability functionality in your app, the [System.Collections.Immutable](#) namespace offers generic collection types you can use. All of the immutable collection types offer [Builder](#) classes that can optimize performance when you're performing multiple mutations. The [Builder](#) class batches operations in a mutable state. When all mutations have been completed, call the [ToImmutable](#) method to "freeze" all nodes and create an immutable generic collection, for example, an [ImmutableList<T>](#).

The `Builder` object can be created by calling the nongeneric `CreateBuilder()` method. From a `Builder` instance, you can call `ToImmutable()`. Likewise, from the `Immutable*` collection, you can call `ToBuilder()` to create a builder instance from the generic immutable collection. The following are the various `Builder` types.

- `ImmutableArray<T>.Builder`
- `ImmutableDictionary< TKey, TValue >.Builder`
- `ImmutableHashSet<T>.Builder`
- `ImmutableList<T>.Builder`
- `ImmutableSortedDictionary< TKey, TValue >.Builder`
- `ImmutableSortedSet<T>.Builder`

LINQ to Objects

The LINQ to Objects feature enables you to use LINQ queries to access in-memory objects as long as the object type implements the `System.Collections.IEnumerable` or `System.Collections.Generic.IEnumerable<T>` interface. LINQ queries provide a common pattern for accessing data; are typically more concise and readable than standard `foreach` loops; and provide filtering, ordering, and grouping capabilities. LINQ queries can also improve performance. For more information, see [LINQ to Objects \(C#\)](#), [LINQ to Objects \(Visual Basic\)](#), and [Parallel LINQ \(PLINQ\)](#).

Additional Functionality

Some of the generic types have functionality that is not found in the nongeneric collection types. For example, the `List<T>` class, which corresponds to the nongeneric `ArrayList` class, has a number of methods that accept generic delegates, such as the `Predicate<T>` delegate that allows you to specify methods for searching the list, the `Action<T>` delegate that represents methods that act on each element of the list, and the `Converter<TInput,TOutput>` delegate that lets you define conversions between types.

The `List<T>` class allows you to specify your own `IComparer<T>` generic interface implementations for sorting and searching the list. The `SortedDictionary< TKey, TValue >` and `SortedList< TKey, TValue >` classes also have this capability. In addition, these classes let you specify comparers when the collection is created. In similar fashion, the `Dictionary< TKey, TValue >` and `KeyedCollection< TKey, TItem >` classes let you specify your own equality comparers.

See also

- [Collections and Data Structures](#)
- [Commonly Used Collection Types](#)
- [Generics](#)

Comparisons and sorts within collections

9/20/2022 • 7 minutes to read • [Edit Online](#)

The [System.Collections](#) classes perform comparisons in almost all the processes involved in managing collections, whether searching for the element to remove or returning the value of a key-and-value pair.

Collections typically utilize an equality comparer and/or an ordering comparer. Two constructs are used for comparisons.

Check for equality

Methods such as `Contains`, `IndexOf`, `LastIndexOf`, and `Remove` use an equality comparer for the collection elements. If the collection is generic, then items are compared for equality according to the following guidelines:

- If type T implements the `IEquatable<T>` generic interface, then the equality comparer is the `Equals` method of that interface.
- If type T does not implement `IEquatable<T>`, `Object.Equals` is used.

In addition, some constructor overloads for dictionary collections accept an `IEqualityComparer<T>` implementation, which is used to compare keys for equality. For an example, see the `Dictionary< TKey, TValue >` constructor.

Determine sort order

Methods such as `BinarySearch` and `Sort` use an ordering comparer for the collection elements. The comparisons can be between elements of the collection, or between an element and a specified value. For comparing objects, there is the concept of a `default comparer` and an `explicit comparer`.

The default comparer relies on at least one of the objects being compared to implement the `IComparable` interface. It is a good practice to implement `IComparable` on all classes used as values in a list collection or as keys in a dictionary collection. For a generic collection, equality comparison is determined according to the following:

- If type T implements the `System.IComparable<T>` generic interface, then the default comparer is the `IComparable<T>.CompareTo(T)` method of that interface
- If type T implements the non-generic `System.IComparable` interface, then the default comparer is the `IComparable.CompareTo(Object)` method of that interface.
- If type T doesn't implement either interface, then there is no default comparer, and a comparer or comparison delegate must be provided explicitly.

To provide explicit comparisons, some methods accept an `IComparer` implementation as a parameter. For example, the `List<T>.Sort` method accepts an `System.Collections.Generic.IComparer<T>` implementation.

The current culture setting of the system can affect the comparisons and sorts within a collection. By default, the comparisons and sorts in the `Collections` classes are culture-sensitive. To ignore the culture setting and therefore obtain consistent comparison and sorting results, use the `InvariantCulture` with member overloads that accept a `CultureInfo`. For more information, see [Perform culture-insensitive string operations in collections](#) and [Perform culture-insensitive string operations in arrays](#).

Equality and sort example

The following code demonstrates an implementation of `IEquatable<T>` and `IComparable<T>` on a simple business object. In addition, when the object is stored in a list and sorted, you will see that calling the `Sort()` method results in the use of the default comparer for the `Part` type, and the `Sort(Comparison<T>)` method implemented by using an anonymous method.

```
using System;
using System.Collections.Generic;

// Simple business object. A PartId is used to identify the
// type of part but the part name can change.
public class Part : IEquatable<Part>, IComparable<Part>
{
    public string PartName { get; set; }

    public int PartId { get; set; }

    public override string ToString() =>
        $"ID: {PartId} Name: {PartName}";

    public override bool Equals(object obj) =>
        (obj is Part part)
            ? Equals(part)
            : false;

    public int SortByNameAscending(string name1, string name2) =>
        name1?.CompareTo(name2) ?? 1;

    // Default comparer for Part type.
    // A null value means that this object is greater.
    public int CompareTo(Part comparePart) =>
        comparePart == null ? 1 : PartId.CompareTo(comparePart.PartId);

    public override int GetHashCode() => PartId;

    public bool Equals(Part other) =>
        other is null ? false : PartId.Equals(other.PartId);

    // Should also override == and != operators.
}

public class Example
{
    public static void Main()
    {
        // Create a list of parts.
        var parts = new List<Part>
        {
            // Add parts to the list.
            new Part { PartName = "regular seat", PartId = 1434 },
            new Part { PartName = "crank arm", PartId = 1234 },
            new Part { PartName = "shift lever", PartId = 1634 },
            // Name intentionally left null.
            new Part { PartId = 1334 },
            new Part { PartName = "banana seat", PartId = 1444 },
            new Part { PartName = "cassette", PartId = 1534 }
        };

        // Write out the parts in the list. This will call the overridden
        // ToString method in the Part class.
        Console.WriteLine("\nBefore sort:");
        parts.ForEach(Console.WriteLine);

        // Call Sort on the list. This will use the
        // default comparer, which is the Compare method
    }
}
```

```

// implemented on Part.
parts.Sort();

Console.WriteLine("\nAfter sort by part number:");
parts.ForEach(Console.WriteLine);

// This shows calling the Sort(Comparison<T> comparison) overload using
// a lambda expression as the Comparison<T> delegate.
// This method treats null as the lesser of two values.
parts.Sort((Part x, Part y) =>
    x.PartName == null && y.PartName == null
        ? 0
        : x.PartName == null
            ? -1
            : y.PartName == null
                ? 1
                : x.PartName.CompareTo(y.PartName));

Console.WriteLine("\nAfter sort by name:");
parts.ForEach(Console.WriteLine);

/*
Before sort:
ID: 1434  Name: regular seat
ID: 1234  Name: crank arm
ID: 1634  Name: shift lever
ID: 1334  Name:
ID: 1444  Name: banana seat
ID: 1534  Name: cassette

After sort by part number:
ID: 1234  Name: crank arm
ID: 1334  Name:
ID: 1434  Name: regular seat
ID: 1444  Name: banana seat
ID: 1534  Name: cassette
ID: 1634  Name: shift lever

After sort by name:
ID: 1334  Name:
ID: 1444  Name: banana seat
ID: 1534  Name: cassette
ID: 1234  Name: crank arm
ID: 1434  Name: regular seat
ID: 1634  Name: shift lever

*/
}
}

```

```

Imports System.Collections.Generic

' Simple business object. A PartId is used to identify the type of part
' but the part name can change.
Public Class Part
    Implements IComparable(Of Part)
    Implements IComparable(Of Part)
    Public Property PartName() As String
        Get
            Return m_PartName
        End Get
        Set(value As String)
            m_PartName = Value
        End Set
    End Property
    Private m_PartName As String

```

```

    Public Property PartId() As Integer
        Get
            Return m_PartId
        End Get
        Set(value As Integer)
            m_PartId = Value
        End Set
    End Property
    Private m_PartId As Integer

    Public Overrides Function ToString() As String
        Return "ID: " & PartId & " Name: " & PartName
    End Function

    Public Overrides Function Equals(obj As Object) As Boolean
        If obj Is Nothing Then
            Return False
        End If
        Dim objAsPart As Part = TryCast(obj, Part)
        If objAsPart Is Nothing Then
            Return False
        Else
            Return Equals(objAsPart)
        End If
    End Function

    Public Function SortByNameAscending(name1 As String, name2 As String) As Integer
        Return name1.CompareTo(name2)
    End Function

    ' Default comparer for Part.
    Public Function CompareTo(comparePart As Part) As Integer _
        Implements IComparable(Of ListSortVB.Part).CompareTo
        ' A null value means that this object is greater.
        If comparePart Is Nothing Then
            Return 1
        Else

            Return Me.PartId.CompareTo(comparePart.PartId)
        End If
    End Function
    Public Overrides Function GetHashCode() As Integer
        Return PartId
    End Function
    Public Overloads Function Equals(other As Part) As Boolean Implements IEquatable(Of
ListSortVB.Part).Equals
        If other Is Nothing Then
            Return False
        End If
        Return (Me.PartId.Equals(other.PartId))
    End Function
    ' Should also override == and != operators.

End Class
Public Class Example
    Public Shared Sub Main()
        ' Create a list of parts.
        Dim parts As New List(Of Part)()

        ' Add parts to the list.
        parts.Add(New Part() With {
            .PartName = "regular seat",
            .PartId = 1434
        })
        parts.Add(New Part() With {
            .PartName = "crank arm",
            .PartId = 1234
        })
    End Sub
End Class

```

```

    })

parts.Add(New Part() With {
    .PartName = "shift lever", _
    .PartId = 1634 _
})

' Name intentionally left null.
parts.Add(New Part() With {
    .PartId = 1334 _
})
parts.Add(New Part() With {
    .PartName = "banana seat", _
    .PartId = 1444 _
})
parts.Add(New Part() With {
    .PartName = "cassette", _
    .PartId = 1534 _
})

' Write out the parts in the list. This will call the overridden
' ToString method in the Part class.
Console.WriteLine(vbLf & "Before sort:")
For Each aPart As Part In parts
    Console.WriteLine(aPart)
Next

' Call Sort on the list. This will use the
' default comparer, which is the Compare method
' implemented on Part.
parts.Sort()

Console.WriteLine(vbLf & "After sort by part number:")
For Each aPart As Part In parts
    Console.WriteLine(aPart)
Next

' This shows calling the Sort(Comparison(T) overload using
' an anonymous delegate method.
' This method treats null as the lesser of two values.
parts.Sort(Function(x As Part, y As Part)
    If x.PartName Is Nothing AndAlso y.PartName Is Nothing Then
        Return 0
    ElseIf x.PartName Is Nothing Then
        Return -1
    ElseIf y.PartName Is Nothing Then
        Return 1
    Else
        Return x.PartName.CompareTo(y.PartName)
    End If
End Function)

Console.WriteLine(vbLf & "After sort by name:")
For Each aPart As Part In parts
    Console.WriteLine(aPart)
Next

'

'

' Before sort:
' ID: 1434  Name: regular seat
' ID: 1234  Name: crank arm
' ID: 1634  Name: shift lever
' ID: 1334  Name:
' ID: 1444  Name: banana seat

```

```
'           ID: 1534  Name: cassette

'           After sort by part number:
'           ID: 1234  Name: crank arm
'           ID: 1334  Name:
'           ID: 1434  Name: regular seat
'           ID: 1444  Name: banana seat
'           ID: 1534  Name: cassette
'           ID: 1634  Name: shift lever

'           After sort by name:
'           ID: 1334  Name:
'           ID: 1444  Name: banana seat
'           ID: 1534  Name: cassette
'           ID: 1234  Name: crank arm
'           ID: 1434  Name: regular seat
'           ID: 1634  Name: shift lever

End Sub
End Class
```

See also

- [IComparer](#)
- [IEquatable<T>](#)
- [IComparer<T>](#)
- [IComparable](#)
- [IComparable<T>](#)

Sorted Collection Types

9/20/2022 • 2 minutes to read • [Edit Online](#)

The [System.Collections.SortedList](#) class, the [System.Collections.Generic.SortedList<TKey,TValue>](#) generic class, and the [System.Collections.Generic.SortedDictionary<TKey,TValue>](#) generic class are similar to the [Hashtable](#) class and the [Dictionary<TKey,TValue>](#) generic class in that they implement the [IDictionary](#) interface, but they maintain their elements in sort order by key, and they do not have the O(1) insertion and retrieval characteristic of hash tables. The three classes have several features in common:

- All three classes implement the [System.Collections.IDictionary](#) interface. The two generic classes also implement the [System.Collections.Generic.IDictionary<TKey,TValue>](#) generic interface.
- Each element is a key/value pair for enumeration purposes.

NOTE

The nongeneric [SortedList](#) class returns [DictionaryEntry](#) objects when enumerated, although the two generic types return [KeyValuePair<TKey,TValue>](#) objects.

- Elements are sorted according to a [System.Collections.IComparer](#) implementation (for nongeneric [SortedList](#)) or a [System.Collections.Generic.IComparer<T>](#) implementation (for the two generic classes).
- Each class provides properties that return collections containing only the keys or only the values.

The following table lists some of the differences between the two sorted list classes and the [SortedDictionary<TKey,TValue>](#) class.

SORTEDLIST NONGENERIC CLASS AND SORTEDLIST<TKEY,TVALUE> GENERIC CLASS	SORTEDDICTIONARY<TKEY,TVALUE> GENERIC CLASS
The properties that return keys and values are indexed, allowing efficient indexed retrieval.	No indexed retrieval.
Retrieval is O(log n).	Retrieval is O(log n).
Insertion and removal are generally O(n); however, insertion is O(log n) for data that are already in sort order, so that each element is added to the end of the list. (This assumes that a resize is not required.)	Insertion and removal are O(log n).
Uses less memory than a SortedDictionary<TKey,TValue> .	Uses more memory than the SortedList nongeneric class and the SortedDictionary<TKey,TValue> generic class.

For sorted lists or dictionaries that must be accessible concurrently from multiple threads, you can add sorting logic to a class that derives from [ConcurrentDictionary<TKey,TValue>](#). When considering immutability, the following corresponding immutable types follow similar sorting semantics: [ImmutableSortedSet<T>](#) and [ImmutableSortedDictionary<TKey,TValue>](#).

NOTE

For values that contain their own keys (for example, employee records that contain an employee ID number), you can create a keyed collection that has some characteristics of a list and some characteristics of a dictionary by deriving from the [KeyedCollection<TKey,TItem>](#) generic class.

Starting with .NET Framework 4, the [SortedSet<T>](#) class provides a self-balancing tree that maintains data in sorted order after insertions, deletions, and searches. This class and the [HashSet<T>](#) class implement the [ISet<T>](#) interface.

See also

- [System.Collections.IDictionary](#)
- [System.Collections.Generic.IDictionary<TKey,TValue>](#)
- [ConcurrentDictionary<TKey,TValue>](#)
- [Commonly Used Collection Types](#)

Hashtable and Dictionary Collection Types

9/20/2022 • 2 minutes to read • [Edit Online](#)

The [System.Collections.Hashtable](#) class, and the [System.Collections.Generic.Dictionary<TKey,TValue>](#) and [System.Collections.Concurrent.ConcurrentDictionary<TKey,TValue>](#) generic classes, implement the [System.Collections.IDictionary](#) interface. The [Dictionary<TKey,TValue>](#) generic class also implements the [IDictionary<TKey,TValue>](#) generic interface. Therefore, each element in these collections is a key-and-value pair.

A [Hashtable](#) object consists of buckets that contain the elements of the collection. A bucket is a virtual subgroup of elements within the [Hashtable](#), which makes searching and retrieving easier and faster than in most collections. Each bucket is associated with a hash code, which is generated using a hash function and is based on the key of the element.

The generic [HashSet<T>](#) class is an unordered collection for containing unique elements.

A hash function is an algorithm that returns a numeric hash code based on a key. The key is the value of some property of the object being stored. A hash function must always return the same hash code for the same key. It is possible for a hash function to generate the same hash code for two different keys, but a hash function that generates a unique hash code for each unique key results in better performance when retrieving elements from the hash table.

Each object that is used as an element in a [Hashtable](#) must be able to generate a hash code for itself by using an implementation of the [GetHashCode](#) method. However, you can also specify a hash function for all elements in a [Hashtable](#) by using a [Hashtable](#) constructor that accepts an [IHashCodeProvider](#) implementation as one of its parameters.

When an object is added to a [Hashtable](#), it is stored in the bucket that is associated with the hash code that matches the object's hash code. When a value is being searched for in the [Hashtable](#), the hash code is generated for that value, and the bucket associated with that hash code is searched.

For example, a hash function for a string might take the ASCII codes of each character in the string and add them together to generate a hash code. The string "picnic" would have a hash code that is different from the hash code for the string "basket"; therefore, the strings "picnic" and "basket" would be in different buckets. In contrast, "stressed" and "desserts" would have the same hash code and would be in the same bucket.

The [Dictionary<TKey,TValue>](#) and [ConcurrentDictionary<TKey,TValue>](#) classes have the same functionality as the [Hashtable](#) class. A [Dictionary<TKey,TValue>](#) of a specific type (other than [Object](#)) provides better performance than a [Hashtable](#) for value types. This is because the elements of [Hashtable](#) are of type [Object](#); therefore, boxing and unboxing typically occur when you store or retrieve a value type. The [ConcurrentDictionary<TKey,TValue>](#) class should be used when multiple threads might be accessing the collection simultaneously.

See also

- [Hashtable](#)
- [IDictionary](#)
- [IHashCodeProvider](#)
- [Dictionary<TKey,TValue>](#)
- [System.Collections.Generic.IDictionary<TKey,TValue>](#)
- [System.Collections.Concurrent.ConcurrentDictionary<TKey,TValue>](#)
- [Commonly Used Collection Types](#)

Thread-Safe Collections

9/20/2022 • 3 minutes to read • [Edit Online](#)

The .NET Framework 4 introduces the [System.Collections.Concurrent](#) namespace, which includes several collection classes that are both thread-safe and scalable. Multiple threads can safely and efficiently add or remove items from these collections, without requiring additional synchronization in user code. When you write new code, use the concurrent collection classes whenever multiple threads will write to the collection concurrently. If you are only reading from a shared collection, then you can use the classes in the [System.Collections.Generic](#) namespace. We recommend that you do not use 1.0 collection classes unless you are required to target the .NET Framework 1.1 or earlier runtime.

Thread Synchronization in the .NET Framework 1.0 and 2.0 Collections

The collections introduced in the .NET Framework 1.0 are found in the [System.Collections](#) namespace. These collections, which include the commonly used [ArrayList](#) and [Hashtable](#), provide some thread-safety through the [Synchronized](#) property, which returns a thread-safe wrapper around the collection. The wrapper works by locking the entire collection on every add or remove operation. Therefore, each thread that is attempting to access the collection must wait for its turn to take the one lock. This is not scalable and can cause significant performance degradation for large collections. Also, the design is not completely protected from race conditions. For more information, see [Synchronization in Generic Collections](#).

The collection classes introduced in the .NET Framework 2.0 are found in the [System.Collections.Generic](#) namespace. These include [List<T>](#), [Dictionary< TKey, TValue >](#), and so on. These classes provide improved type safety and performance compared to the .NET Framework 1.0 classes. However, the .NET Framework 2.0 collection classes do not provide any thread synchronization; user code must provide all synchronization when items are added or removed on multiple threads concurrently.

We recommend the concurrent collections classes in the .NET Framework 4 because they provide not only the type safety of the .NET Framework 2.0 collection classes, but also more efficient and more complete thread safety than the .NET Framework 1.0 collections provide.

Fine-Grained Locking and Lock-Free Mechanisms

Some of the concurrent collection types use lightweight synchronization mechanisms such as [SpinLock](#), [SpinWait](#), [SemaphoreSlim](#), and [CountdownEvent](#), which are new in the .NET Framework 4. These synchronization types typically use *busy spinning* for brief periods before they put the thread into a true Wait state. When wait times are expected to be very short, spinning is far less computationally expensive than waiting, which involves an expensive kernel transition. For collection classes that use spinning, this efficiency means that multiple threads can add and remove items at a very high rate. For more information about spinning vs. blocking, see [SpinLock](#) and [SpinWait](#).

The [ConcurrentQueue<T>](#) and [ConcurrentStack<T>](#) classes do not use locks at all. Instead, they rely on [Interlocked](#) operations to achieve thread-safety.

NOTE

Because the concurrent collections classes support [ICollection](#), they provide implementations for the [IsSynchronized](#) and [SyncRoot](#) properties, even though these properties are irrelevant. [IsSynchronized](#) always returns [false](#) and [SyncRoot](#) is always [null](#) ([Nothing](#) in Visual Basic).

The following table lists the collection types in the [System.Collections.Concurrent](#) namespace.

TYPE	DESCRIPTION
BlockingCollection<T>	Provides bounding and blocking functionality for any type that implements IProducerConsumerCollection<T> . For more information, see BlockingCollection Overview .
ConcurrentDictionary< TKey, TValue >	Thread-safe implementation of a dictionary of key-value pairs.
ConcurrentQueue<T>	Thread-safe implementation of a FIFO (first-in, first-out) queue.
ConcurrentStack<T>	Thread-safe implementation of a LIFO (last-in, first-out) stack.
ConcurrentBag<T>	Thread-safe implementation of an unordered collection of elements.
IProducerConsumerCollection<T>	The interface that a type must implement to be used in a BlockingCollection .

Related Topics

TITLE	DESCRIPTION
BlockingCollection Overview	Describes the functionality provided by the BlockingCollection<T> type.
How to: Add and Remove Items from a ConcurrentDictionary	Describes how to add and remove elements from a ConcurrentDictionary< TKey, TValue >
How to: Add and Take Items Individually from a BlockingCollection	Describes how to add and retrieve items from a blocking collection without using the read-only enumerator.
How to: Add Bounding and Blocking Functionality to a Collection	Describes how to use any collection class as the underlying storage mechanism for an IProducerConsumerCollection<T> collection.
How to: Use ForEach to Remove Items in a BlockingCollection	Describes how to use <code>foreach</code> (<code>For Each</code> in Visual Basic) to remove all items in a blocking collection.
How to: Use Arrays of Blocking Collections in a Pipeline	Describes how to use multiple blocking collections at the same time to implement a pipeline.
How to: Create an Object Pool by Using a ConcurrentBag	Shows how to use a concurrent bag to improve performance in scenarios where you can reuse objects instead of continually creating new ones.

Reference

[System.Collections.Concurrent](#)

Delegates and lambdas

9/20/2022 • 4 minutes to read • [Edit Online](#)

A delegate defines a type that represents references to methods that have a particular parameter list and return type. A method (static or instance) whose parameter list and return type match can be assigned to a variable of that type, then called directly (with the appropriate arguments) or passed as an argument itself to another method and then called. The following example demonstrates delegate use.

```
using System;
using System.Linq;

public class Program
{
    public delegate string Reverse(string s);

    static string ReverseString(string s)
    {
        return new string(s.Reverse().ToArray());
    }

    static void Main(string[] args)
    {
        Reverse rev = ReverseString;

        Console.WriteLine(rev("a string"));
    }
}
```

- The `public delegate string Reverse(string s);` line creates a delegate type of a method that takes a string parameter and then returns a string parameter.
- The `static string ReverseString(string s)` method, which has the exact same parameter list and return type as the defined delegate type, implements the delegate.
- The `Reverse rev = ReverseString;` line shows that you can assign a method to a variable of the corresponding delegate type.
- The `Console.WriteLine(rev("a string"));` line demonstrates how to use a variable of a delegate type to invoke the delegate.

In order to streamline the development process, .NET includes a set of delegate types that programmers can reuse and not have to create new types. These types are `Func<>`, `Action<>` and `Predicate<>`, and they can be used without having to define new delegate types. There are some differences between the three types that have to do with the way they were intended to be used:

- `Action<>` is used when there is a need to perform an action using the arguments of the delegate. The method it encapsulates does not return a value.
- `Func<>` is used usually when you have a transformation on hand, that is, you need to transform the arguments of the delegate into a different result. Projections are a good example. The method it encapsulates returns a specified value.
- `Predicate<>` is used when you need to determine if the argument satisfies the condition of the delegate. It can also be written as a `Func<T, bool>`, which means the method returns a boolean value.

We can now take our example above and rewrite it using the `Func<>` delegate instead of a custom type. The program will continue running exactly the same.

```

using System;
using System.Linq;

public class Program
{
    static string ReverseString(string s)
    {
        return new string(s.Reverse().ToArray());
    }

    static void Main(string[] args)
    {
        Func<string, string> rev = ReverseString;

        Console.WriteLine(rev("a string"));
    }
}

```

For this simple example, having a method defined outside of the `Main` method seems a bit superfluous. .NET Framework 2.0 introduced the concept of *anonymous delegates*, which let you create "inline" delegates without having to specify any additional type or method.

In the following example, an anonymous delegate filters a list to just the even numbers and then prints them to the console.

```

using System;
using System.Collections.Generic;

public class Program
{
    public static void Main(string[] args)
    {
        List<int> list = new List<int>();

        for (int i = 1; i <= 100; i++)
        {
            list.Add(i);
        }

        List<int> result = list.FindAll(
            delegate (int no)
            {
                return (no % 2 == 0);
            }
        );

        foreach (var item in result)
        {
            Console.WriteLine(item);
        }
    }
}

```

As you can see, the body of the delegate is just a set of expressions, as any other delegate. But instead of it being a separate definition, we've introduced it *ad hoc* in our call to the `List<T>.FindAll` method.

However, even with this approach, there is still much code that we can throw away. This is where *lambda expressions* come into play. Lambda expressions, or just "lambdas" for short, were introduced in C# 3.0 as one of the core building blocks of Language Integrated Query (LINQ). They are just a more convenient syntax for using delegates. They declare a parameter list and method body, but don't have a formal identity of their own, unless they are assigned to a delegate. Unlike delegates, they can be directly assigned as the right-hand side of event registration or in various LINQ clauses and methods.

Since a lambda expression is just another way of specifying a delegate, we should be able to rewrite the above sample to use a lambda expression instead of an anonymous delegate.

```
using System;
using System.Collections.Generic;

public class Program
{
    public static void Main(string[] args)
    {
        List<int> list = new List<int>();

        for (int i = 1; i <= 100; i++)
        {
            list.Add(i);
        }

        List<int> result = list.FindAll(i => i % 2 == 0);

        foreach (var item in result)
        {
            Console.WriteLine(item);
        }
    }
}
```

In the preceding example, the lambda expression used is `i => i % 2 == 0`. Again, it is just a convenient syntax for using delegates. What happens under the covers is similar to what happens with the anonymous delegate.

Again, lambdas are just delegates, which means that they can be used as an event handler without any problems, as the following code snippet illustrates.

```
public MainWindow()
{
    InitializeComponent();

    Loaded += (o, e) =>
    {
        this.Title = "Loaded";
    };
}
```

The `+=` operator in this context is used to subscribe to an [event](#). For more information, see [How to subscribe to and unsubscribe from events](#).

Further reading and resources

- [Delegates](#)
- [Lambda expressions](#)

Handle and raise events

9/20/2022 • 7 minutes to read • [Edit Online](#)

Events in .NET are based on the delegate model. The delegate model follows the [observer design pattern](#), which enables a subscriber to register with and receive notifications from a provider. An event sender pushes a notification that an event has happened, and an event receiver receives that notification and defines a response to it. This article describes the major components of the delegate model, how to consume events in applications, and how to implement events in your code.

Events

An event is a message sent by an object to signal the occurrence of an action. The action can be caused by user interaction, such as a button click, or it can result from some other program logic, such as changing a property's value. The object that raises the event is called the *event sender*. The event sender doesn't know which object or method will receive (handle) the events it raises. The event is typically a member of the event sender; for example, the [Click](#) event is a member of the [Button](#) class, and the [PropertyChanged](#) event is a member of the class that implements the [INotifyPropertyChanged](#) interface.

To define an event, you use the C# `event` or the Visual Basic `Event` keyword in the signature of your event class, and specify the type of delegate for the event. Delegates are described in the next section.

Typically, to raise an event, you add a method that is marked as `protected` and `virtual` (in C#) or `Protected` and `Overridable` (in Visual Basic). Name this method `on EventName`; for example, `OnDataReceived`. The method should take one parameter that specifies an event data object, which is an object of type [EventArgs](#) or a derived type. You provide this method to enable derived classes to override the logic for raising the event. A derived class should always call the `on EventName` method of the base class to ensure that registered delegates receive the event.

The following example shows how to declare an event named `ThresholdReached`. The event is associated with the [EventHandler](#) delegate and raised in a method named `OnThresholdReached`.

```
class Counter
{
    public event EventHandler ThresholdReached;

    protected virtual void OnThresholdReached(EventArgs e)
    {
        EventHandler handler = ThresholdReached;
        handler?.Invoke(this, e);
    }

    // provide remaining implementation for the class
}
```

```

Public Class Counter
    Public Event ThresholdReached As EventHandler

    Protected Overridable Sub OnThresholdReached(e As EventArgs)
        RaiseEvent ThresholdReached(Me, e)
    End Sub

    ' provide remaining implementation for the class
End Class

```

Delegates

A delegate is a type that holds a reference to a method. A delegate is declared with a signature that shows the return type and parameters for the methods it references, and it can hold references only to methods that match its signature. A delegate is thus equivalent to a type-safe function pointer or a callback. A delegate declaration is sufficient to define a delegate class.

Delegates have many uses in .NET. In the context of events, a delegate is an intermediary (or pointer-like mechanism) between the event source and the code that handles the event. You associate a delegate with an event by including the delegate type in the event declaration, as shown in the example in the previous section. For more information about delegates, see the [Delegate](#) class.

.NET provides the [EventHandler](#) and [EventHandler<TEventArgs>](#) delegates to support most event scenarios. Use the [EventHandler](#) delegate for all events that do not include event data. Use the [EventHandler<TEventArgs>](#) delegate for events that include data about the event. These delegates have no return type value and take two parameters (an object for the source of the event, and an object for event data).

Delegates are [multicast](#), which means that they can hold references to more than one event-handling method. For details, see the [Delegate](#) reference page. Delegates provide flexibility and fine-grained control in event handling. A delegate acts as an event dispatcher for the class that raises the event by maintaining a list of registered event handlers for the event.

For scenarios where the [EventHandler](#) and [EventHandler<TEventArgs>](#) delegates do not work, you can define a delegate. Scenarios that require you to define a delegate are very rare, such as when you must work with code that does not recognize generics. You mark a delegate with the C# `delegate` and Visual Basic `Delegate` keyword in the declaration. The following example shows how to declare a delegate named `ThresholdReachedEventHandler`.

```
public delegate void ThresholdReachedEventHandler(object sender, ThresholdReachedEventArgs e);
```

```
Public Delegate Sub ThresholdReachedEventHandler(sender As Object, e As ThresholdReachedEventArgs)
```

Event data

Data that is associated with an event can be provided through an event data class. .NET provides many event data classes that you can use in your applications. For example, the [SerialDataReceivedEventArgs](#) class is the event data class for the [SerialPort.DataReceived](#) event. .NET follows a naming pattern of ending all event data classes with `EventArgs`. You determine which event data class is associated with an event by looking at the delegate for the event. For example, the [SerialDataReceivedEventHandler](#) delegate includes the [SerialDataReceivedEventArgs](#) class as one of its parameters.

The [EventArgs](#) class is the base type for all event data classes. [EventArgs](#) is also the class you use when an event does not have any data associated with it. When you create an event that is only meant to notify other classes

that something happened and does not need to pass any data, include the `EventArgs` class as the second parameter in the delegate. You can pass the `EventArgs.Empty` value when no data is provided. The `EventHandler` delegate includes the `EventArgs` class as a parameter.

When you want to create a customized event data class, create a class that derives from `EventArgs`, and then provide any members needed to pass data that is related to the event. Typically, you should use the same naming pattern as .NET and end your event data class name with `EventArgs`.

The following example shows an event data class named `ThresholdReachedEventArgs`. It contains properties that are specific to the event being raised.

```
public class ThresholdReachedEventArgs : EventArgs
{
    public int Threshold { get; set; }
    public DateTime TimeReached { get; set; }
}
```

```
Public Class ThresholdReachedEventArgs
    Inherits EventArgs

    Public Property Threshold As Integer
    Public Property TimeReached As DateTime
End Class
```

Event handlers

To respond to an event, you define an event handler method in the event receiver. This method must match the signature of the delegate for the event you are handling. In the event handler, you perform the actions that are required when the event is raised, such as collecting user input after the user clicks a button. To receive notifications when the event occurs, your event handler method must subscribe to the event.

The following example shows an event handler method named `c_ThresholdReached` that matches the signature for the `EventHandler` delegate. The method subscribes to the `ThresholdReached` event.

```
class Program
{
    static void Main()
    {
        var c = new Counter();
        c.ThresholdReached += c_ThresholdReached;

        // provide remaining implementation for the class
    }

    static void c_ThresholdReached(object sender, EventArgs e)
    {
        Console.WriteLine("The threshold was reached.");
    }
}
```

```

Module Module1

Sub Main()
    Dim c As New Counter()
    AddHandler c.ThresholdReached, AddressOf c_ThresholdReached

    ' provide remaining implementation for the class
End Sub

Sub c_ThresholdReached(sender As Object, e As EventArgs)
    Console.WriteLine("The threshold was reached.")
End Sub
End Module

```

Static and dynamic event handlers

.NET allows subscribers to register for event notifications either statically or dynamically. Static event handlers are in effect for the entire life of the class whose events they handle. Dynamic event handlers are explicitly activated and deactivated during program execution, usually in response to some conditional program logic. For example, they can be used if event notifications are needed only under certain conditions or if an application provides multiple event handlers and run-time conditions define the appropriate one to use. The example in the previous section shows how to dynamically add an event handler. For more information, see [Events](#) (in Visual Basic) and [Events](#) (in C#).

Raising multiple events

If your class raises multiple events, the compiler generates one field per event delegate instance. If the number of events is large, the storage cost of one field per delegate may not be acceptable. For those situations, .NET provides event properties that you can use with another data structure of your choice to store event delegates.

Event properties consist of event declarations accompanied by event accessors. Event accessors are methods that you define to add or remove event delegate instances from the storage data structure. Note that event properties are slower than event fields, because each event delegate must be retrieved before it can be invoked. The trade-off is between memory and speed. If your class defines many events that are infrequently raised, you will want to implement event properties. For more information, see [How to: Handle Multiple Events Using Event Properties](#).

Related articles

TITLE	DESCRIPTION
How to: Raise and Consume Events	Contains examples of raising and consuming events.
How to: Handle Multiple Events Using Event Properties	Shows how to use event properties to handle multiple events.
Observer Design Pattern	Describes the design pattern that enables a subscriber to register with, and receive notifications from, a provider.

See also

- [EventHandler](#)
- [EventHandler<TEventArgs>](#)
- [EventArgs](#)

- [Delegate](#)
- [Events \(Visual Basic\)](#)
- [Events \(C# Programming Guide\)](#)
- [Events and routed events overview \(UWP apps\)](#)
- [Events in Windows Store 8.x apps](#)

How to: Raise and Consume Events

9/20/2022 • 5 minutes to read • [Edit Online](#)

The examples in this topic show how to work with events. They include examples of the `EventHandler` delegate, the `EventHandler<TEventArgs>` delegate, and a custom delegate, to illustrate events with and without data.

The examples use concepts described in the [Events](#) article.

Example 1

The first example shows how to raise and consume an event that doesn't have data. It contains a class named `Counter` that has an event named `ThresholdReached`. This event is raised when a counter value equals or exceeds a threshold value. The `EventHandler` delegate is associated with the event, because no event data is provided.

```
using System;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            Counter c = new Counter(new Random().Next(10));
            c.ThresholdReached += c_ThresholdReached;

            Console.WriteLine("press 'a' key to increase total");
            while (Console.ReadKey(true).KeyChar == 'a')
            {
                Console.WriteLine("adding one");
                c.Add(1);
            }
        }

        static void c_ThresholdReached(object sender, EventArgs e)
        {
            Console.WriteLine("The threshold was reached.");
            Environment.Exit(0);
        }
    }

    class Counter
    {
        private int threshold;
        private int total;

        public Counter(int passedThreshold)
        {
            threshold = passedThreshold;
        }

        public void Add(int x)
        {
            total += x;
            if (total >= threshold)
            {
                ThresholdReached?.Invoke(this, EventArgs.Empty);
            }
        }

        public event EventHandler ThresholdReached;
    }
}
```

```

Module Module1

Sub Main()
    Dim c As Counter = New Counter(New Random().Next(10))
    AddHandler c.ThresholdReached, AddressOf c_ThresholdReached

    Console.WriteLine("press 'a' key to increase total")
    While Console.ReadKey(True).KeyChar = "a"
        Console.WriteLine("adding one")
        c.Add(1)
    End While
End Sub

Sub c_ThresholdReached(sender As Object, e As EventArgs)
    Console.WriteLine("The threshold was reached.")
    Environment.Exit(0)
End Sub
End Module

Class Counter
    Private threshold As Integer
    Private total As Integer

    Public Sub New(passedThreshold As Integer)
        threshold = passedThreshold
    End Sub

    Public Sub Add(x As Integer)
        total = total + x
        If (total >= threshold) Then
            ThresholdReached?.Invoke(this, EventArgs.Empty)
        End If
    End Sub

    Public Event ThresholdReached As EventHandler
End Class

```

Example 2

The next example shows how to raise and consume an event that provides data. The [EventHandler<TEventArgs>](#) delegate is associated with the event, and an instance of a custom event data object is provided.

```

using namespace System;

public ref class ThresholdReachedEventArgs : public EventArgs
{
public:
    property int Threshold;
    property DateTime TimeReached;
};

public ref class Counter
{
private:
    int threshold;
    int total;

public:
    Counter() {};

    Counter(int passedThreshold)
    {
        threshold = passedThreshold;
    }

    void Add(int x)
    {
        total += x;
        if (total >= threshold) {
            ThresholdReachedEventArgs^ args = gcnew ThresholdReachedEventArgs();
            args->Threshold = threshold;
            args->TimeReached = DateTime::Now;
            OnThresholdReached(args);
        }
    }
}

event EventHandler<ThresholdReachedEventArgs^>^ ThresholdReached;

protected:
    virtual void OnThresholdReached(ThresholdReachedEventArgs^ e)
    {
        ThresholdReached(this, e);
    }
};

public ref class SampleHandler
{
public:
    static void c_ThresholdReached(Object^ sender, ThresholdReachedEventArgs^ e)
    {
        Console::WriteLine("The threshold of {0} was reached at {1}.",
                           e->Threshold, e->TimeReached);
        Environment::Exit(0);
    }
};

void main()
{
    Counter^ c = gcnew Counter((gcnew Random())->Next(10));
    c->ThresholdReached += gcnew EventHandler<ThresholdReachedEventArgs^>(SampleHandler::c_ThresholdReached);

    Console::WriteLine("press 'a' key to increase total");
    while (Console::ReadKey(true).KeyChar == 'a') {
        Console::WriteLine("adding one");
        c->Add(1);
    }
}

```

```

using System;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            Counter c = new Counter(new Random().Next(10));
            c.ThresholdReached += c_ThresholdReached;

            Console.WriteLine("press 'a' key to increase total");
            while (Console.ReadKey(true).KeyChar == 'a')
            {
                Console.WriteLine("adding one");
                c.Add(1);
            }
        }

        static void c_ThresholdReached(object sender, ThresholdReachedEventArgs e)
        {
            Console.WriteLine("The threshold of {0} was reached at {1}.", e.Threshold, e.TimeReached);
            Environment.Exit(0);
        }
    }

    class Counter
    {
        private int threshold;
        private int total;

        public Counter(int passedThreshold)
        {
            threshold = passedThreshold;
        }

        public void Add(int x)
        {
            total += x;
            if (total >= threshold)
            {
                ThresholdReachedEventArgs args = new ThresholdReachedEventArgs();
                args.Threshold = threshold;
                args.TimeReached = DateTime.Now;
                OnThresholdReached(args);
            }
        }

        protected virtual void OnThresholdReached(ThresholdReachedEventArgs e)
        {
            EventHandler<ThresholdReachedEventArgs> handler = ThresholdReached;
            if (handler != null)
            {
                handler(this, e);
            }
        }

        public event EventHandler<ThresholdReachedEventArgs> ThresholdReached;
    }

    public class ThresholdReachedEventArgs : EventArgs
    {
        public int Threshold { get; set; }
        public DateTime TimeReached { get; set; }
    }
}

```

```

Module Module1

Sub Main()
    Dim c As Counter = New Counter(New Random().Next(10))
    AddHandler c.ThresholdReached, AddressOf c_ThresholdReached

    Console.WriteLine("press 'a' key to increase total")
    While Console.ReadKey(True).KeyChar = "a"
        Console.WriteLine("adding one")
        c.Add(1)
    End While
End Sub

Sub c_ThresholdReached(sender As Object, e As ThresholdReachedEventArgs)
    Console.WriteLine("The threshold of {0} was reached at {1}.", e.Threshold, e.TimeReached)
    Environment.Exit(0)
End Sub
End Module

Class Counter
    Private threshold As Integer
    Private total As Integer

    Public Sub New(passedThreshold As Integer)
        threshold = passedThreshold
    End Sub

    Public Sub Add(x As Integer)
        total = total + x
        If (total >= threshold) Then
            Dim args As ThresholdReachedEventArgs = New ThresholdReachedEventArgs()
            args.Threshold = threshold
            args.TimeReached = DateTime.Now
            OnThresholdReached(args)
        End If
    End Sub

    Protected Overridable Sub OnThresholdReached(e As ThresholdReachedEventArgs)
        RaiseEvent ThresholdReached(Me, e)
    End Sub

    Public Event ThresholdReached As EventHandler(Of ThresholdReachedEventArgs)
End Class

Class ThresholdReachedEventArgs
    Inherits EventArgs

    Public Property Threshold As Integer
    Public Property TimeReached As DateTime
End Class

```

Example 3

The next example shows how to declare a delegate for an event. The delegate is named `ThresholdReachedEventHandler`. This is just an illustration. Typically, you do not have to declare a delegate for an event, because you can use either the `EventHandler` or the `EventHandler<TEventArgs>` delegate. You should declare a delegate only in rare scenarios, such as making your class available to legacy code that cannot use generics.

```

using System;

namespace ConsoleApplication1
{
    class Program

```

```

class Program
{
    static void Main(string[] args)
    {
        Counter c = new Counter(new Random().Next(10));
        c.ThresholdReached += c_ThresholdReached;

        Console.WriteLine("press 'a' key to increase total");
        while (Console.ReadKey(true).KeyChar == 'a')
        {
            Console.WriteLine("adding one");
            c.Add(1);
        }
    }

    static void c_ThresholdReached(Object sender, ThresholdReachedEventArgs e)
    {
        Console.WriteLine("The threshold of {0} was reached at {1}.", e.Threshold, e.TimeReached);
        Environment.Exit(0);
    }
}

class Counter
{
    private int threshold;
    private int total;

    public Counter(int passedThreshold)
    {
        threshold = passedThreshold;
    }

    public void Add(int x)
    {
        total += x;
        if (total >= threshold)
        {
            ThresholdReachedEventArgs args = new ThresholdReachedEventArgs();
            args.Threshold = threshold;
            args.TimeReached = DateTime.Now;
            OnThresholdReached(args);
        }
    }

    protected virtual void OnThresholdReached(ThresholdReachedEventArgs e)
    {
        ThresholdReachedEventHandler handler = ThresholdReached;
        if (handler != null)
        {
            handler(this, e);
        }
    }

    public event ThresholdReachedEventHandler ThresholdReached;
}

public class ThresholdReachedEventArgs : EventArgs
{
    public int Threshold { get; set; }
    public DateTime TimeReached { get; set; }
}

public delegate void ThresholdReachedEventHandler(Object sender, ThresholdReachedEventArgs e);
}

```

```

Module Module1

Sub Main()
    Dim c As Counter = New Counter(New Random().Next(10))
    AddHandler c.ThresholdReached, AddressOf c_ThresholdReached

    Console.WriteLine("press 'a' key to increase total")
    While Console.ReadKey(True).KeyChar = "a"
        Console.WriteLine("adding one")
        c.Add(1)
    End While
End Sub

Sub c_ThresholdReached(sender As Object, e As ThresholdReachedEventArgs)
    Console.WriteLine("The threshold of {0} was reached at {1}.", e.Threshold, e.TimeReached)
    Environment.Exit(0)
End Sub
End Module

Class Counter
    Private threshold As Integer
    Private total As Integer

    Public Sub New(passedThreshold As Integer)
        threshold = passedThreshold
    End Sub

    Public Sub Add(x As Integer)
        total = total + x
        If (total >= threshold) Then
            Dim args As ThresholdReachedEventArgs = New ThresholdReachedEventArgs()
            args.Threshold = threshold
            args.TimeReached = DateTime.Now
            OnThresholdReached(args)
        End If
    End Sub

    Protected Overridable Sub OnThresholdReached(e As ThresholdReachedEventArgs)
        RaiseEvent ThresholdReached(Me, e)
    End Sub

    Public Event ThresholdReached As ThresholdReachedEventHandler
End Class

Public Class ThresholdReachedEventArgs
    Inherits EventArgs

    Public Property Threshold As Integer
    Public Property TimeReached As DateTime
End Class

Public Delegate Sub ThresholdReachedEventHandler(sender As Object, e As ThresholdReachedEventArgs)

```

See also

- [Events](#)

How to: Handle Multiple Events Using Event Properties

9/20/2022 • 4 minutes to read • [Edit Online](#)

To use event properties, you define the event properties in the class that raises the events, and then set the delegates for the event properties in classes that handle the events. To implement multiple event properties in a class, the class should internally store and maintain the delegate defined for each event. For each field-like-event, a corresponding backing-field reference type is generated. This can lead to unnecessary allocations when the number of events increases. As an alternative, a common approach is to maintain an [EventHandlerList](#) which stores events by key.

To store the delegates for each event, you can use the [EventHandlerList](#) class, or implement your own collection. The collection class must provide methods for setting, accessing, and retrieving the event handler delegate based on the event key. For example, you could use a [Hashtable](#) class, or derive a custom class from the [DictionaryBase](#) class. The implementation details of the delegate collection do not need to be exposed outside your class.

Each event property within the class defines an add accessor method and a remove accessor method. The add accessor for an event property adds the input delegate instance to the delegate collection. The remove accessor for an event property removes the input delegate instance from the delegate collection. The event property accessors use the predefined key for the event property to add and remove instances from the delegate collection.

To handle multiple events using event properties

1. Define a delegate collection within the class that raises the events.
2. Define a key for each event.
3. Define the event properties in the class that raises the events.
4. Use the delegate collection to implement the add and remove accessor methods for the event properties.
5. Use the public event properties to add and remove event handler delegates in the classes that handle the events.

Example

The following C# example implements the event properties `MouseDown` and `MouseUp`, using an [EventHandlerList](#) to store each event's delegate. The keywords of the event property constructs are in bold type.

```

// The class SampleControl defines two event properties, MouseUp and MouseDown.
ref class SampleControl : Component
{
    // :
    // Define other control methods and properties.
    // :

    // Define the delegate collection.

protected:
    EventHandlerList^ listEventDelegates;

private:
    // Define a unique key for each event.
    static Object^ mouseDownEventKey = gcnew Object();
    static Object^ mouseUpEventKey = gcnew Object();

    // Define the MouseDown event property.

public:
    SampleControl()
    {
        listEventDelegates = gcnew EventHandlerList();
    }

    event MouseEventHandler^ MouseDown
    {
        // Add the input delegate to the collection.
        void add(MouseEventHandler^ value)
        {
            listEventDelegates->AddHandler(mouseDownEventKey, value);
        }
        // Remove the input delegate from the collection.
        void remove(MouseEventHandler^ value)
        {
            listEventDelegates->RemoveHandler(mouseDownEventKey, value);
        }
        // Raise the event with the delegate specified by mouseDownEventKey
        void raise(Object^ sender, MouseEventArgs^ e)
        {
            MouseEventHandler^ mouseEventDelegate =
                (MouseEventHandler^)listEventDelegates[mouseDownEventKey];
            mouseEventDelegate(sender, e);
        }
    }

    // Define the MouseUp event property.

    event MouseEventHandler^ MouseUp
    {
        // Add the input delegate to the collection.
        void add(MouseEventHandler^ value)
        {
            listEventDelegates->AddHandler(mouseUpEventKey, value);
        }
        // Remove the input delegate from the collection.
        void remove(MouseEventHandler^ value)
        {
            listEventDelegates->RemoveHandler(mouseUpEventKey, value);
        }
        // Raise the event with the delegate specified by mouseUpEventKey
        void raise(Object^ sender, MouseEventArgs^ e)
        {
            MouseEventHandler^ mouseEventDelegate =
                (MouseEventHandler^)listEventDelegates[mouseUpEventKey];
            mouseEventDelegate(sender, e);
        }
    }
};


```

```
// The class SampleControl defines two event properties, MouseUp and MouseDown.
class SampleControl : Component
{
    // :
    // Define other control methods and properties.
    // :

    // Define the delegate collection.
    protected EventHandlerList listEventDelegates = new EventHandlerList();

    // Define a unique key for each event.
    static readonly object mouseDownEventKey = new object();
    static readonly object mouseUpEventKey = new object();

    // Define the MouseDown event property.
    public event MouseEventHandler MouseDown
    {
        // Add the input delegate to the collection.
        add
        {
            listEventDelegates.AddHandler(mouseDownEventKey, value);
        }
        // Remove the input delegate from the collection.
        remove
        {
            listEventDelegates.RemoveHandler(mouseDownEventKey, value);
        }
    }

    // Raise the event with the delegate specified by mouseDownEventKey
    private void OnMouseDown(MouseEventArgs e)
    {
        MouseEventHandler mouseEventDelegate =
            (MouseEventHandler)listEventDelegates[mouseDownEventKey];
        mouseEventDelegate(this, e);
    }

    // Define the MouseUp event property.
    public event MouseEventHandler MouseUp
    {
        // Add the input delegate to the collection.
        add
        {
            listEventDelegates.AddHandler(mouseUpEventKey, value);
        }
        // Remove the input delegate from the collection.
        remove
        {
            listEventDelegates.RemoveHandler(mouseUpEventKey, value);
        }
    }

    // Raise the event with the delegate specified by mouseUpEventKey
    private void OnMouseUp(MouseEventArgs e)
    {
        MouseEventHandler mouseEventDelegate =
            (MouseEventHandler)listEventDelegates[mouseUpEventKey];
        mouseEventDelegate(this, e);
    }
}
```

```

' The class SampleControl defines two event properties, MouseUp and MouseDown.
Class SampleControl
    Inherits Component
    ' :
    ' Define other control methods and properties.
    ' :

    ' Define the delegate collection.
    Protected listEventDelegates As New EventHandlerList()

    ' Define a unique key for each event.
    Shared ReadOnly mouseDownEventKey As New Object()
    Shared ReadOnly mouseUpEventKey As New Object()

    ' Define the MouseDown event property.
    Public Custom Event MouseDown As MouseEventHandler
        ' Add the input delegate to the collection.
        AddHandler(Value As MouseEventHandler)
            listEventDelegates.AddHandler(mouseDownEventKey, Value)
        End AddHandler
        ' Remove the input delegate from the collection.
        RemoveHandler(Value As MouseEventHandler)
            listEventDelegates.RemoveHandler(mouseDownEventKey, Value)
        End RemoveHandler
        ' Raise the event with the delegate specified by mouseDownEventKey
        RaiseEvent(sender As Object, e As MouseEventArgs)
            Dim mouseEventDelegate As MouseEventHandler = _
                listEventDelegates(mouseDownEventKey)
            mouseEventDelegate(sender, e)
        End RaiseEvent
    End Event

    ' Define the MouseUp event property.
    Public Custom Event MouseUp As MouseEventHandler
        ' Add the input delegate to the collection.
        AddHandler(Value As MouseEventHandler)
            listEventDelegates.AddHandler(mouseUpEventKey, Value)
        End AddHandler
        ' Remove the input delegate from the collection.
        RemoveHandler(Value As MouseEventHandler)
            listEventDelegates.RemoveHandler(mouseUpEventKey, Value)
        End RemoveHandler
        ' Raise the event with the delegate specified by mouseUpEventKey
        RaiseEvent(sender As Object, e As MouseEventArgs)
            Dim mouseEventDelegate As MouseEventHandler = _
                listEventDelegates(mouseUpEventKey)
            mouseEventDelegate(sender, e)
        End RaiseEvent
    End Event
End Class

```

See also

- [System.ComponentModel.EventHandlerList](#)
- [Events](#)
- [Control.Events](#)
- [How to: Declare Custom Events To Conserve Memory](#)

Observer Design Pattern

9/20/2022 • 13 minutes to read • [Edit Online](#)

The observer design pattern enables a subscriber to register with and receive notifications from a provider. It is suitable for any scenario that requires push-based notification. The pattern defines a *provider* (also known as a *subject* or an *observable*) and zero, one, or more *observers*. Observers register with the provider, and whenever a predefined condition, event, or state change occurs, the provider automatically notifies all observers by calling one of their methods. In this method call, the provider can also provide current state information to observers. In .NET, the observer design pattern is applied by implementing the generic `System.IObservable<T>` and `System.IObserver<T>` interfaces. The generic type parameter represents the type that provides notification information.

Applying the Pattern

The observer design pattern is suitable for distributed push-based notifications, because it supports a clean separation between two different components or application layers, such as a data source (business logic) layer and a user interface (display) layer. The pattern can be implemented whenever a provider uses callbacks to supply its clients with current information.

Implementing the pattern requires that you provide the following:

- A provider or subject, which is the object that sends notifications to observers. A provider is a class or structure that implements the `IObservable<T>` interface. The provider must implement a single method, `IObservable<T>.Subscribe`, which is called by observers that wish to receive notifications from the provider.
- An observer, which is an object that receives notifications from a provider. An observer is a class or structure that implements the `IObserver<T>` interface. The observer must implement three methods, all of which are called by the provider:
 - `IObserver<T>.OnNext`, which supplies the observer with new or current information.
 - `IObserver<T>.OnError`, which informs the observer that an error has occurred.
 - `IObserver<T>.OnCompleted`, which indicates that the provider has finished sending notifications.
- A mechanism that allows the provider to keep track of observers. Typically, the provider uses a container object, such as a `System.Collections.Generic.List<T>` object, to hold references to the `IObserver<T>` implementations that have subscribed to notifications. Using a storage container for this purpose enables the provider to handle zero to an unlimited number of observers. The order in which observers receive notifications is not defined; the provider is free to use any method to determine the order.
- An `IDisposable` implementation that enables the provider to remove observers when notification is complete. Observers receive a reference to the `IDisposable` implementation from the `Subscribe` method, so they can also call the `IDisposable.Dispose` method to unsubscribe before the provider has finished sending notifications.
- An object that contains the data that the provider sends to its observers. The type of this object corresponds to the generic type parameter of the `IObservable<T>` and `IObserver<T>` interfaces. Although this object can be the same as the `IObservable<T>` implementation, most commonly it is a separate type.

NOTE

In addition to implementing the observer design pattern, you may be interested in exploring libraries that are built using the `IObservable<T>` and `IObserver<T>` interfaces. For example, [Reactive Extensions for .NET \(Rx\)](#) consist of a set of extension methods and LINQ standard sequence operators to support asynchronous programming.

Implementing the Pattern

The following example uses the observer design pattern to implement an airport baggage claim information system. A `BaggageInfo` class provides information about arriving flights and the carousels where baggage from each flight is available for pickup. It is shown in the following example.

```
using System;
using System.Collections.Generic;

public class BaggageInfo
{
    private int flightNo;
    private string origin;
    private int location;

    internal BaggageInfo(int flight, string from, int carousel)
    {
        this.flightNo = flight;
        this.origin = from;
        this.location = carousel;
    }

    public int FlightNumber {
        get { return this.flightNo; }
    }

    public string From {
        get { return this.origin; }
    }

    public int Carousel {
        get { return this.location; }
    }
}
```

```

Public Class BaggageInfo
    Private flightNo As Integer
    Private origin As String
    Private location As Integer

    Friend Sub New(ByVal flight As Integer, ByVal from As String, ByVal carousel As Integer)
        Me.flightNo = flight
        Me.origin = from
        Me.location = carousel
    End Sub

    Public ReadOnly Property FlightNumber As Integer
        Get
            Return Me.flightNo
        End Get
    End Property

    Public ReadOnly Property From As String
        Get
            Return Me.origin
        End Get
    End Property

    Public ReadOnly Property Carousel As Integer
        Get
            Return Me.location
        End Get
    End Property
End Class

```

A `BaggageHandler` class is responsible for receiving information about arriving flights and baggage claim carousels. Internally, it maintains two collections:

- `observers` - A collection of clients that will receive updated information.
- `flights` - A collection of flights and their assigned carousels.

Both collections are represented by generic `List<T>` objects that are instantiated in the `BaggageHandler` class constructor. The source code for the `BaggageHandler` class is shown in the following example.

```

public class BaggageHandler : IObservable<BaggageInfo>
{
    private List<IObserver<BaggageInfo>> observers;
    private List<BaggageInfo> flights;

    public BaggageHandler()
    {
        observers = new List<IObserver<BaggageInfo>>();
        flights = new List<BaggageInfo>();
    }

    public IDisposable Subscribe(IObserver<BaggageInfo> observer)
    {
        // Check whether observer is already registered. If not, add it
        if (!observers.Contains(observer)) {
            observers.Add(observer);
            // Provide observer with existing data.
            foreach (var item in flights)
                observer.OnNext(item);
        }
        return new Unsubscriber<BaggageInfo>(observers, observer);
    }

    // Called to indicate all baggage is now unloaded.
    public void BaggageStatus(int flightNo)
    {
        BaggageStatus(flightNo, String.Empty, 0);
    }

    public void BaggageStatus(int flightNo, string from, int carousel)
    {
        var info = new BaggageInfo(flightNo, from, carousel);

        // Carousel is assigned, so add new info object to list.
        if (carousel > 0 && !flights.Contains(info)) {
            flights.Add(info);
            foreach (var observer in observers)
                observer.OnNext(info);
        }
        else if (carousel == 0) {
            // Baggage claim for flight is done
            var flightsToRemove = new List<BaggageInfo>();
            foreach (var flight in flights) {
                if (info.FlightNumber == flight.FlightNumber) {
                    flightsToRemove.Add(flight);
                    foreach (var observer in observers)
                        observer.OnNext(info);
                }
            }
            foreach (var flightToRemove in flightsToRemove)
                flights.Remove(flightToRemove);

            flightsToRemove.Clear();
        }
    }

    public void LastBaggageClaimed()
    {
        foreach (var observer in observers)
            observer.OnCompleted();

        observers.Clear();
    }
}

```

```

Public Class BaggageHandler : Implements IObservable(Of BaggageInfo)

    Private observers As List(Of IObserver(Of BaggageInfo))
    Private flights As List(Of BaggageInfo)

    Public Sub New()
        observers = New List(Of IObserver(Of BaggageInfo))
        flights = New List(Of BaggageInfo)
    End Sub

    Public Function Subscribe(ByVal observer As IObserver(Of BaggageInfo)) As IDisposable _
        Implements IObservable(Of BaggageInfo).Subscribe
        ' Check whether observer is already registered. If not, add it
        If Not observers.Contains(observer) Then
            observers.Add(observer)
            ' Provide observer with existing data.
            For Each item In flights
                observer.OnNext(item)
            Next
        End If
        Return New Unsubscriber(Of BaggageInfo)(observers, observer)
    End Function

    ' Called to indicate all baggage is now unloaded.
    Public Sub BaggageStatus(ByVal flightNo As Integer)
        BaggageStatus(flightNo, String.Empty, 0)
    End Sub

    Public Sub BaggageStatus(ByVal flightNo As Integer, ByVal from As String, ByVal carousel As Integer)
        Dim info As New BaggageInfo(flightNo, from, carousel)

        ' Carousel is assigned, so add new info object to list.
        If carousel > 0 And Not flights.Contains(info) Then
            flights.Add(info)
            For Each observer In observers
                observer.OnNext(info)
            Next
        ElseIf carousel = 0 Then
            ' Baggage claim for flight is done
            Dim flightsToRemove As New List(Of BaggageInfo)
            For Each flight In flights
                If info.FlightNumber = flight.FlightNumber Then
                    flightsToRemove.Add(flight)
                    For Each observer In observers
                        observer.OnNext(info)
                    Next
                End If
            Next
            For Each flightToRemove In flightsToRemove
                flights.Remove(flightToRemove)
            Next
            flightsToRemove.Clear()
        End If
    End Sub

    Public Sub LastBaggageClaimed()
        For Each observer In observers
            observer.OnCompleted()
        Next
        observers.Clear()
    End Sub
End Class

```

Clients that wish to receive updated information call the `BaggageHandler.Subscribe` method. If the client has not previously subscribed to notifications, a reference to the client's `IObserver<T>` implementation is added to the `observers` collection.

The overloaded `BaggageHandler.BaggageStatus` method can be called to indicate that baggage from a flight either is being unloaded or is no longer being unloaded. In the first case, the method is passed a flight number, the airport from which the flight originated, and the carousel where baggage is being unloaded. In the second case, the method is passed only a flight number. For baggage that is being unloaded, the method checks whether the `BaggageInfo` information passed to the method exists in the `flights` collection. If it does not, the method adds the information and calls each observer's `OnNext` method. For flights whose baggage is no longer being unloaded, the method checks whether information on that flight is stored in the `flights` collection. If it is, the method calls each observer's `OnNext` method and removes the `BaggageInfo` object from the `flights` collection.

When the last flight of the day has landed and its baggage has been processed, the `BaggageHandler.LastBaggageClaimed` method is called. This method calls each observer's `OnCompleted` method to indicate that all notifications have completed, and then clears the `observers` collection.

The provider's `Subscribe` method returns an `IDisposable` implementation that enables observers to stop receiving notifications before the `OnCompleted` method is called. The source code for this `Unsubscriber<Of BaggageInfo>` class is shown in the following example. When the class is instantiated in the `BaggageHandler.Subscribe` method, it is passed a reference to the `observers` collection and a reference to the observer that is added to the collection. These references are assigned to local variables. When the object's `Dispose` method is called, it checks whether the observer still exists in the `observers` collection, and, if it does, removes the observer.

```
internal class Unsubscriber<Of BaggageInfo> : IDisposable
{
    private List<IObserver<BaggageInfo>> _observers;
    private IObserver<BaggageInfo> _observer;

    internal Unsubscriber(List<IObserver<BaggageInfo>> observers, IObserver<BaggageInfo> observer)
    {
        this._observers = observers;
        this._observer = observer;
    }

    public void Dispose()
    {
        if (_observers.Contains(_observer))
            _observers.Remove(_observer);
    }
}
```

```
Friend Class Unsubscriber(Of BaggageInfo) : Implements IDisposable
    Private _observers As List(Of IObserver(Of BaggageInfo))
    Private _observer As IObserver(Of BaggageInfo)

    Friend Sub New(ByVal observers As List(Of IObserver(Of BaggageInfo)), ByVal observer As IObserver(Of BaggageInfo))
        Me._observers = observers
        Me._observer = observer
    End Sub

    Public Sub Dispose() Implements IDisposable.Dispose
        If _observers.Contains(_observer) Then
            _observers.Remove(_observer)
        End If
    End Sub
End Class
```

The following example provides an `IObserver<T>` implementation named `ArrivalsMonitor`, which is a base class that displays baggage claim information. The information is displayed alphabetically, by the name of the

originating city. The methods of `ArrivalsMonitor` are marked as `overridable` (in Visual Basic) or `virtual` (in C#), so they can all be overridden by a derived class.

```
using System;
using System.Collections.Generic;

public class ArrivalsMonitor : IObserver<BaggageInfo>
{
    private string name;
    private List<string> flightInfos = new List<string>();
    private IDisposable cancellation;
    private string fmt = "{0,-20} {1,5} {2, 3}";

    public ArrivalsMonitor(string name)
    {
        if (String.IsNullOrEmpty(name))
            throw new ArgumentNullException("The observer must be assigned a name.");

        this.name = name;
    }

    public virtual void Subscribe(BaggageHandler provider)
    {
        cancellation = provider.Subscribe(this);
    }

    public virtual void Unsubscribe()
    {
        cancellation.Dispose();
        flightInfos.Clear();
    }

    public virtual void OnCompleted()
    {
        flightInfos.Clear();
    }

    // No implementation needed: Method is not called by the BaggageHandler class.
    public virtual void OnError(Exception e)
    {
        // No implementation.
    }

    // Update information.
    public virtual void OnNext(BaggageInfo info)
    {
        bool updated = false;

        // Flight has unloaded its baggage; remove from the monitor.
        if (info.Carousel == 0) {
            var flightsToRemove = new List<string>();
            string flightNo = String.Format("{0,5}", info.FlightNumber);

            foreach (var flightInfo in flightInfos) {
                if (flightInfo.Substring(21, 5).Equals(flightNo)) {
                    flightsToRemove.Add(flightInfo);
                    updated = true;
                }
            }
            foreach (var flightToRemove in flightsToRemove)
                flightInfos.Remove(flightToRemove);

            flightsToRemove.Clear();
        }
        else {
            // Add flight if it does not exist in the collection.
            string flightInfo = String.Format(fmt, info.From, info.FlightNumber, info.Carousel);
            flightInfos.Add(flightInfo);
        }
    }
}
```

```

        if (! flightInfos.Contains(flightInfo)) {
            flightInfos.Add(flightInfo);
            updated = true;
        }
    }
    if (updated) {
        flightInfos.Sort();
        Console.WriteLine("Arrivals information from {0}", this.name);
        foreach (var flightInfo in flightInfos)
            Console.WriteLine(flightInfo);

        Console.WriteLine();
    }
}
}

```

```

Public Class ArrivalsMonitor : Implements IObservable(Of BaggageInfo)
    Private name As String
    Private flightInfos As New List(Of String)
    Private cancellation As IDisposable
    Private fmt As String = "{0,-20} {1,5} {2, 3}"

    Public Sub New(ByVal name As String)
        If String.IsNullOrEmpty(name) Then Throw New ArgumentNullException("The observer must be assigned a name.")
        Me.name = name
    End Sub

    Public Overridable Sub Subscribe(ByVal provider As BaggageHandler)
        cancellation = provider.Subscribe(Me)
    End Sub

    Public Overridable Sub Unsubscribe()
        cancellation.Dispose()
        flightInfos.Clear()
    End Sub

    Public Overridable Sub OnCompleted() Implements System.IObservable(Of BaggageInfo).OnCompleted
        flightInfos.Clear()
    End Sub

    ' No implementation needed: Method is not called by the BaggageHandler class.
    Public Overridable Sub OnError(ByVal e As System.Exception) Implements System.IObservable(Of BaggageInfo).OnError
        ' No implementation.
    End Sub

    ' Update information.
    Public Overridable Sub OnNext(ByVal info As BaggageInfo) Implements System.IObservable(Of BaggageInfo).OnNext
        Dim updated As Boolean = False

        ' Flight has unloaded its baggage; remove from the monitor.
        If info.Carousel = 0 Then
            Dim flightsToRemove As New List(Of String)
            Dim flightNo As String = String.Format("{0,5}", info.FlightNumber)
            For Each flightInfo In flightInfos
                If flightInfo.Substring(21, 5).Equals(flightNo) Then
                    flightsToRemove.Add(flightInfo)
                    updated = True
                End If
            Next
            For Each flightToRemove In flightsToRemove
                flightInfos.Remove(flightToRemove)
            Next
            flightsToRemove.Clear()
        End If
    End Sub

```

```
        Else
            ' Add flight if it does not exist in the collection.
            Dim flightInfo As String = String.Format(fmt, info.From, info.FlightNumber, info.Carousel)
            If Not flightInfos.Contains(flightInfo) Then
                flightInfos.Add(flightInfo)
                updated = True
            End If
        End If
        If updated Then
            flightInfos.Sort()
            Console.WriteLine("Arrivals information from {0}", Me.name)
            For Each flightInfo In flightInfos
                Console.WriteLine(flightInfo)
            Next
            Console.WriteLine()
        End If
    End Sub
End Class
```

The `ArrivalsMonitor` class includes the `Subscribe` and `Unsubscribe` methods. The `Subscribe` method enables the class to save the `IDisposable` implementation returned by the call to `Subscribe` to a private variable. The `Unsubscribe` method enables the class to unsubscribe from notifications by calling the provider's `Dispose` implementation. `ArrivalsMonitor` also provides implementations of the `OnNext`, `OnError`, and `OnCompleted` methods. Only the `OnNext` implementation contains a significant amount of code. The method works with a private, sorted, generic `List<T>` object that maintains information about the airports of origin for arriving flights and the carousels on which their baggage is available. If the `BaggageHandler` class reports a new flight arrival, the `OnNext` method implementation adds information about that flight to the list. If the `BaggageHandler` class reports that the flight's baggage has been unloaded, the `OnNext` method removes that flight from the list. Whenever a change is made, the list is sorted and displayed to the console.

The following example contains the application entry point that instantiates the `BaggageHandler` class as well as two instances of the `ArrivalsMonitor` class, and uses the `BaggageHandler.BaggageStatus` method to add and remove information about arriving flights. In each case, the observers receive updates and correctly display baggage claim information.

```
using System;
using System.Collections.Generic;

public class Example
{
    public static void Main()
    {
        BaggageHandler provider = new BaggageHandler();
        ArrivalsMonitor observer1 = new ArrivalsMonitor("BaggageClaimMonitor1");
        ArrivalsMonitor observer2 = new ArrivalsMonitor("SecurityExit");

        provider.BaggageStatus(712, "Detroit", 3);
        observer1.Subscribe(provider);
        provider.BaggageStatus(712, "Kalamazoo", 3);
        provider.BaggageStatus(400, "New York-Kennedy", 1);
        provider.BaggageStatus(712, "Detroit", 3);
        observer2.Subscribe(provider);
        provider.BaggageStatus(511, "San Francisco", 2);
        provider.BaggageStatus(712);
        observer2.Unsubscribe();
        provider.BaggageStatus(400);
        provider.LastBaggageClaimed();
    }
}

// The example displays the following output:
//      Arrivals information from BaggageClaimMonitor1
//      Detroit          712      3
```

```
//      Arrivals information from BaggageClaimMonitor1
//      Detroit          712    3
//      Kalamazoo        712    3
//
//      Arrivals information from BaggageClaimMonitor1
//      Detroit          712    3
//      Kalamazoo        712    3
//      New York-Kennedy 400    1
//
//      Arrivals information from SecurityExit
//      Detroit          712    3
//
//      Arrivals information from SecurityExit
//      Detroit          712    3
//      Kalamazoo        712    3
//
//      Arrivals information from SecurityExit
//      Detroit          712    3
//      Kalamazoo        712    3
//      New York-Kennedy 400    1
//
//      Arrivals information from BaggageClaimMonitor1
//      Detroit          712    3
//      Kalamazoo        712    3
//      New York-Kennedy 400    1
//      San Francisco    511    2
//
//      Arrivals information from SecurityExit
//      Detroit          712    3
//      Kalamazoo        712    3
//      New York-Kennedy 400    1
//      San Francisco    511    2
//
//      Arrivals information from BaggageClaimMonitor1
//      New York-Kennedy 400    1
//      San Francisco    511    2
//
//      Arrivals information from SecurityExit
//      New York-Kennedy 400    1
//      San Francisco    511    2
//
//      Arrivals information from BaggageClaimMonitor1
//      San Francisco    511    2
```

Module Example

```
Public Sub Main()
    Dim provider As New BaggageHandler()
    Dim observer1 As New ArrivalsMonitor("BaggageClaimMonitor1")
    Dim observer2 As New ArrivalsMonitor("SecurityExit")

    provider.BaggageStatus(712, "Detroit", 3)
    observer1.Subscribe(provider)
    provider.BaggageStatus(712, "Kalamazoo", 3)
    provider.BaggageStatus(400, "New York-Kennedy", 1)
    provider.BaggageStatus(712, "Detroit", 3)
    observer2.Subscribe(provider)
    provider.BaggageStatus(511, "San Francisco", 2)
    provider.BaggageStatus(712)
    observer2.Unsubscribe()
    provider.BaggageStatus(400)
    provider.LastBaggageClaimed()

End Sub
End Module

' The example displays the following output:
'   Arrivals information from BaggageClaimMonitor1
'   Detroit          712      3
'
'   Arrivals information from BaggageClaimMonitor1
'   Detroit          712      3
'   Kalamazoo        712      3
'
'   Arrivals information from BaggageClaimMonitor1
'   Detroit          712      3
'   Kalamazoo        712      3
'   New York-Kennedy 400      1
'
'   Arrivals information from SecurityExit
'   Detroit          712      3
'
'   Arrivals information from SecurityExit
'   Detroit          712      3
'   Kalamazoo        712      3
'
'   Arrivals information from SecurityExit
'   Detroit          712      3
'   Kalamazoo        712      3
'   New York-Kennedy 400      1
'
'   Arrivals information from BaggageClaimMonitor1
'   Detroit          712      3
'   Kalamazoo        712      3
'   New York-Kennedy 400      1
'   San Francisco    511      2
'
'   Arrivals information from SecurityExit
'   Detroit          712      3
'   Kalamazoo        712      3
'   New York-Kennedy 400      1
'   San Francisco    511      2
'
'   Arrivals information from BaggageClaimMonitor1
'   New York-Kennedy 400      1
'   San Francisco    511      2
'
'   Arrivals information from SecurityExit
'   New York-Kennedy 400      1
'   San Francisco    511      2
'
'   Arrivals information from BaggageClaimMonitor1
'   San Francisco    511      2
```

Related Topics

TITLE	DESCRIPTION
Observer Design Pattern Best Practices	Describes best practices to adopt when developing applications that implement the observer design pattern.
How to: Implement a Provider	Provides a step-by-step implementation of a provider for a temperature monitoring application.
How to: Implement an Observer	Provides a step-by-step implementation of an observer for a temperature monitoring application.

Observer Design Pattern Best Practices

9/20/2022 • 3 minutes to read • [Edit Online](#)

In .NET, the observer design pattern is implemented as a set of interfaces. The [System.IObservable<T>](#) interface represents the data provider, which is also responsible for providing an [IDisposable](#) implementation that lets observers unsubscribe from notifications. The [System.IObserver<T>](#) interface represents the observer. This topic describes the best practices that developers should follow when implementing the observer design pattern using these interfaces.

Threading

Typically, a provider implements the [IObservable<T>.Subscribe](#) method by adding a particular observer to a subscriber list that is represented by some collection object, and it implements the [IDisposable.Dispose](#) method by removing a particular observer from the subscriber list. An observer can call these methods at any time. In addition, because the provider/observer contract does not specify who is responsible for unsubscribing after the [IObserver<T>.OnCompleted](#) callback method, the provider and observer may both try to remove the same member from the list. Because of this possibility, both the [Subscribe](#) and [Dispose](#) methods should be thread-safe. Typically, this involves using a [concurrent collection](#) or a lock. Implementations that are not thread-safe should explicitly document that they are not.

Any additional guarantees have to be specified in a layer on top of the provider/observer contract. Implementers should clearly call out when they impose additional requirements to avoid user confusion about the observer contract.

Handling Exceptions

Because of the loose coupling between a data provider and an observer, exceptions in the observer design pattern are intended to be informational. This affects how providers and observers handle exceptions in the observer design pattern.

The Provider -- Calling the OnError Method

The [OnError](#) method is intended as an informational message to observers, much like the [IObserver<T>.OnNext](#) method. However, the [OnNext](#) method is designed to provide an observer with current or updated data, whereas the [OnError](#) method is designed to indicate that the provider is unable to provide valid data.

The provider should follow these best practices when handling exceptions and calling the [OnError](#) method:

- The provider must handle its own exceptions if it has any specific requirements.
- The provider should not expect or require that observers handle exceptions in any particular way.
- The provider should call the [OnError](#) method when it handles an exception that compromises its ability to provide updates. Information on such exceptions can be passed to the observer. In other cases, there is no need to notify observers of an exception.

Once the provider calls the [OnError](#) or [IObserver<T>.OnCompleted](#) method, there should be no further notifications, and the provider can unsubscribe its observers. However, the observers can also unsubscribe themselves at any time, including both before and after they receive an [OnError](#) or [IObserver<T>.OnCompleted](#) notification. The observer design pattern does not dictate whether the provider or the observer is responsible for unsubscribing; therefore, there is a possibility that both may attempt to unsubscribe. Typically, when observers unsubscribe, they are removed from a subscribers collection. In a single-threaded application, the

`IDisposable.Dispose` implementation should ensure that an object reference is valid and that the object is a member of the subscribers collection before attempting to remove it. In a multithreaded application, a thread-safe collection object, such as a `System.Collections.Concurrent.BlockingCollection<T>` object, should be used.

The Observer -- Implementing the OnError Method

When an observer receives an error notification from a provider, the observer should treat the exception as informational and should not be required to take any particular action.

The observer should follow these best practices when responding to an `OnError` method call from a provider:

- The observer should not throw exceptions from its interface implementations, such as `OnNext` or `OnError`. However, if the observer does throw exceptions, it should expect these exceptions to go unhandled.
- To preserve the call stack, an observer that wishes to throw an `Exception` object that was passed to its `OnError` method should wrap the exception before throwing it. A standard exception object should be used for this purpose.

Additional Best Practices

Attempting to unregister in the `IObservable<T>.Subscribe` method may result in a null reference. Therefore, we recommend that you avoid this practice.

Although it is possible to attach an observer to multiple providers, the recommended pattern is to attach an `IObserver<T>` instance to only one `IObservable<T>` instance.

See also

- [Observer Design Pattern](#)
- [How to: Implement an Observer](#)
- [How to: Implement a Provider](#)

How to: Implement a Provider

9/20/2022 • 7 minutes to read • [Edit Online](#)

The observer design pattern requires a division between a provider, which monitors data and sends notifications, and one or more observers, which receive notifications (callbacks) from the provider. This topic discusses how to create a provider. A related topic, [How to: Implement an Observer](#), discusses how to create an observer.

To create a provider

1. Define the data that the provider is responsible for sending to observers. Although the provider and the data that it sends to observers can be a single type, they are generally represented by different types. For example, in a temperature monitoring application, the `Temperature` structure defines the data that the provider (which is represented by the `TemperatureMonitor` class defined in the next step) monitors and to which observers subscribe.

```
using System;

public struct Temperature
{
    private decimal temp;
    private DateTime tempDate;

    public Temperature(decimal temperature, DateTime dateAndTime)
    {
        this.temp = temperature;
        this.tempDate = dateAndTime;
    }

    public decimal Degrees
    { get { return this.temp; } }

    public DateTime Date
    { get { return this.tempDate; } }
}
```

```
Public Structure Temperature
    Private temp As Decimal
    Private tempDate As DateTime

    Public Sub New(ByVal temperature As Decimal, ByVal dateAndTime As DateTime)
        Me.temp = temperature
        Me.tempDate = dateAndTime
    End Sub

    Public ReadOnly Property Degrees As Decimal
        Get
            Return Me.temp
        End Get
    End Property

    Public ReadOnly Property [Date] As DateTime
        Get
            Return tempDate
        End Get
    End Property
End Structure
```

2. Define the data provider, which is a type that implements the `System.IObservable<T>` interface. The provider's generic type argument is the type that the provider sends to observers. The following example defines a `TemperatureMonitor` class, which is a constructed `System.IObservable<T>` implementation with a generic type argument of `Temperature`.

```
using System;
using System.Collections.Generic;

public class TemperatureMonitor : IObservable<Temperature>
{
```

```
Imports System.Collections.Generic

Public Class TemperatureMonitor : Implements IObservable(Of Temperature)
```

3. Determine how the provider will store references to observers so that each observer can be notified when appropriate. Most commonly, a collection object such as a generic `List<T>` object is used for this purpose. The following example defines a private `List<T>` object that is instantiated in the `TemperatureMonitor` class constructor.

```
using System;
using System.Collections.Generic;

public class TemperatureMonitor : IObservable<Temperature>
{
    List<IObserver<Temperature>> observers;

    public TemperatureMonitor()
    {
        observers = new List<IObserver<Temperature>>();
    }
}
```

```
Imports System.Collections.Generic

Public Class TemperatureMonitor : Implements IObservable(Of Temperature)
    Dim observers As List(Of IObserver(Of Temperature))

    Public Sub New()
        observers = New List(Of IObserver(Of Temperature))
    End Sub

```

4. Define an `IDisposable` implementation that the provider can return to subscribers so that they can stop receiving notifications at any time. The following example defines a nested `Unsubscriber` class that is passed a reference to the subscribers collection and to the subscriber when the class is instantiated. This code enables the subscriber to call the object's `IDisposable.Dispose` implementation to remove itself from the subscribers collection.

```

private class Unsubscriber : IDisposable
{
    private List<IObserver<Temperature>> _observers;
    private IObserver<Temperature> _observer;

    public Unsubscriber(List<IObserver<Temperature>> observers, IObserver<Temperature> observer)
    {
        this._observers = observers;
        this._observer = observer;
    }

    public void Dispose()
    {
        if (!(_observer == null)) _observers.Remove(_observer);
    }
}

```

```

Private Class Unsubscriber : Implements IDisposable
    Private _observers As List(Of IObserver(Of Temperature))
    Private _observer As IObserver(Of Temperature)

    Public Sub New(ByVal observers As List(Of IObserver(Of Temperature)), ByVal observer As
IOObserver(Of Temperature))
        Me._observers = observers
        Me._observer = observer
    End Sub

    Public Sub Dispose() Implements IDisposable.Dispose
        If _observer IsNot Nothing Then _observers.Remove(_observer)
    End Sub
End Class

```

5. Implement the `IObservable<T>.Subscribe` method. The method is passed a reference to the `System.IObserver<T>` interface and should be stored in the object designed for that purpose in step 3. The method should then return the `IDisposable` implementation developed in step 4. The following example shows the implementation of the `Subscribe` method in the `TemperatureMonitor` class.

```

public IDisposable Subscribe(IObserver<Temperature> observer)
{
    if (!observers.Contains(observer))
        observers.Add(observer);

    return new Unsubscriber(observers, observer);
}

```

```

Public Function Subscribe(ByVal observer As System.IObserver(Of Temperature)) As System.IDisposable
Implements System.IObservable(Of Temperature).Subscribe
    If Not observers.Contains(observer) Then
        observers.Add(observer)
    End If
    Return New Unsubscriber(observers, observer)
End Function

```

6. Notify observers as appropriate by calling their `IObserver<T>.OnNext`, `IObserver<T>.OnError`, and `IObserver<T>.OnCompleted` implementations. In some cases, a provider may not call the `OnError` method when an error occurs. For example, the following `GetTemperature` method simulates a monitor that reads temperature data every five seconds and notifies observers if the temperature has changed by at least .1 degree since the previous reading. If the device does not report a temperature (that is, if its value is null), the provider notifies observers that the transmission is complete. Note that, in addition to

calling each observer's `OnCompleted` method, the `GetTemperature` method clears the `List<T>` collection. In this case, the provider makes no calls to the `OnError` method of its observers.

```
public void GetTemperature()
{
    // Create an array of sample data to mimic a temperature device.
    Nullable<Decimal>[] temps = {14.6m, 14.65m, 14.7m, 14.9m, 14.9m, 15.2m, 15.25m, 15.2m,
                                   15.4m, 15.45m, null };

    // Store the previous temperature, so notification is only sent after at least .1 change.
    Nullable<Decimal> previous = null;
    bool start = true;

    foreach (var temp in temps) {
        System.Threading.Thread.Sleep(2500);
        if (temp.HasValue) {
            if (start || (Math.Abs(temp.Value - previous.Value) >= 0.1m)) {
                Temperature tempData = new Temperature(temp.Value, DateTime.Now);
                foreach (var observer in observers)
                    observer.OnNext(tempData);
                previous = temp;
                if (start) start = false;
            }
        }
        else {
            foreach (var observer in observers.ToArray())
                if (observer != null) observer.OnCompleted();

            observers.Clear();
            break;
        }
    }
}
```

```
Public Sub GetTemperature()
    ' Create an array of sample data to mimic a temperature device.
    Dim temps() As Nullable(Of Decimal) = {14.6D, 14.65D, 14.7D, 14.9D, 14.9D, 15.2D, 15.25D, 15.2D,
                                           15.4D, 15.45D, Nothing}

    ' Store the previous temperature, so notification is only sent after at least .1 change.
    Dim previous As Nullable(Of Decimal)
    Dim start As Boolean = True

    For Each temp In temps
        System.Threading.Thread.Sleep(2500)

        If temp.HasValue Then
            If start OrElse Math.Abs(temp.Value - previous.Value) >= 0.1 Then
                Dim tempData As New Temperature(temp.Value, Date.Now)
                For Each observer In observers
                    observer.OnNext(tempData)
                Next
                previous = temp
                If start Then start = False
            End If
        Else
            For Each observer In observers.ToArray()
                If observer IsNot Nothing Then observer.OnCompleted()
            Next
            observers.Clear()
            Exit For
        End If
    Next
End Sub
```

Example

The following example contains the complete source code for defining an `IObservable<T>` implementation for a temperature monitoring application. It includes the `Temperature` structure, which is the data sent to observers, and the `TemperatureMonitor` class, which is the `IObservable<T>` implementation.

```
using System.Threading;
using System;
using System.Collections.Generic;

public class TemperatureMonitor : IObservable<Temperature>
{
    List<IObserver<Temperature>> observers;

    public TemperatureMonitor()
    {
        observers = new List<IObserver<Temperature>>();
    }

    private class Unsubscriber : IDisposable
    {
        private List<IObserver<Temperature>> _observers;
        private IObserver<Temperature> _observer;

        public Unsubscriber(List<IObserver<Temperature>> observers, IObserver<Temperature> observer)
        {
            this._observers = observers;
            this._observer = observer;
        }

        public void Dispose()
        {
            if (!(_observer == null)) _observers.Remove(_observer);
        }
    }

    public IDisposable Subscribe(IObserver<Temperature> observer)
    {
        if (!observers.Contains(observer))
            observers.Add(observer);

        return new Unsubscriber(observers, observer);
    }

    public void GetTemperature()
    {
        // Create an array of sample data to mimic a temperature device.
        Nullable<Decimal>[] temps = {14.6m, 14.65m, 14.7m, 14.9m, 14.9m, 15.2m, 15.25m, 15.2m,
                                       15.4m, 15.45m, null};

        // Store the previous temperature, so notification is only sent after at least .1 change.
        Nullable<Decimal> previous = null;
        bool start = true;

        foreach (var temp in temps) {
            System.Threading.Thread.Sleep(2500);
            if (temp.HasValue) {
                if (start || (Math.Abs(temp.Value - previous.Value) >= 0.1m)) {
                    Temperature tempData = new Temperature(temp.Value, DateTime.Now);
                    foreach (var observer in observers)
                        observer.OnNext(tempData);
                    previous = temp;
                    if (start) start = false;
                }
            }
            else {
                foreach (var observer in observers.ToArray())
                    if (observer != null) observer.OnCompleted();
            }
        }
    }
}
```

```
        if (observer == null) observer.OnCompleted(),  
            observers.Clear();  
        break;  
    }  
}  
}  
}
```

```

Imports System.Threading
Imports System.Collections.Generic

Public Class TemperatureMonitor : Implements IObservable(Of Temperature)
    Dim observers As List(Of IObserver(Of Temperature))

    Public Sub New()
        observers = New List(Of IObserver(Of Temperature))
    End Sub

    Private Class Unsubscriber : Implements IDisposable
        Private _observers As List(Of IObserver(Of Temperature))
        Private _observer As IObserver(Of Temperature)

        Public Sub New(ByVal observers As List(Of IObserver(Of Temperature)), ByVal observer As IObserver(Of Temperature))
            Me._observers = observers
            Me._observer = observer
        End Sub

        Public Sub Dispose() Implements IDisposable.Dispose
            If _observer IsNot Nothing Then _observers.Remove(_observer)
        End Sub
    End Class

    Public Function Subscribe(ByVal observer As System.IObserver(Of Temperature)) As System.IDisposable
        Implements System.IObservable(Of Temperature).Subscribe
        If Not observers.Contains(observer) Then
            observers.Add(observer)
        End If
        Return New Unsubscriber(observers, observer)
    End Function

    Public Sub GetTemperature()
        ' Create an array of sample data to mimic a temperature device.
        Dim temps() As Nullable(Of Decimal) = {14.6D, 14.65D, 14.7D, 14.9D, 14.9D, 15.2D, 15.25D, 15.2D,
                                                15.4D, 15.45D, Nothing}
        ' Store the previous temperature, so notification is only sent after at least .1 change.
        Dim previous As Nullable(Of Decimal)
        Dim start As Boolean = True

        For Each temp In temps
            System.Threading.Thread.Sleep(2500)

            If temp.HasValue Then
                If start OrElse Math.Abs(temp.Value - previous.Value) >= 0.1 Then
                    Dim tempData As New Temperature(temp.Value, Date.Now)
                    For Each observer In observers
                        observer.OnNext(tempData)
                    Next
                    previous = temp
                    If start Then start = False
                End If
            Else
                For Each observer In observers.ToArray()
                    If observer IsNot Nothing Then observer.OnCompleted()
                Next
                observers.Clear()
                Exit For
            End If
        Next
    End Sub
End Class

```

See also

- [IObservable<T>](#)
- [Observer Design Pattern](#)
- [How to: Implement an Observer](#)
- [Observer Design Pattern Best Practices](#)

How to: Implement an Observer

9/20/2022 • 3 minutes to read • [Edit Online](#)

The observer design pattern requires a division between an observer, which registers for notifications, and a provider, which monitors data and sends notifications to one or more observers. This topic discusses how to create an observer. A related topic, [How to: Implement a Provider](#), discusses how to create a provider.

To create an observer

1. Define the observer, which is a type that implements the [System.IObserver<T>](#) interface. For example, the following code defines a type named `TemperatureReporter` that is a constructed [System.IObserver<T>](#) implementation with a generic type argument of `Temperature`.

```
public class TemperatureReporter : IObserver<Temperature>
```

```
Public Class TemperatureReporter : Implements IObserver(Of Temperature)
```

2. If the observer can stop receiving notifications before the provider calls its [IObserver<T>.OnCompleted](#) implementation, define a private variable that will hold the [IDisposable](#) implementation returned by the provider's [IObservable<T>.Subscribe](#) method. You should also define a subscription method that calls the provider's [Subscribe](#) method and stores the returned [IDisposable](#) object. For example, the following code defines a private variable named `unsubscriber` and defines a `Subscribe` method that calls the provider's [Subscribe](#) method and assigns the returned object to the `unsubscriber` variable.

```
public class TemperatureReporter : IObserver<Temperature>
{
    private IDisposable unsubscriber;
    private bool first = true;
    private Temperature last;

    public virtual void Subscribe(IObservable<Temperature> provider)
    {
        unsubscriber = provider.Subscribe(this);
    }
}
```

```
Public Class TemperatureReporter : Implements IObserver(Of Temperature)

    Private unsubscriber As IDisposable
    Private first As Boolean = True
    Private last As Temperature

    Public Overridable Sub Subscribe(ByVal provider As IObservable(Of Temperature))
        unsubscriber = provider.Subscribe(Me)
    End Sub
}
```

3. Define a method that enables the observer to stop receiving notifications before the provider calls its [IObserver<T>.OnCompleted](#) implementation, if this feature is required. The following example defines an `Unsubscribe` method.

```
public virtual void Unsubscribe()
{
    subscriber.Dispose();
}
```

```
Public Overridable Sub Unsubscribe()
    unsubscriber.Dispose()
End Sub
```

4. Provide implementations of the three methods defined by the `IObserver<T>` interface:

`IObserver<T>.OnNext`, `IObserver<T>.OnError`, and `IObserver<T>.OnCompleted`. Depending on the provider and the needs of the application, the `OnError` and `OnCompleted` methods can be stub implementations. Note that the `OnError` method should not handle the passed `Exception` object as an exception, and the `OnCompleted` method is free to call the provider's `IDisposable.Dispose` implementation. The following example shows the `IObserver<T>` implementation of the `TemperatureReporter` class.

```
public virtual void OnCompleted()
{
    Console.WriteLine("Additional temperature data will not be transmitted.");
}

public virtual void OnError(Exception error)
{
    // Do nothing.
}

public virtual void OnNext(Temperature value)
{
    Console.WriteLine("The temperature is {0}°C at {1:g}", value.Degrees, value.Date);
    if (first)
    {
        last = value;
        first = false;
    }
    else
    {
        Console.WriteLine("    Change: {0}° in {1:g}", value.Degrees - last.Degrees,
                           value.Date.ToUniversalTime() -
last.Date.ToUniversalTime());
    }
}
```

```
Public Overridable Sub OnCompleted() Implements System.IObserver(Of Temperature).OnCompleted
    Console.WriteLine("Additional temperature data will not be transmitted.")
End Sub

Public Overridable Sub OnError(ByVal [error] As System.Exception) Implements System.IObserver(Of
Temperature).OnError
    ' Do nothing.
End Sub

Public Overridable Sub OnNext(ByVal value As Temperature) Implements System.IObserver(Of
Temperature).OnNext
    Console.WriteLine("The temperature is {0}°C at {1:g}", value.Degrees, value.Date)
    If first Then
        last = value
        first = False
    Else
        Console.WriteLine("    Change: {0}° in {1:g}", value.Degrees - last.Degrees,
                           value.Date.ToUniversalTime -
                           last.Date.ToUniversalTime)
    End If
End Sub
```

Example

The following example contains the complete source code for the `TemperatureReporter` class, which provides the `IObserver<T>` implementation for a temperature monitoring application.

```
public class TemperatureReporter : IObserver<Temperature>
{
    private IDisposable unsubscriber;
    private bool first = true;
    private Temperature last;

    public virtual void Subscribe(IObservable<Temperature> provider)
    {
        unsubscriber = provider.Subscribe(this);
    }

    public virtual void Unsubscribe()
    {
        unsubscriber.Dispose();
    }

    public virtual void OnCompleted()
    {
        Console.WriteLine("Additional temperature data will not be transmitted.");
    }

    public virtual void OnError(Exception error)
    {
        // Do nothing.
    }

    public virtual void OnNext(Temperature value)
    {
        Console.WriteLine("The temperature is {0}°C at {1:g}", value.Degrees, value.Date);
        if (first)
        {
            last = value;
            first = false;
        }
        else
        {
            Console.WriteLine("    Change: {0}° in {1:g}", value.Degrees - last.Degrees,
                value.Date.ToUniversalTime() -
                last.Date.ToUniversalTime());
        }
    }
}
```

```

Public Class TemperatureReporter : Implements IObservable(Of Temperature)

    Private unsubscriber As IDisposable
    Private first As Boolean = True
    Private last As Temperature

    Public Overridable Sub Subscribe(ByVal provider As IObservable(Of Temperature))
        unsubscriber = provider.Subscribe(Me)
    End Sub

    Public Overridable Sub Unsubscribe()
        unsubscriber.Dispose()
    End Sub

    Public Overridable Sub OnCompleted() Implements System.IObservable(Of Temperature).OnCompleted
        Console.WriteLine("Additional temperature data will not be transmitted.")
    End Sub

    Public Overridable Sub OnError(ByVal [error] As System.Exception) Implements System.IObservable(Of Temperature).OnError
        ' Do nothing.
    End Sub

    Public Overridable Sub OnNext(ByVal value As Temperature) Implements System.IObservable(Of Temperature).OnNext
        Console.WriteLine("The temperature is {0}°C at {1:g}", value.Degrees, value.Date)
        If first Then
            last = value
            first = False
        Else
            Console.WriteLine("    Change: {0}° in {1:g}", value.Degrees - last.Degrees,
                value.Date.ToUniversalTime -
                last.Date.ToUniversalTime)
        End If
    End Sub
End Class

```

See also

- [IObserver<T>](#)
- [Observer Design Pattern](#)
- [How to: Implement a Provider](#)
- [Observer Design Pattern Best Practices](#)

Handling and throwing exceptions in .NET

9/20/2022 • 2 minutes to read • [Edit Online](#)

Applications must be able to handle errors that occur during execution in a consistent manner. .NET provides a model for notifying applications of errors in a uniform way: .NET operations indicate failure by throwing exceptions.

Exceptions

An exception is any error condition or unexpected behavior that is encountered by an executing program. Exceptions can be thrown because of a fault in your code or in code that you call (such as a shared library), unavailable operating system resources, unexpected conditions that the runtime encounters (such as code that can't be verified), and so on. Your application can recover from some of these conditions, but not from others. Although you can recover from most application exceptions, you can't recover from most runtime exceptions.

In .NET, an exception is an object that inherits from the [System.Exception](#) class. An exception is thrown from an area of code where a problem has occurred. The exception is passed up the stack until the application handles it or the program terminates.

Exceptions vs. traditional error-handling methods

Traditionally, a language's error-handling model relied on either the language's unique way of detecting errors and locating handlers for them, or on the error-handling mechanism provided by the operating system. The way .NET implements exception handling provides the following advantages:

- Exception throwing and handling works the same for .NET programming languages.
- Doesn't require any particular language syntax for handling exceptions, but allows each language to define its own syntax.
- Exceptions can be thrown across process and even machine boundaries.
- Exception-handling code can be added to an application to increase program reliability.

Exceptions offer advantages over other methods of error notification, such as return codes. Failures don't go unnoticed because if an exception is thrown and you don't handle it, the runtime terminates your application. Invalid values don't continue to propagate through the system as a result of code that fails to check for a failure return code.

Common exceptions

The following table lists some common exceptions with examples of what can cause them.

EXCEPTION TYPE	DESCRIPTION	EXAMPLE
Exception	Base class for all exceptions.	None (use a derived class of this exception).
IndexOutOfRangeException	Thrown by the runtime only when an array is indexed improperly.	Indexing an array outside its valid range: <code>arr[arr.Length+1]</code>

Exception Type	Description	Example
NullReferenceException	Thrown by the runtime only when a null object is referenced.	<pre>object o = null; o.ToString();</pre>
InvalidOperationException	Thrown by methods when in an invalid state.	Calling <code>Enumerator.MoveNext()</code> after removing an item from the underlying collection.
ArgumentException	Base class for all argument exceptions.	None (use a derived class of this exception).
ArgumentNullException	Thrown by methods that do not allow an argument to be null.	<pre>String s = null; "Calculate".IndexOf(s);</pre>
ArgumentOutOfRangeException	Thrown by methods that verify that arguments are in a given range.	<pre>String s = "string"; s.Substring(s.Length+1);</pre>

See also

- [Exception Class and Properties](#)
- [How to: Use the Try-Catch Block to Catch Exceptions](#)
- [How to: Use Specific Exceptions in a Catch Block](#)
- [How to: Explicitly Throw Exceptions](#)
- [How to: Create User-Defined Exceptions](#)
- [Using User-Filtered Exception Handlers](#)
- [How to: Use Finally Blocks](#)
- [Handling COM Interop Exceptions](#)
- [Best Practices for Exceptions](#)
- [What Every Dev needs to Know About Exceptions in the Runtime](#)

Exception class and properties

9/20/2022 • 2 minutes to read • [Edit Online](#)

The [Exception](#) class is the base class from which exceptions inherit. For example, the [InvalidOperationException](#) class hierarchy is as follows:

```
Object
  |
  Exception
    |
    SystemException
      |
      InvalidOperationException
```

The [Exception](#) class has the following properties that help make understanding an exception easier.

PROPERTY NAME	DESCRIPTION
Data	An IDictionary that holds arbitrary data in key-value pairs.
HelpLink	Can hold a URL (or URN) to a help file that provides extensive information about the cause of an exception.
InnerException	This property can be used to create and preserve a series of exceptions during exception handling. You can use it to create a new exception that contains previously caught exceptions. The original exception can be captured by the second exception in the InnerException property, allowing code that handles the second exception to examine the additional information. For example, suppose you have a method that receives an argument that's improperly formatted. The code tries to read the argument, but an exception is thrown. The method catches the exception and throws a FormatException . To improve the caller's ability to determine the reason an exception is thrown, it is sometimes desirable for a method to catch an exception thrown by a helper routine and then throw an exception more indicative of the error that has occurred. A new and more meaningful exception can be created, where the inner exception reference can be set to the original exception. This more meaningful exception can then be thrown to the caller. Note that with this functionality, you can create a series of linked exceptions that ends with the exception that was thrown first.
Message	Provides details about the cause of an exception.
Source	Gets or sets the name of the application or the object that causes the error.
StackTrace	Contains a stack trace that can be used to determine where an error occurred. The stack trace includes the source file name and program line number if debugging information is available.

Most of the classes that inherit from [Exception](#) do not implement additional members or provide additional functionality; they simply inherit from [Exception](#). Therefore, the most important information for an exception can be found in the hierarchy of exception classes, the exception name, and the information contained in the

exception.

We recommend that you throw and catch only objects that derive from [Exception](#), but you can throw any object that derives from the [Object](#) class as an exception. Note that not all languages support throwing and catching objects that do not derive from [Exception](#).

See also

- [Exceptions](#)

How to use the try/catch block to catch exceptions

9/20/2022 • 2 minutes to read • [Edit Online](#)

Place any code statements that might raise or throw an exception in a `try` block, and place statements used to handle the exception or exceptions in one or more `catch` blocks below the `try` block. Each `catch` block includes the exception type and can contain additional statements needed to handle that exception type.

In the following example, a `StreamReader` opens a file called `data.txt` and retrieves a line from the file. Since the code might throw any of three exceptions, it's placed in a `try` block. Three `catch` blocks catch the exceptions and handle them by displaying the results to the console.

```
using namespace System;
using namespace System::IO;

public ref class ProcessFile
{
public:
    static void Main()
    {
        try
        {
            StreamReader^ sr = File::OpenText("data.txt");
            Console::WriteLine("The first line of this file is {0}", sr->ReadLine());
            sr->Close();
        }
        catch (Exception^ e)
        {
            Console::WriteLine("An error occurred: '{0}'", e);
        }
    }
};

int main()
{
    ProcessFile::Main();
}
```

```

using System;
using System.IO;

public class ProcessFile
{
    public static void Main()
    {
        try
        {
            using (StreamReader sr = File.OpenText("data.txt"))
            {
                Console.WriteLine($"The first line of this file is {sr.ReadLine()}");
            }
        }
        catch (FileNotFoundException e)
        {
            Console.WriteLine($"The file was not found: '{e}'");
        }
        catch (DirectoryNotFoundException e)
        {
            Console.WriteLine($"The directory was not found: '{e}'");
        }
        catch (IOException e)
        {
            Console.WriteLine($"The file could not be opened: '{e}'");
        }
    }
}

```

```

Imports System.IO

Public Class ProcessFile
    Public Shared Sub Main()
        Try
            Using sr As StreamReader = File.OpenText("data.txt")
                Console.WriteLine($"The first line of this file is {sr.ReadLine()}")
            End Using
        Catch e As FileNotFoundException
            Console.WriteLine($"The file was not found: '{e}'")
        Catch e As DirectoryNotFoundException
            Console.WriteLine($"The directory was not found: '{e}'")
        Catch e As IOException
            Console.WriteLine($"The file could not be opened: '{e}'")
        End Try
    End Sub
End Class

```

The Common Language Runtime (CLR) catches exceptions not handled by `catch` blocks. If an exception is caught by the CLR, one of the following results may occur depending on your CLR configuration:

- A **Debug** dialog box appears.
- The program stops execution and a dialog box with exception information appears.
- An error prints out to the [standard error output stream](#).

NOTE

Most code can throw an exception, and some exceptions, like [OutOfMemoryException](#), can be thrown by the CLR itself at any time. While applications aren't required to deal with these exceptions, be aware of the possibility when writing libraries to be used by others. For suggestions on when to set code in a `try` block, see [Best Practices for Exceptions](#).

See also

- [Exceptions](#)
- [Handling I/O errors in .NET](#)

How to use specific exceptions in a catch block

9/20/2022 • 2 minutes to read • [Edit Online](#)

In general, it's good programming practice to catch a specific type of exception rather than use a basic `catch` statement.

When an exception occurs, it is passed up the stack and each catch block is given the opportunity to handle it. The order of catch statements is important. Put catch blocks targeted to specific exceptions before a general exception catch block or the compiler might issue an error. The proper catch block is determined by matching the type of the exception to the name of the exception specified in the catch block. If there is no specific catch block, the exception is caught by a general catch block, if one exists.

The following code example uses a `try / catch` block to catch an `InvalidOperationException`. The sample creates a class called `Employee` with a single property, employee level (`Emlevel`). A method, `PromoteEmployee`, takes an object and increments the employee level. An `InvalidOperationException` occurs when a `DateTime` instance is passed to the `PromoteEmployee` method.

```

using namespace System;

public ref class Employee
{
public:
    Employee()
    {
        emlevel = 0;
    }

    //Create employee level property.
    property int Emlevel
    {
        int get()
        {
            return emlevel;
        }
        void set(int value)
        {
            emlevel = value;
        }
    }
}

private:
    int emlevel;
};

public ref class Ex13
{
public:
    static void PromoteEmployee(Object^ emp)
    {
        //Cast object to Employee.
        Employee^ e = (Employee^) emp;
        // Increment employee level.
        e->Emlevel++;
    }

    static void Main()
    {
        try
        {
            Object^ o = gcnew Employee();
            DateTime^ newyears = gcnew DateTime(2001, 1, 1);
            //Promote the new employee.
            PromoteEmployee(o);
            //Promote DateTime; results in InvalidCastException as newyears is not an employee instance.
            PromoteEmployee(newyears);
        }
        catch (InvalidCastException^ e)
        {
            Console::WriteLine("Error passing data to PromoteEmployee method. " + e->Message);
        }
    }
};

int main()
{
    Ex13::Main();
}

```

```
using System;

public class Employee
{
    //Create employee level property.
    public int Emlevel
    {
        get
        {
            return(emlevel);
        }
        set
        {
            emlevel = value;
        }
    }

    private int emlevel = 0;
}

public class Ex13
{
    public static void PromoteEmployee(Object emp)
    {
        // Cast object to Employee.
        var e = (Employee) emp;
        // Increment employee level.
        e.Emlevel = e.Emlevel + 1;
    }

    static void Main()
    {
        try
        {
            Object o = new Employee();
            DateTime newYears = new DateTime(2001, 1, 1);
            // Promote the new employee.
            PromoteEmployee(o);
            // Promote DateTime; results in InvalidCastException as newYears is not an employee instance.
            PromoteEmployee(newYears);
        }
        catch (InvalidCastException e)
        {
            Console.WriteLine("Error passing data to PromoteEmployee method. " + e.Message);
        }
    }
}
```

```

Public Class Employee
    'Create employee level property.
    Public Property Emlevel As Integer
        Get
            Return emlevelValue
        End Get
        Set
            emlevelValue = Value
        End Set
    End Property

    Private emlevelValue As Integer = 0
End Class

Public Class Ex13
    Public Shared Sub PromoteEmployee(emp As Object)
        ' Cast object to Employee.
        Dim e As Employee = CType(emp, Employee)
        ' Increment employee level.
        e.Emlevel = e.Emlevel + 1
    End Sub

    Public Shared Sub Main()
        Try
            Dim o As Object = New Employee()
            Dim newYears As New DateTime(2001, 1, 1)
            ' Promote the new employee.
            PromoteEmployee(o)
            ' Promote DateTime; results in InvalidCastException as newYears is not an employee instance.
            PromoteEmployee(newYears)
        Catch e As InvalidCastException
            Console.WriteLine("Error passing data to PromoteEmployee method. " + e.Message)
        End Try
    End Sub
End Class

```

See also

- [Exceptions](#)

How to explicitly throw exceptions

9/20/2022 • 2 minutes to read • [Edit Online](#)

You can explicitly throw an exception using the C# `throw` or the Visual Basic `Throw` statement. You can also throw a caught exception again using the `throw` statement. It is good coding practice to add information to an exception that is re-thrown to provide more information when debugging.

The following code example uses a `try / catch` block to catch a possible `FileNotFoundException`. Following the `try` block is a `catch` block that catches the `FileNotFoundException` and writes a message to the console if the data file is not found. The next statement is the `throw` statement that throws a new `FileNotFoundException` and adds text information to the exception.

```
using System;
using System.IO;

public class ProcessFile
{
    public static void Main()
    {
        FileStream fs;
        try
        {
            // Opens a text file.
            fs = new FileStream(@"C:\temp\data.txt", FileMode.Open);
            var sr = new StreamReader(fs);

            // A value is read from the file and output to the console.
            string line = sr.ReadLine();
            Console.WriteLine(line);
        }
        catch(FileNotFoundException e)
        {
            Console.WriteLine($"[Data File Missing] {e}");
            throw new FileNotFoundException("[data.txt not in c:\temp directory]", e);
        }
        finally
        {
            if (fs != null)
                fs.Close();
        }
    }
}
```

```
Option Strict On

Imports System.IO

Public Class ProcessFile

    Public Shared Sub Main()
        Dim fs As FileStream
        Try
            ' Opens a text file.
            fs = New FileStream("c:\temp\data.txt", FileMode.Open)
            Dim sr As New StreamReader(fs)

            ' A value is read from the file and output to the console.
            Dim line As String = sr.ReadLine()
            Console.WriteLine(line)
        Catch e As FileNotFoundException
            Console.WriteLine($"[Data File Missing] {e}")
            Throw New FileNotFoundException("[data.txt not in c:\temp directory]", e)
        Finally
            If fs IsNot Nothing Then fs.Close()
        End Try
    End Sub
End Class
```

See also

- [Exceptions](#)

How to create user-defined exceptions

9/20/2022 • 2 minutes to read • [Edit Online](#)

.NET provides a hierarchy of exception classes ultimately derived from the [Exception](#) base class. However, if none of the predefined exceptions meet your needs, you can create your own exception classes by deriving from the [Exception](#) class.

When creating your own exceptions, end the class name of the user-defined exception with the word "Exception", and implement the three common constructors, as shown in the following example. The example defines a new exception class named `EmployeeListNotFoundException`. The class is derived from the [Exception](#) base class and includes three constructors.

```
using namespace System;

public ref class EmployeeListNotFoundException : Exception
{
public:
    EmployeeListNotFoundException()
    {
    }

    EmployeeListNotFoundException(String^ message)
        : Exception(message)
    {
    }

    EmployeeListNotFoundException(String^ message, Exception^ inner)
        : Exception(message, inner)
    {
    }
};
```

```
using System;

public class EmployeeListNotFoundException : Exception
{
    public EmployeeListNotFoundException()
    {
    }

    public EmployeeListNotFoundException(string message)
        : base(message)
    {
    }

    public EmployeeListNotFoundException(string message, Exception inner)
        : base(message, inner)
    {
    }
}
```

```
Public Class EmployeeListNotFoundException
    Inherits Exception

    Public Sub New()
        End Sub

    Public Sub New(message As String)
        MyBase.New(message)
        End Sub

    Public Sub New(message As String, inner As Exception)
        MyBase.New(message, inner)
        End Sub
End Class
```

NOTE

In situations where you're using remoting, you must ensure that the metadata for any user-defined exceptions is available at the server (callee) and to the client (the proxy object or caller). For more information, see [Best practices for exceptions](#).

See also

- [Exceptions](#)

How to create user-defined exceptions with localized exception messages

9/20/2022 • 2 minutes to read • [Edit Online](#)

In this article, you will learn how to create user-defined exceptions that are inherited from the base `Exception` class with localized exception messages using satellite assemblies.

Create custom exceptions

.NET contains many different exceptions that you can use. However, in some cases when none of them meets your needs, you can create your own custom exceptions.

Let's assume you want to create a `StudentNotFoundException` that contains a `StudentName` property. To create a custom exception, follow these steps:

1. Create a serializable class that inherits from `Exception`. The class name should end in "Exception":

```
[Serializable]
public class StudentNotFoundException : Exception { }
```

```
<Serializable>
Public Class StudentNotFoundException
    Inherits Exception
End Class
```

2. Add the default constructors:

```
[Serializable]
public class StudentNotFoundException : Exception
{
    public StudentNotFoundException() { }

    public StudentNotFoundException(string message)
        : base(message) { }

    public StudentNotFoundException(string message, Exception inner)
        : base(message, inner) { }
}
```

```

<Serializable>
Public Class StudentNotFoundException
    Inherits Exception

    Public Sub New()
    End Sub

    Public Sub New(message As String)
        MyBase.New(message)
    End Sub

    Public Sub New(message As String, inner As Exception)
        MyBase.New(message, inner)
    End Sub
End Class

```

3. Define any additional properties and constructors:

```

[Serializable]
public class StudentNotFoundException : Exception
{
    public string StudentName { get; }

    public StudentNotFoundException() { }

    public StudentNotFoundException(string message)
        : base(message) { }

    public StudentNotFoundException(string message, Exception inner)
        : base(message, inner) { }

    public StudentNotFoundException(string message, string studentName)
        : this(message)
    {
        StudentName = studentName;
    }
}

```

```

<Serializable>
Public Class StudentNotFoundException
    Inherits Exception

    Public ReadOnly Property StudentName As String

    Public Sub New()
    End Sub

    Public Sub New(message As String)
        MyBase.New(message)
    End Sub

    Public Sub New(message As String, inner As Exception)
        MyBase.New(message, inner)
    End Sub

    Public Sub New(message As String, studentName As String)
        Me.New(message)
        StudentName = studentName
    End Sub
End Class

```

Create localized exception messages

You have created a custom exception, and you can throw it anywhere with code like the following:

```
throw new StudentNotFoundException("The student cannot be found.", "John");
```

```
Throw New StudentNotFoundException("The student cannot be found.", "John")
```

The problem with the previous line is that "The student cannot be found." is just a constant string. In a localized application, you want to have different messages depending on user culture. **Satellite assemblies** are a good way to do that. A satellite assembly is a .dll that contains resources for a specific language. When you ask for a specific resources at run time, the CLR finds that resource depending on user culture. If no satellite assembly is found for that culture, the resources of the default culture are used.

To create the localized exception messages:

1. Create a new folder named *Resources* to hold the resource files.
2. Add a new resource file to it. To do that in Visual Studio, right-click the folder in **Solution Explorer**, and select **Add > New Item > Resources File**. Name the file *ExceptionMessages.resx*. This is the default resources file.
3. Add a name/value pair for your exception message, like the following image shows:

Name	Value
StudentNotFound	The student cannot be found.
*	

4. Add a new resource file for French. Name it *ExceptionMessages.fr-FR.resx*.
5. Add a name/value pair for the exception message again, but with a French value:

Name	Value
StudentNotFound	L'étudiant est introuvable.
*	

6. After you build the project, the build output folder should contain the *fr-FR* folder with a *.dll* file, which is the satellite assembly.
7. You throw the exception with code like the following:

```
var resourceManager = new ResourceManager("FULLY_QUALIFIED_NAME_OF_RESOURCE_FILE",
Assembly.GetExecutingAssembly());
throw new StudentNotFoundException(resourceManager.GetString("StudentNotFound"), "John");
```

```
Dim resourceManager As New ResourceManager("FULLY_QUALIFIED_NAME_OF_RESOURCE_FILE",
Assembly.GetExecutingAssembly())
Throw New StudentNotFoundException(resourceManager.GetString("StudentNotFound"), "John")
```

NOTE

If the project name is `TestProject` and the resource file *ExceptionMessages.resx* resides in the *Resources* folder of the project, the fully qualified name of the resource file is `TestProject.Resources.ExceptionMessages`.

See also

- [How to create user-defined exceptions](#)
- [Create satellite assemblies](#)
- [base \(C# Reference\)](#)
- [this \(C# Reference\)](#)

How to use finally blocks

9/20/2022 • 2 minutes to read • [Edit Online](#)

When an exception occurs, execution stops and control is given to the appropriate exception handler. This often means that lines of code you expect to be executed are bypassed. Some resource cleanup, such as closing a file, needs to be done even if an exception is thrown. To do this, you can use a `finally` block. A `finally` block always executes, regardless of whether an exception is thrown.

The following code example uses a `try / catch` block to catch an `ArgumentOutOfRangeException`. The `Main` method creates two arrays and attempts to copy one to the other. The action generates an `ArgumentOutOfRangeException` and the error is written to the console. The `finally` block executes regardless of the outcome of the copy action.

```
using namespace System;

ref class ArgumentOutOfRangeExceptionExample
{
public:
    static void Main()
    {
        array<int>^ array1 = {0, 0};
        array<int>^ array2 = {0, 0};

        try
        {
            Array::Copy(array1, array2, -1);
        }
        catch (ArgumentOutOfRangeException^ e)
        {
            Console::WriteLine("Error: {0}", e);
            throw;
        }
        finally
        {
            Console::WriteLine("This statement is always executed.");
        }
    }
};

int main()
{
    ArgumentOutOfRangeExceptionExample::Main();
}
```

```
using System;

class ArgumentOutOfRangeExceptionExample
{
    public static void Main()
    {
        int[] array1 = {0, 0};
        int[] array2 = {0, 0};

        try
        {
            Array.Copy(array1, array2, -1);
        }
        catch (ArgumentOutOfRangeException e)
        {
            Console.WriteLine("Error: {0}", e);
            throw;
        }
        finally
        {
            Console.WriteLine("This statement is always executed.");
        }
    }
}
```

```
Class ArgumentOutOfRangeExceptionExample
    Public Shared Sub Main()
        Dim array1() As Integer = {0, 0}
        Dim array2() As Integer = {0, 0}

        Try
            Array.Copy(array1, array2, -1)
        Catch e As ArgumentOutOfRangeException
            Console.WriteLine("Error: {0}", e)
            Throw
        Finally
            Console.WriteLine("This statement is always executed.")
        End Try
    End Sub
End Class
```

See also

- [Exceptions](#)

Use user-filtered exception handlers

9/20/2022 • 2 minutes to read • [Edit Online](#)

User-filtered exception handlers catch and handle exceptions based on requirements you define for the exception. These handlers use the `catch` statement with the `when` keyword (`Catch` and `When` in Visual Basic).

This technique is useful when a particular exception object corresponds to multiple errors. In this case, the object typically has a property that contains the specific error code associated with the error. You can use the error code property in the expression to select only the particular error you want to handle in that `catch` clause.

The following example illustrates the `catch / when` statement.

```
try
{
    //Try statements.
}
catch (Exception ex) when (ex.Message.Contains("404"))
{
    //Catch statements.
}
```

```
Try
    'Try statements.
    Catch When Err = VBErr_ClassLoadException
        'Catch statements.
End Try
```

The expression of the user-filtered clause is not restricted in any way. If an exception occurs during execution of the user-filtered expression, that exception is discarded and the filter expression is considered to have evaluated to false. In this case, the common language runtime continues the search for a handler for the current exception.

Combine the specific exception and the user-filtered clauses

A `catch` statement can contain both the specific exception and the user-filtered clauses. The runtime tests the specific exception first. If the specific exception succeeds, the runtime executes the user filter. The generic filter can contain a reference to the variable declared in the class filter. Note that the order of the two filter clauses cannot be reversed.

The following example shows a specific exception in the `catch` statement as well as the user-filtered clause using the `when` keyword.

```
try
{
    //Try statements.
}
catch (System.Net.Http.HttpRequestException ex) when (ex.Message.Contains("404"))
{
    //Catch statements.
}
```

```
Try
    'Try statements.
    Catch cle As ClassLoadException When cle.IsRecoverable()
        'Catch statements.
    End Try
```

See also

- [Exceptions](#)

Handling COM Interop Exceptions

9/20/2022 • 2 minutes to read • [Edit Online](#)

Managed and unmanaged code can work together to handle exceptions. If a method throws an exception in managed code, the common language runtime can pass an HRESULT to a COM object. If a method fails in unmanaged code by returning a failure HRESULT, the runtime throws an exception that can be caught by managed code.

The runtime automatically maps the HRESULT from COM interop to more specific exceptions. For example, E_ACCESSDENIED becomes [UnauthorizedAccessException](#), E_OUTOFMEMORY becomes [OutOfMemoryException](#), and so on.

If the HRESULT is a custom result or if it is unknown to the runtime, the runtime passes a generic [COMException](#) to the client. The [ErrorCode](#) property of the [COMException](#) contains the HRESULT value.

Working with IErrorInfo

When an error is passed from COM to managed code, the runtime populates the exception object with error information. COM objects that support [IErrorInfo](#) and return HRESULTS provide this information to managed code exceptions. For example, the runtime maps the Description from the COM error to the exception's [Message](#) property. If the HRESULT provides no additional error information, the runtime fills many of the exception's properties with default values.

If a method fails in unmanaged code, an exception can be passed to a managed code segment. The topic [HRESULTS and Exceptions](#) contains a table showing how HRESULTS map to runtime exception objects.

See also

- [Exceptions](#)

Best practices for exceptions

9/20/2022 • 10 minutes to read • [Edit Online](#)

A well-designed app handles exceptions and errors to prevent app crashes. This section describes best practices for handling and creating exceptions.

Use try/catch/finally blocks to recover from errors or release resources

Use `try / catch` blocks around code that can potentially generate an exception, and your code can recover from that exception. In `catch` blocks, always order exceptions from the most derived to the least derived. All exceptions derive from the [Exception](#) class. More derived exceptions aren't handled by a catch clause that's preceded by a catch clause for a base exception class. When your code can't recover from an exception, don't catch that exception. Enable methods further up the call stack to recover if possible.

Clean up resources that are allocated with either `using` statements or `finally` blocks. Prefer `using` statements to automatically clean up resources when exceptions are thrown. Use `finally` blocks to clean up resources that don't implement [IDisposable](#). Code in a `finally` clause is almost always executed even when exceptions are thrown.

Handle common conditions without throwing exceptions

For conditions that are likely to occur but might trigger an exception, consider handling them in a way that will avoid the exception. For example, if you try to close a connection that's already closed, you'll get an `InvalidOperationException`. You can avoid that by using an `if` statement to check the connection state before trying to close it.

```
if (conn->State != ConnectionState::Closed)
{
    conn->Close();
}
```

```
if (conn.State != ConnectionState.Closed)
{
    conn.Close();
}
```

```
If conn.State <> ConnectionState.Closed Then
    conn.Close()
End IF
```

If you don't check the connection state before closing, you can catch the `InvalidOperationException` exception.

```
try
{
    conn->Close();
}
catch (InvalidOperationException^ ex)
{
    Console::WriteLine(ex->GetType()->FullName);
    Console::WriteLine(ex->Message);
}
```

```
try
{
    conn.Close();
}
catch (InvalidOperationException ex)
{
    Console.WriteLine(ex.GetType().FullName);
    Console.WriteLine(ex.Message);
}
```

```
Try
    conn.Close()
Catch ex As InvalidOperationException
    Console.WriteLine(ex.GetType().FullName)
    Console.WriteLine(ex.Message)
End Try
```

The method to choose depends on how often you expect the event to occur.

- Use exception handling if the event doesn't occur often, that is, if the event is truly exceptional and indicates an error, such as an unexpected end-of-file. When you use exception handling, less code is executed in normal conditions.
- Check for error conditions in code if the event happens routinely and could be considered part of normal execution. When you check for common error conditions, less code is executed because you avoid exceptions.

Design classes so that exceptions can be avoided

A class can provide methods or properties that enable you to avoid making a call that would trigger an exception. For example, a [FileStream](#) class provides methods that help determine whether the end of the file has been reached. These methods can be used to avoid the exception that's thrown if you read past the end of the file. The following example shows how to read to the end of a file without triggering an exception:

```
class FileRead
{
public:
    void ReadAll(FileStream^ fileToRead)
    {
        // This if statement is optional
        // as it is very unlikely that
        // the stream would ever be null.
        if (fileToRead == nullptr)
        {
            throw gcnew System::ArgumentNullException();
        }

        int b;

        // Set the stream position to the beginning of the file.
        fileToRead->Seek(0, SeekOrigin::Begin);

        // Read each byte to the end of the file.
        for (int i = 0; i < fileToRead->Length; i++)
        {
            b = fileToRead->ReadByte();
            Console::Write(b.ToString());
            // Or do something else with the byte.
        }
    }
};
```

```
class FileRead
{
    public void ReadAll(FileStream fileToRead)
    {
        // This if statement is optional
        // as it is very unlikely that
        // the stream would ever be null.
        if (fileToRead == null)
        {
            throw new ArgumentNullException();
        }

        int b;

        // Set the stream position to the beginning of the file.
        fileToRead.Seek(0, SeekOrigin.Begin);

        // Read each byte to the end of the file.
        for (int i = 0; i < fileToRead.Length; i++)
        {
            b = fileToRead.ReadByte();
            Console.WriteLine(b.ToString());
            // Or do something else with the byte.
        }
    }
}
```

```

Class FileRead
    Public Sub ReadAll(fileToRead As FileStream)
        ' This if statement is optional
        ' as it is very unlikely that
        ' the stream would ever be null.
        If fileToRead Is Nothing Then
            Throw New System.ArgumentNullException()
        End If

        Dim b As Integer

        ' Set the stream position to the beginning of the file.
        fileToRead.Seek(0, SeekOrigin.Begin)

        ' Read each byte to the end of the file.
        For i As Integer = 0 To fileToRead.Length - 1
            b = fileToRead.ReadByte()
            Console.WriteLine(b.ToString())
            ' Or do something else with the byte.
        Next i
    End Sub
End Class

```

Another way to avoid exceptions is to return null (or default) for most common error cases instead of throwing an exception. A common error case can be considered a normal flow of control. By returning null (or default) in these cases, you minimize the performance impact to an app.

For value types, whether to use `Nullable<T>` or default as your error indicator is something to consider for your app. By using `Nullable<Guid>`, `default` becomes `null` instead of `Guid.Empty`. Sometimes, adding `Nullable<T>` can make it clearer when a value is present or absent. Other times, adding `Nullable<T>` can create extra cases to check that aren't necessary and only serve to create potential sources of errors.

Throw exceptions instead of returning an error code

Exceptions ensure that failures don't go unnoticed because the calling code didn't check a return code.

Use the predefined .NET exception types

Introduce a new exception class only when a predefined one doesn't apply. For example:

- Throw an `InvalidOperationException` exception if a property set or method call isn't appropriate given the object's current state.
- Throw an `ArgumentException` exception or one of the predefined classes that derive from `ArgumentException` if invalid parameters are passed.

End exception class names with the word `Exception`

When a custom exception is necessary, name it appropriately and derive it from the `Exception` class. For example:

```

public ref class MyFileNotFoundException : public Exception
{
};

```

```
public class MyFileNotFoundException : Exception
{
}
```

```
Public Class MyFileNotFoundException
    Inherits Exception
End Class
```

Include three constructors in custom exception classes

Use at least the three common constructors when creating your own exception classes: the parameterless constructor, a constructor that takes a string message, and a constructor that takes a string message and an inner exception.

- [Exception\(\)](#), which uses default values.
- [Exception\(String\)](#), which accepts a string message.
- [Exception\(String, Exception\)](#), which accepts a string message and an inner exception.

For an example, see [How to: Create User-Defined Exceptions](#).

Ensure that exception data is available when code executes remotely

When you create user-defined exceptions, ensure that the metadata for the exceptions is available to code that's executing remotely.

For example, on .NET implementations that support App Domains, exceptions might occur across App domains. Suppose App Domain A creates App Domain B, which executes code that throws an exception. For App Domain A to properly catch and handle the exception, it must be able to find the assembly that contains the exception thrown by App Domain B. If App Domain B throws an exception that is contained in an assembly under its application base, but not under App Domain A's application base, App Domain A won't be able to find the exception, and the common language runtime will throw a [FileNotFoundException](#) exception. To avoid this situation, you can deploy the assembly that contains the exception information in either of two ways:

- Put the assembly into a common application base shared by both app domains.
- If the domains don't share a common application base, sign the assembly that contains the exception information with a strong name and deploy the assembly into the global assembly cache.

Use grammatically correct error messages

Write clear sentences and include ending punctuation. Each sentence in the string assigned to the [Exception.Message](#) property should end in a period. For example, "The log table has overflowed." would be an appropriate message string.

Include a localized string message in every exception

The error message the user sees is derived from the [Exception.Message](#) property of the exception that was thrown, and not from the name of the exception class. Typically, you assign a value to the [Exception.Message](#) property by passing the message string to the `message` argument of an [Exception constructor](#).

For localized applications, you should provide a localized message string for every exception that your application can throw. You use resource files to provide localized error messages. For information on localizing applications and retrieving localized strings, see the following articles:

- [How to: create user-defined exceptions with localized exception messages](#)
- [Resources in .NET Apps](#)
- [System.Resources.ResourceManager](#)

In custom exceptions, provide additional properties as needed

Provide additional properties for an exception (in addition to the custom message string) only when there's a programmatic scenario where the additional information is useful. For example, the [FileNotFoundException](#) provides the [FileName](#) property.

Place throw statements so that the stack trace will be helpful

The stack trace begins at the statement where the exception is thrown and ends at the `catch` statement that catches the exception.

Use exception builder methods

It's common for a class to throw the same exception from different places in its implementation. To avoid excessive code, use helper methods that create the exception and return it. For example:

```
ref class FileReader
{
private:
    String^ fileName;

public:
    FileReader(String^ path)
    {
        fileName = path;
    }

    array<Byte>^ Read(int bytes)
    {
        array<Byte>^ results = FileUtils::ReadFromFile(fileName, bytes);
        if (results == nullptr)
        {
            throw NewFileIOException();
        }
        return results;
    }

    FileReaderException^ NewFileIOException()
    {
        String^ description = "My NewFileIOException Description";

        return gcnew FileReaderException(description);
    }
};
```

```

class FileReader
{
    private string fileName;

    public FileReader(string path)
    {
        fileName = path;
    }

    public byte[] Read(int bytes)
    {
        byte[] results = FileUtils.ReadFromFile(fileName, bytes);
        if (results == null)
        {
            throw NewFileIOException();
        }
        return results;
    }

    FileReaderException NewFileIOException()
    {
        string description = "My NewFileIOException Description";

        return new FileReaderException(description);
    }
}

```

```

Class FileReader
    Private fileName As String

    Public Sub New(path As String)
        fileName = path
    End Sub

    Public Function Read(bytes As Integer) As Byte()
        Dim results() As Byte = FileUtils.ReadFromFile(fileName, bytes)
        If results Is Nothing
            Throw NewFileIOException()
        End If
        Return results
    End Function

    Function NewFileIOException() As FileReaderException
        Dim description As String = "My NewFileIOException Description"

        Return New FileReaderException(description)
    End Function
End Class

```

In some cases, it's more appropriate to use the exception's constructor to build the exception. An example is a global exception class such as [ArgumentException](#).

Restore state when methods don't complete due to exceptions

Callers should be able to assume that there are no side effects when an exception is thrown from a method. For example, if you have code that transfers money by withdrawing from one account and depositing in another account, and an exception is thrown while executing the deposit, you don't want the withdrawal to remain in effect.

```

public void TransferFunds(Account from, Account to, decimal amount)
{
    from.Withdrawal(amount);
    // If the deposit fails, the withdrawal shouldn't remain in effect.
    to.Deposit(amount);
}

```

```

Public Sub TransferFunds(from As Account, [to] As Account, amount As Decimal)
    from.Withdrawal(amount)
    ' If the deposit fails, the withdrawal shouldn't remain in effect.
    [to].Deposit(amount)
End Sub

```

The preceding method doesn't directly throw any exceptions. However, you must write the method so that the withdrawal is reversed if the deposit operation fails.

One way to handle this situation is to catch any exceptions thrown by the deposit transaction and roll back the withdrawal.

```

private static void TransferFunds(Account from, Account to, decimal amount)
{
    string withdrawalTrxID = from.Withdrawal(amount);
    try
    {
        to.Deposit(amount);
    }
    catch
    {
        from.RollbackTransaction(withdrawalTrxID);
        throw;
    }
}

```

```

Private Shared Sub TransferFunds(from As Account, [to] As Account, amount As Decimal)
    Dim withdrawalTrxID As String = from.Withdrawal(amount)
    Try
        [to].Deposit(amount)
    Catch
        from.RollbackTransaction(withdrawalTrxID)
        Throw
    End Try
End Sub

```

This example illustrates the use of `throw` to re-throw the original exception, making it easier for callers to see the real cause of the problem without having to examine the `InnerException` property. An alternative is to throw a new exception and include the original exception as the inner exception.

```

catch (Exception ex)
{
    from.RollbackTransaction(withdrawalTrxID);
    throw new TransferFundsException("Withdrawal failed.", innerException: ex)
    {
        From = from,
        To = to,
        Amount = amount
    };
}

```

```
Catch ex As Exception
    from.RollbackTransaction(withdrawalTrxID)
    Throw New TransferFundsException("Withdrawal failed.", innerException:=ex) With
    {
        .From = from,
        .[To] = [to],
        .Amount = amount
    }
End Try
```

See also

- [Exceptions](#)

Numerics in .NET

9/20/2022 • 4 minutes to read • [Edit Online](#)

.NET provides a range of numeric integer and floating-point primitives, as well as:

- [System.Half](#), which represents a half-precision floating-point number.
- [System.Decimal](#), which represents a decimal floating-point number.
- [System.Numerics.BigInteger](#), which is an integral type with no theoretical upper or lower bound.
- [System.Numerics.Complex](#), which represents complex numbers.
- A set of SIMD-enabled types in the [System.Numerics](#) namespace.

Integer types

.NET supports both signed and unsigned 8-bit, 16-bit, 32-bit, and 64-bit integer types, which are listed in the following tables.

Signed integer types

TYPE	SIZE (IN BYTES)	MINIMUM VALUE	MAXIMUM VALUE
System.Int16	2	-32,768	32,767
System.Int32	4	-2,147,483,648	2,147,483,647
System.Int64	8	-9,223,372,036,854,775,808	9,223,372,036,854,775,807
System.SByte	1	-128	127
System.IntPtr (in 32-bit process)	4	-2,147,483,648	2,147,483,647
System.IntPtr (in 64-bit process)	8	-9,223,372,036,854,775,808	9,223,372,036,854,775,807

Unsigned integer types

TYPE	SIZE (IN BYTES)	MINIMUM VALUE	MAXIMUM VALUE
System.Byte	1	0	255
System.UInt16	2	0	65,535
System.UInt32	4	0	4,294,967,295
System.UInt64	8	0	18,446,744,073,709,551,615
System.UIntPtr (in 32-bit process)	4	0	4,294,967,295

TYPE	SIZE (IN BYTES)	MINIMUM VALUE	MAXIMUM VALUE
System.UIntPtr (in 64-bit process)	8	0	18,446,744,073,709,551,615

Each integer type supports a set of standard arithmetic operators. The [System.Math](#) class provides methods for a broader set of mathematical functions.

You can also work with the individual bits in an integer value by using the [System.BitConverter](#) class.

NOTE

The unsigned integer types are not CLS-compliant. For more information, see [Language independence and language-independent components](#).

BigInteger

The [System.Numerics.BigInteger](#) structure is an immutable type that represents an arbitrarily large integer whose value in theory has no upper or lower bounds. The methods of the [BigInteger](#) type closely parallel those of the other integral types.

Floating-point types

.NET includes the following floating-point types:

TYPE	SIZE (IN BYTES)	APPROXIMATE RANGE	PRIMITIVE?	NOTES
System.Half	2	± 65504	No	Introduced in .NET 5
System.Single	4	$\pm 3.4 \times 10^{38}$	Yes	
System.Double	8	$\pm 1.7 \times 10^{308}$	Yes	
System.Decimal	16	$\pm 7.9228 \times 10^{28}$	No	

The [Half](#), [Single](#), and [Double](#) types support special values that represent not-a-number and infinity. For example, the [Double](#) type provides the following values: [Double.NaN](#), [Double.NegativeInfinity](#), and [Double.PositiveInfinity](#). You use the [Double.IsNaN](#), [Double.IsInfinity](#), [Double.IsPositiveInfinity](#), and [Double.IsNegativeInfinity](#) methods to test for these special values.

Each floating-point type supports a set of standard arithmetic operators. The [System.Math](#) class provides methods for a broader set of mathematical functions. .NET Core 2.0 and later includes the [System.MathF](#) class, which provides methods that accept arguments of the [Single](#) type.

You can also work with the individual bits in [Double](#), [Single](#), and [Half](#) values by using the [System.BitConverter](#) class. The [System.Decimal](#) structure has its own methods, [Decimal.GetBits](#) and [Decimal\(Int32\[\]\)](#), for working with a decimal value's individual bits, as well as its own set of methods for performing some additional mathematical operations.

The [Double](#), [Single](#), and [Half](#) types are intended to be used for values that, by their nature, are imprecise (for example, the distance between two stars) and for applications in which a high degree of precision and small rounding error is not required. Use the [System.Decimal](#) type for cases in which greater precision is required and rounding errors should be minimized.

NOTE

The [Decimal](#) type doesn't eliminate the need for rounding. Rather, it minimizes errors due to rounding.

Complex

The [System.Numerics.Complex](#) structure represents a complex number, that is, a number with a real number part and an imaginary number part. It supports a standard set of arithmetic, comparison, equality, explicit and implicit conversion operators, as well as mathematical, algebraic, and trigonometric methods.

SIMD-enabled types

The [System.Numerics](#) namespace includes a set of .NET SIMD-enabled types. SIMD (Single Instruction Multiple Data) operations can be parallelized at the hardware level. That increases the throughput of the vectorized computations, which are common in mathematical, scientific, and graphics apps.

The .NET SIMD-enabled types include the following:

- The [Vector2](#), [Vector3](#), and [Vector4](#) types, which represent vectors with 2, 3, and 4 [Single](#) values.
- Two matrix types, [Matrix3x2](#), which represents a 3x2 matrix, and [Matrix4x4](#), which represents a 4x4 matrix.
- The [Plane](#) type, which represents a plane in three-dimensional space.
- The [Quaternion](#) type, which represents a vector that is used to encode three-dimensional physical rotations.
- The [Vector<T>](#) type, which represents a vector of a specified numeric type and provides a broad set of operators that benefit from SIMD support. The count of a [Vector<T>](#) instance is fixed, but its value [Vector<T>.Count](#) depends on the CPU of the machine, on which code is executed.

NOTE

The [Vector<T>](#) type is included with .NET Core and .NET 5+, but not .NET Framework. If you're using .NET Framework, install the [System.Numerics.Vectors](#) NuGet package to get access to this type.

The SIMD-enabled types are implemented in such a way that they can be used with non-SIMD-enabled hardware or JIT compilers. To take advantage of SIMD instructions, your 64-bit apps must be run by the runtime that uses the RyuJIT compiler, which is included in .NET Core and in .NET Framework 4.6 and later versions. It adds SIMD support when targeting 64-bit processors.

For more information, see [Use SIMD-accelerated numeric types](#).

See also

- [Standard numeric format strings](#)
- [Floating-point numeric types in C#](#)

Dates, times, and time zones

9/20/2022 • 3 minutes to read • [Edit Online](#)

In addition to the basic [DateTime](#) structure, .NET provides the following classes that support working with time zones:

- [TimeZone](#)

Use this class to work with the system's local time zone and the Coordinated Universal Time (UTC) zone.

The functionality of the [TimeZone](#) class is largely superseded by the [TimeZoneInfo](#) class.

- [TimeZoneInfo](#)

Use this class to work with any time zone that is predefined on a system, to create new time zones, and to easily convert dates and times from one time zone to another. For new development, use the

[TimeZoneInfo](#) class instead of the [TimeZone](#) class.

- [DateTimeOffset](#)

Use this structure to work with dates and times whose offset (or difference) from UTC is known. The

[DateTimeOffset](#) structure combines a date and time value with that time's offset from UTC. Because of its relationship to UTC, an individual date and time value unambiguously identifies a single point in time.

This makes a [DateTimeOffset](#) value more portable from one computer to another than a [DateTime](#) value.

Starting with .NET 6, the following types are available:

- [DateOnly](#)

Use this structure when working with a value that only represents a date. The date represents the entire day, from the start of the day to the end. [DateOnly](#) has a range of `0001-01-01` through `9999-12-31`. And, this type represents the month, day, and year combination without a specific time. If you previously used a [DateTime](#) type in your code to represent a date that disregarded the time, use this type in its place.

- [TimeOnly](#)

Use this structure to represent a time without a date. The time represents the hours, minutes, and seconds of a non-specific day. [TimeOnly](#) has a range of `00:00:00.000000` to `23:59:59.999999`. This type can be used to replace [DateTime](#) and [TimeSpan](#) types in your code when you used those types to represent a time.

The next section provides the information that you need to work with time zones and to create time zone-aware applications that can convert dates and times from one time zone to another.

In this section

[Time zone overview](#)

Discusses the terminology, concepts, and issues involved in creating time zone-aware applications.

[Choosing between DateTime, DateTimeOffset, TimeSpan, and TimeZoneInfo](#)

Discusses when to use the [DateTime](#), [DateTimeOffset](#), and [TimeZoneInfo](#) types when working with date and time data.

[Finding the time zones defined on a local system](#)

Describes how to enumerate the time zones found on a local system.

[How to: Enumerate time zones present on a computer](#)

Provides examples that enumerate the time zones defined in a computer's registry and that let users select a predefined time zone from a list.

[How to: Access the predefined UTC and local time zone objects](#)

Describes how to access Coordinated Universal Time and the local time zone.

[How to: Instantiate a TimeZoneInfo object](#)

Describes how to instantiate a [TimeZoneInfo](#) object from the local system registry.

[Instantiating a DateTimeOffset object](#)

Discusses the ways in which a [DateTimeOffset](#) object can be instantiated, and the ways in which a [DateTime](#) value can be converted to a [DateTimeOffset](#) value.

[How to: Create time zones without adjustment rules](#)

Describes how to create a custom time zone that does not support the transition to and from daylight saving time.

[How to: Create time zones with adjustment rules](#)

Describes how to create a custom time zone that supports one or more transitions to and from daylight saving time.

[Saving and restoring time zones](#)

Describes [TimeZoneInfo](#) support for serialization and deserialization of time zone data and illustrates some of the scenarios in which these features can be used.

[How to: Save time zones to an embedded resource](#)

Describes how to create a custom time zone and save its information in a resource file.

[How to: Restore time zones from an embedded resource](#)

Describes how to instantiate custom time zones that have been saved to an embedded resource file.

[Performing arithmetic operations with dates and times](#)

Discusses the issues involved in adding, subtracting, and comparing [DateTime](#) and [DateTimeOffset](#) values.

[How to: Use time zones in date and time arithmetic](#)

Discusses how to perform date and time arithmetic that reflects a time zone's adjustment rules.

[Converting between DateTime and DateTimeOffset](#)

Describes how to convert between [DateTime](#) and [DateTimeOffset](#) values.

[Converting times between time zones](#)

Describes how to convert times from one time zone to another.

[How to: Resolve ambiguous times](#)

Describes how to resolve an ambiguous time by mapping it to the time zone's standard time.

[How to: Let users resolve ambiguous times](#)

Describes how to let a user determine the mapping between an ambiguous local time and Coordinated Universal Time.

Reference

[System.TimeZoneInfo](#)

Extend metadata using attributes

9/20/2022 • 2 minutes to read • [Edit Online](#)

The common language runtime allows you to add keyword-like descriptive declarations, called attributes, to annotate programming elements such as types, fields, methods, and properties. When you compile your code for the runtime, it is converted into Microsoft intermediate language (MSIL) and placed inside a portable executable (PE) file along with metadata generated by the compiler. Attributes allow you to place extra descriptive information into metadata that can be extracted using runtime reflection services. The compiler creates attributes when you declare instances of special classes that derive from [System.Attribute](#).

.NET uses attributes for a variety of reasons and to address a number of issues. Attributes describe how to serialize data, specify characteristics that are used to enforce security, and limit optimizations by the just-in-time (JIT) compiler so the code remains easy to debug. Attributes can also record the name of a file or the author of code, or control the visibility of controls and members during forms development.

Related articles

TITLE	DESCRIPTION
Applying Attributes	Describes how to apply an attribute to an element of your code.
Writing Custom Attributes	Describes how to design custom attribute classes.
Retrieving Information Stored in Attributes	Describes how to retrieve custom attributes for code that is loaded into the execution context.
Metadata and Self-Describing Components	Provides an overview of metadata and describes how it is implemented in a .NET portable executable (PE) file.
How to: Load Assemblies into the Reflection-Only Context	Explains how to retrieve custom attribute information in the reflection-only context.

Reference

- [System.Attribute](#)

Apply attributes

9/20/2022 • 3 minutes to read • [Edit Online](#)

Use the following process to apply an attribute to an element of your code.

1. Define a new attribute or use an existing .NET attribute.
2. Apply the attribute to the code element by placing it immediately before the element.

Each language has its own attribute syntax. In C++ and C#, the attribute is surrounded by square brackets and separated from the element by white space, which can include a line break. In Visual Basic, the attribute is surrounded by angle brackets and must be on the same logical line; the line continuation character can be used if a line break is desired.

3. Specify positional parameters and named parameters for the attribute.

Positional parameters are required and must come before any named parameters; they correspond to the parameters of one of the attribute's constructors. *Named* parameters are optional and correspond to read/write properties of the attribute. In C++, and C#, specify `name=value` for each optional parameter, where `name` is the name of the property. In Visual Basic, specify `name:=value`.

The attribute is emitted into metadata when you compile your code and is available to the common language runtime and any custom tool or application through the runtime reflection services.

By convention, all attribute names end with "Attribute". However, several languages that target the runtime, such as Visual Basic and C#, do not require you to specify the full name of an attribute. For example, if you want to initialize `System.ObsoleteAttribute`, you only need to reference it as `Obsolete`.

Apply an attribute to a method

The following code example shows how to use `System.ObsoleteAttribute`, which marks code as obsolete. The string `"Will be removed in next version"` is passed to the attribute. This attribute causes a compiler warning that displays the passed string when code that the attribute describes is called.

```

public ref class Example
{
    // Specify attributes between square brackets in C#.
    // This attribute is applied only to the Add method.
public:
    [Obsolete("Will be removed in next version.")]
    static int Add(int a, int b)
    {
        return (a + b);
    }
};

ref class Test
{
public:
    static void Main()
    {
        // This generates a compile-time warning.
        int i = Example::Add(2, 2);
    }
};

int main()
{
    Test::Main();
}

```

```

public class Example
{
    // Specify attributes between square brackets in C#.
    // This attribute is applied only to the Add method.
    [Obsolete("Will be removed in next version.")]
    public static int Add(int a, int b)
    {
        return (a + b);
    }
}

class Test
{
    public static void Main()
    {
        // This generates a compile-time warning.
        int i = Example.Add(2, 2);
    }
}

```

```

Public Class Example
    ' Specify attributes between square brackets in C#.
    ' This attribute is applied only to the Add method.
    <Obsolete("Will be removed in next version.")>
    Public Shared Function Add(a As Integer, b As Integer) As Integer
        Return a + b
    End Function
End Class

Class Test
    Public Shared Sub Main()
        ' This generates a compile-time warning.
        Dim i As Integer = Example.Add(2, 2)
    End Sub
End Class

```

Apply attributes at the assembly level

If you want to apply an attribute at the assembly level, use the `assembly` (`Assembly` in Visual Basic) keyword.

The following code shows the `AssemblyTitleAttribute` applied at the assembly level.

```
using namespace System::Reflection;  
[assembly: AssemblyTitle("My Assembly")];
```

```
using System.Reflection;  
[assembly: AssemblyTitle("My Assembly")]
```

```
Imports System.Reflection  
<Assembly: AssemblyTitle("My Assembly")>
```

When this attribute is applied, the string `"My Assembly"` is placed in the assembly manifest in the metadata portion of the file. You can view the attribute either by using the [MSIL Disassembler \(Ildasm.exe\)](#) or by creating a custom program to retrieve the attribute.

See also

- [Attributes](#)
- [Retrieving Information Stored in Attributes](#)
- [Concepts](#)
- [Attributes \(C#\)](#)
- [Attributes overview \(Visual Basic\)](#)

Writing Custom Attributes

9/20/2022 • 9 minutes to read • [Edit Online](#)

To design your own custom attributes, you do not need to learn many new concepts. If you are familiar with object-oriented programming and know how to design classes, you already have most of the knowledge needed. Custom attributes are essentially traditional classes that derive directly or indirectly from [System.Attribute](#). Just like traditional classes, custom attributes contain methods that store and retrieve data.

The primary steps to properly design custom attribute classes are as follows:

- [Applying the AttributeUsageAttribute](#)
- [Declaring the attribute class](#)
- [Declaring constructors](#)
- [Declaring properties](#)

This section describes each of these steps and concludes with a [custom attribute example](#).

Applying the AttributeUsageAttribute

A custom attribute declaration begins with the [System.AttributeUsageAttribute](#), which defines some of the key characteristics of your attribute class. For example, you can specify whether your attribute can be inherited by other classes or specify which elements the attribute can be applied to. The following code fragment demonstrates how to use the [AttributeUsageAttribute](#).

```
[AttributeUsage(AttributeTargets::All, Inherited = false, AllowMultiple = true)]
```

```
[AttributeUsage(AttributeTargets.All, Inherited = false, AllowMultiple = true)]
```

```
<AttributeUsage(AttributeTargets.All, Inherited:=False, AllowMultiple:=True)>
Public Class SomeClass
    Inherits Attribute
    ...
End Class
```

The [AttributeUsageAttribute](#) has three members that are important for the creation of custom attributes: [AttributeTargets](#), [Inherited](#), and [AllowMultiple](#).

AttributeTargets Member

In the previous example, [AttributeTargets.All](#) is specified, indicating that this attribute can be applied to all program elements. Alternatively, you can specify [AttributeTargets.Class](#), indicating that your attribute can be applied only to a class, or [AttributeTargets.Method](#), indicating that your attribute can be applied only to a method. All program elements can be marked for description by a custom attribute in this manner.

You can also pass multiple [AttributeTargets](#) values. The following code fragment specifies that a custom attribute can be applied to any class or method.

```
[AttributeUsage(AttributeTargets::Class | AttributeTargets::Method)]
```

```
[AttributeUsage(AttributeTargets.Class | AttributeTargets.Method)]
```

```
<AttributeUsage(AttributeTargets.Class Or AttributeTargets.Method)>
Public Class SomeOtherClass
    Inherits Attribute
    ...
End Class
```

Inherited Property

The [AttributeUsageAttribute.Inherited](#) property indicates whether your attribute can be inherited by classes that are derived from the classes to which your attribute is applied. This property takes either a `true` (the default) or `false` flag. In the following example, `MyAttribute` has a default [Inherited](#) value of `true`, while `YourAttribute` has an [Inherited](#) value of `false`.

```
// This defaults to Inherited = true.
public ref class MyAttribute : Attribute
{
    //...
};

[AttributeUsage(AttributeTargets::Method, Inherited = false)]
public ref class YourAttribute : Attribute
{
    //...
};
```

```
// This defaults to Inherited = true.
public class MyAttribute : Attribute
{
    //...
};

[AttributeUsage(AttributeTargets.Method, Inherited = false)]
public class YourAttribute : Attribute
{
    //...
};
```

```
' This defaults to Inherited = true.
Public Class MyAttribute
    Inherits Attribute
    ...
End Class

<AttributeUsage(AttributeTargets.Method, Inherited:=False)>
Public Class YourAttribute
    Inherits Attribute
    ...
End Class
```

The two attributes are then applied to a method in the base class `MyClass`.

```
public ref class MyClass
{
public:
    [MyAttribute]
    [YourAttribute]
    virtual void MyMethod()
    {
        //...
    }
};
```

```
public class MyClass
{
    [MyAttribute]
    [YourAttribute]
    public virtual void MyMethod()
    {
        //...
    }
}
```

```
Public Class MeClass
    <MyAttribute>
    <YourAttribute>
    Public Overridable Sub MyMethod()
        '...
    End Sub
End Class
```

Finally, the class `YourClass` is inherited from the base class `MyClass`. The method `MyMethod` shows `MyAttribute`, but not `YourAttribute`.

```
public ref class YourClass : MyClass
{
public:
    // MyMethod will have MyAttribute but not YourAttribute.
    virtual void MyMethod() override
    {
        //...
    }
};
```

```
public class YourClass : MyClass
{
    // MyMethod will have MyAttribute but not YourAttribute.
    public override void MyMethod()
    {
        //...
    }
}
```

```

Public Class YourClass
    Inherits MeClass
    ' MyMethod will have MyAttribute but not YourAttribute.
    Public Overrides Sub MyMethod()
        ...
    End Sub

End Class

```

AllowMultiple Property

The [AttributeUsageAttribute.AllowMultiple](#) property indicates whether multiple instances of your attribute can exist on an element. If set to `true`, multiple instances are allowed; if set to `false` (the default), only one instance is allowed.

In the following example, `MyAttribute` has a default [AllowMultiple](#) value of `false`, while `YourAttribute` has a value of `true`.

```

//This defaults to AllowMultiple = false.
public ref class MyAttribute : Attribute
{
};

[AttributeUsage(AttributeTargets::Method, AllowMultiple = true)]
public ref class YourAttribute : Attribute
{
};

```

```

//This defaults to AllowMultiple = false.
public class MyAttribute : Attribute
{
};

[AttributeUsage(AttributeTargets.Method, AllowMultiple = true)]
public class YourAttribute : Attribute
{
};

```

```

' This defaults to AllowMultiple = false.
Public Class MyAttribute
    Inherits Attribute
End Class

<AttributeUsage(AttributeTargets.Method, AllowMultiple:=true)>
Public Class YourAttribute
    Inherits Attribute
End Class

```

When multiple instances of these attributes are applied, `MyAttribute` produces a compiler error. The following code example shows the valid use of `YourAttribute` and the invalid use of `MyAttribute`.

```

public ref class MyClass
{
public:
    // This produces an error.
    // Duplicates are not allowed.
    [MyAttribute]
    [MyAttribute]
    void MyMethod()
    {
        //...
    }

    // This is valid.
    [YourAttribute]
    [YourAttribute]
    void YourMethod()
    {
        //...
    }
};

```

```

public class MyClass
{
    // This produces an error.
    // Duplicates are not allowed.
    [MyAttribute]
    [MyAttribute]
    public void MyMethod()
    {
        //...
    }

    // This is valid.
    [YourAttribute]
    [YourAttribute]
    public void YourMethod()
    {
        //...
    }
}

```

```

Public Class MyClass
    ' This produces an error.
    ' Duplicates are not allowed.
    <MyAttribute>
    <MyAttribute>
    Public Sub MyMethod()
        '...
    End Sub

    ' This is valid.
    <YourAttribute>
    <YourAttribute>
    Public Sub YourMethod()
        '...
    End Sub
End Class

```

If both the [AllowMultiple](#) property and the [Inherited](#) property are set to `true`, a class that is inherited from another class can inherit an attribute and have another instance of the same attribute applied in the same child class. If [AllowMultiple](#) is set to `false`, the values of any attributes in the parent class will be overwritten by new instances of the same attribute in the child class.

Declaring the Attribute Class

After you apply the [AttributeUsageAttribute](#), you can begin to define the specifics of your attribute. The declaration of an attribute class looks similar to the declaration of a traditional class, as demonstrated by the following code.

```
[AttributeUsage(AttributeTargets::Method)]
public ref class MyAttribute : Attribute
{
    // ...
};
```

```
[AttributeUsage(AttributeTargets.Method)]
public class MyAttribute : Attribute
{
    // ...
}
```

```
<AttributeUsage(AttributeTargets.Method)>
Public Class MyAttribute
    Inherits Attribute
    ' ...
End Class
```

This attribute definition demonstrates the following points:

- Attribute classes must be declared as public classes.
- By convention, the name of the attribute class ends with the word **Attribute**. While not required, this convention is recommended for readability. When the attribute is applied, the inclusion of the word **Attribute** is optional.
- All attribute classes must inherit directly or indirectly from [System.Attribute](#).
- In Microsoft Visual Basic, all custom attribute classes must have the [System.AttributeUsageAttribute](#) attribute.

Declaring Constructors

Attributes are initialized with constructors in the same way as traditional classes. The following code fragment illustrates a typical attribute constructor. This public constructor takes a parameter and sets a member variable equal to its value.

```
MyAttribute(bool myvalue)
{
    this->myvalue = myvalue;
}
```

```
public MyAttribute(bool myvalue)
{
    this.myvalue = myvalue;
}
```

```

Public Sub New(myvalue As Boolean)
    Me.myvalue = myvalue
End Sub

```

You can overload the constructor to accommodate different combinations of values. If you also define a [property](#) for your custom attribute class, you can use a combination of named and positional parameters when initializing the attribute. Typically, you define all required parameters as positional and all optional parameters as named. In this case, the attribute cannot be initialized without the required parameter. All other parameters are optional. Note that in Visual Basic, constructors for an attribute class should not use a ParamArray argument.

The following code example shows how an attribute that uses the previous constructor can be applied using optional and required parameters. It assumes that the attribute has one required Boolean value and one optional string property.

```

// One required (positional) and one optional (named) parameter are applied.
[MyAttribute(false, OptionalParameter = "optional data")]
public ref class SomeClass
{
    //...
};

// One required (positional) parameter is applied.
[MyAttribute(false)]
public ref class SomeOtherClass
{
    //...
};

```

```

// One required (positional) and one optional (named) parameter are applied.
[MyAttribute(false, OptionalParameter = "optional data")]
public class SomeClass
{
    //...
}

// One required (positional) parameter is applied.
[MyAttribute(false)]
public class SomeOtherClass
{
    //...
}

```

```

' One required (positional) and one optional (named) parameter are applied.
<MyAttribute(false, OptionalParameter:="optional data")>
Public Class SomeClass
    ...
End Class

' One required (positional) parameter is applied.
<MyAttribute(false)>
Public Class SomeOtherClass
    ...
End Class

```

Declaring Properties

If you want to define a named parameter or provide an easy way to return the values stored by your attribute, declare a [property](#). Attribute properties should be declared as public entities with a description of the data type that will be returned. Define the variable that will hold the value of your property and associate it with the **get**

and **set** methods. The following code example demonstrates how to implement a simple property in your attribute.

```
property bool MyProperty
{
    bool get() {return this->myvalue;}
    void set(bool value) {this->myvalue = value;}
}
```

```
public bool MyProperty
{
    get {return this.myvalue;}
    set {this.myvalue = value;}
}
```

```
Public Property MyProperty As Boolean
    Get
        Return Me.myvalue
    End Get
    Set
        Me.myvalue = Value
    End Set
End Property
```

Custom Attribute Example

This section incorporates the previous information and shows how to design a simple attribute that documents information about the author of a section of code. The attribute in this example stores the name and level of the programmer, and whether the code has been reviewed. It uses three private variables to store the actual values to save. Each variable is represented by a public property that gets and sets the values. Finally, the constructor is defined with two required parameters.

```
[AttributeUsage(AttributeTargets::All)]
public ref class DeveloperAttribute : Attribute
{
    // Private fields.
private:
    String^ name;
    String^ level;
    bool reviewed;

public:
    // This constructor defines two required parameters: name and level.

    DeveloperAttribute(String^ name, String^ level)
    {
        this->name = name;
        this->level = level;
        this->reviewed = false;
    }

    // Define Name property.
    // This is a read-only attribute.

    virtual property String^ Name
    {
        String^ get() {return name;}
    }

    // Define Level property.
    // This is a read-only attribute.

    virtual property String^ Level
    {
        String^ get() {return level;}
    }

    // Define Reviewed property.
    // This is a read/write attribute.

    virtual property bool Reviewed
    {
        bool get() {return reviewed;}
        void set(bool value) {reviewed = value;}
    }
};
```

```
[AttributeUsage(AttributeTargets.All)]
public class DeveloperAttribute : Attribute
{
    // Private fields.
    private string name;
    private string level;
    private bool reviewed;

    // This constructor defines two required parameters: name and level.

    public DeveloperAttribute(string name, string level)
    {
        this.name = name;
        this.level = level;
        this.reviewed = false;
    }

    // Define Name property.
    // This is a read-only attribute.

    public virtual string Name
    {
        get {return name;}
    }

    // Define Level property.
    // This is a read-only attribute.

    public virtual string Level
    {
        get {return level;}
    }

    // Define Reviewed property.
    // This is a read/write attribute.

    public virtual bool Reviewed
    {
        get {return reviewed;}
        set {reviewed = value;}
    }
}
```

```

<AttributeUsage(AttributeTargets.All)>
Public Class DeveloperAttribute
    Inherits Attribute
    ' Private fields.
    Private myname As String
    Private mylevel As String
    Private myreviewed As Boolean

    ' This constructor defines two required parameters: name and level.

    Public Sub New(name As String, level As String)
        Me.myname = name
        Me.mylevel = level
        Me.myreviewed = False
    End Sub

    ' Define Name property.
    ' This is a read-only attribute.

    Public Overridable ReadOnly Property Name() As String
        Get
            Return myname
        End Get
    End Property

    ' Define Level property.
    ' This is a read-only attribute.

    Public Overridable ReadOnly Property Level() As String
        Get
            Return mylevel
        End Get
    End Property

    ' Define Reviewed property.
    ' This is a read/write attribute.

    Public Overridable Property Reviewed() As Boolean
        Get
            Return myreviewed
        End Get
        Set
            myreviewed = value
        End Set
    End Property
End Class

```

You can apply this attribute using the full name, `DeveloperAttribute`, or using the abbreviated name, `Developer`, in one of the following ways.

```

[Developer("Joan Smith", "1")]

-or-

[Developer("Joan Smith", "1", Reviewed = true)]

```

```

[Developer("Joan Smith", "1")]

-or-

[Developer("Joan Smith", "1", Reviewed = true)]

```

```
<Developer("Joan Smith", "1")>  
-or-  
<Developer("Joan Smith", "1", Reviewed := true)>
```

The first example shows the attribute applied with only the required named parameters, while the second example shows the attribute applied with both the required and optional parameters.

See also

- [System.Attribute](#)
- [System.AttributeUsageAttribute](#)
- [Attributes](#)
- [Attribute parameter types](#)

Retrieving Information Stored in Attributes

9/20/2022 • 9 minutes to read • [Edit Online](#)

Retrieving a custom attribute is a simple process. First, declare an instance of the attribute you want to retrieve. Then, use the `Attribute.GetCustomAttribute` method to initialize the new attribute to the value of the attribute you want to retrieve. Once the new attribute is initialized, you simply use its properties to get the values.

IMPORTANT

This topic describes how to retrieve attributes for code loaded into the execution context. To retrieve attributes for code loaded into the reflection-only context, you must use the `CustomAttributeData` class, as shown in [How to: Load Assemblies into the Reflection-Only Context](#).

This section describes the following ways to retrieve attributes:

- [Retrieving a single instance of an attribute](#)
- [Retrieving multiple instances of an attribute applied to the same scope](#)
- [Retrieving multiple instances of an attribute applied to different scopes](#)

Retrieving a Single Instance of an Attribute

In the following example, the `DeveloperAttribute` (described in the previous section) is applied to the `MainApp` class on the class level. The `GetAttribute` method uses `GetCustomAttribute` to retrieve the values stored in `DeveloperAttribute` on the class level before displaying them to the console.

```
using namespace System;
using namespace System::Reflection;
using namespace CustomCodeAttributes;

[Developer("Joan Smith", "42", Reviewed = true)]
ref class MainApp
{
public:
    static void Main()
    {
        // Call function to get and display the attribute.
        GetAttribute(MainApp::typeid);
    }

    static void GetAttribute(Type^ t)
    {
        // Get instance of the attribute.
        DeveloperAttribute^ MyAttribute =
            (DeveloperAttribute^) Attribute::GetCustomAttribute(t, DeveloperAttribute::typeid);

        if (MyAttribute == nullptr)
        {
            Console::WriteLine("The attribute was not found.");
        }
        else
        {
            // Get the Name value.
            Console::WriteLine("The Name Attribute is: {0}." , MyAttribute->Name);
            // Get the Level value.
            Console::WriteLine("The Level Attribute is: {0}." , MyAttribute->Level);
            // Get the Reviewed value.
            Console::WriteLine("The Reviewed Attribute is: {0}." , MyAttribute->Reviewed);
        }
    }
};
```

```

using System;
using System.Reflection;
using CustomCodeAttributes;

[Developer("Joan Smith", "42", Reviewed = true)]
class MainApp
{
    public static void Main()
    {
        // Call function to get and display the attribute.
        GetAttribute(typeof(MainApp));
    }

    public static void GetAttribute(Type t)
    {
        // Get instance of the attribute.
        DeveloperAttribute MyAttribute =
            (DeveloperAttribute) Attribute.GetCustomAttribute(t, typeof(DeveloperAttribute));

        if (MyAttribute == null)
        {
            Console.WriteLine("The attribute was not found.");
        }
        else
        {
            // Get the Name value.
            Console.WriteLine("The Name Attribute is: {0}." , MyAttribute.Name);
            // Get the Level value.
            Console.WriteLine("The Level Attribute is: {0}." , MyAttribute.Level);
            // Get the Reviewed value.
            Console.WriteLine("The Reviewed Attribute is: {0}." , MyAttribute.Reviewed);
        }
    }
}

```

```

Imports System.Reflection
Imports CustomCodeAttributes

<Developer("Joan Smith", "42", Reviewed:=True)>
Class MainApp
    Public Shared Sub Main()
        ' Call function to get and display the attribute.
        GetAttribute(GetType(MainApp))
    End Sub

    Public Shared Sub GetAttribute(t As Type)
        ' Get instance of the attribute.
        Dim MyAttribute As DeveloperAttribute =
            CType(Attribute.GetCustomAttribute(t, GetType(DeveloperAttribute)), DeveloperAttribute)

        If MyAttribute Is Nothing Then
            Console.WriteLine("The attribute was not found.")
        Else
            ' Get the Name value.
            Console.WriteLine("The Name Attribute is: {0}." , MyAttribute.Name)
            ' Get the Level value.
            Console.WriteLine("The Level Attribute is: {0}." , MyAttribute.Level)
            ' Get the Reviewed value.
            Console.WriteLine("The Reviewed Attribute is: {0}." , MyAttribute.Reviewed)
        End If
    End Sub
End Class

```

This program displays the following text when executed.

```
The Name Attribute is: Joan Smith.  
The Level Attribute is: 42.  
The Reviewed Attribute is: True.
```

If the attribute is not found, the **GetCustomAttribute** method initializes `MyAttribute` to a null value. This example checks `MyAttribute` for such an instance and notifies the user if no attribute is found. If the `DeveloperAttribute` is not found in the class scope, the following message displays to the console.

```
The attribute was not found.
```

This example assumes that the attribute definition is in the current namespace. Remember to import the namespace in which the attribute definition resides if it is not in the current namespace.

Retrieving Multiple Instances of an Attribute Applied to the Same Scope

In the previous example, the class to inspect and the specific attribute to find are passed to **GetCustomAttribute**. That code works well if only one instance of an attribute is applied on the class level. However, if multiple instances of an attribute are applied on the same class level, the **GetCustomAttribute** method does not retrieve all the information. In cases where multiple instances of the same attribute are applied to the same scope, you can use **Attribute.GetCustomAttributes** to place all instances of an attribute into an array. For example, if two instances of `DeveloperAttribute` are applied on the class level of the same class, the `GetAttribute` method can be modified to display the information found in both attributes. Remember, to apply multiple attributes on the same level, the attribute must be defined with the **AllowMultiple** property set to **true** in the **AttributeUsageAttribute**.

The following code example shows how to use the **GetCustomAttributes** method to create an array that references all instances of `DeveloperAttribute` in any given class. The values of all attributes are then displayed to the console.

```
public:  
    static void GetAttribute(Type^ t)  
    {  
        array<DeveloperAttribute^>^ MyAttributes =  
            (array<DeveloperAttribute^>^) Attribute::GetCustomAttributes(t, DeveloperAttribute::typeid);  
  
        if (MyAttributes->Length == 0)  
        {  
            Console::WriteLine("The attribute was not found.");  
        }  
        else  
        {  
            for (int i = 0 ; i < MyAttributes->Length; i++)  
            {  
                // Get the Name value.  
                Console::WriteLine("The Name Attribute is: {0}." , MyAttributes[i]->Name);  
                // Get the Level value.  
                Console::WriteLine("The Level Attribute is: {0}." , MyAttributes[i]->Level);  
                // Get the Reviewed value.  
                Console::WriteLine("The Reviewed Attribute is: {0}." , MyAttributes[i]->Reviewed);  
            }  
        }  
    }
```

```

public static void GetAttribute(Type t)
{
    DeveloperAttribute[] MyAttributes =
        (DeveloperAttribute[]) Attribute.GetCustomAttributes(t, typeof (DeveloperAttribute));

    if (MyAttributes.Length == 0)
    {
        Console.WriteLine("The attribute was not found.");
    }
    else
    {
        for (int i = 0 ; i < MyAttributes.Length ; i++)
        {
            // Get the Name value.
            Console.WriteLine("The Name Attribute is: {0}." , MyAttributes[i].Name);
            // Get the Level value.
            Console.WriteLine("The Level Attribute is: {0}." , MyAttributes[i].Level);
            // Get the Reviewed value.
            Console.WriteLine("The Reviewed Attribute is: {0}." , MyAttributes[i].Reviewed);
        }
    }
}

```

```

Public Shared Sub GetAttribute(t As Type)
    Dim MyAttributes() As DeveloperAttribute =
        CType(Attribute.GetCustomAttributes(t, GetType(DeveloperAttribute)), DeveloperAttribute())

    If MyAttributes.Length = 0 Then
        Console.WriteLine("The attribute was not found.")
    Else
        For i As Integer = 0 To MyAttributes.Length - 1
            ' Get the Name value.
            Console.WriteLine("The Name Attribute is: {0}." , MyAttributes(i).Name)
            ' Get the Level value.
            Console.WriteLine("The Level Attribute is: {0}." , MyAttributes(i).Level)
            ' Get the Reviewed value.
            Console.WriteLine("The Reviewed Attribute is: {0}." , MyAttributes(i).Reviewed)
        Next i
    End If
End Sub

```

If no attributes are found, this code alerts the user. Otherwise, the information contained in both instances of `DeveloperAttribute` is displayed.

Retrieving Multiple Instances of an Attribute Applied to Different Scopes

The `GetCustomAttributes` and `GetCustomAttribute` methods do not search an entire class and return all instances of an attribute in that class. Rather, they search only one specified method or member at a time. If you have a class with the same attribute applied to every member and you want to retrieve the values in all the attributes applied to those members, you must supply every method or member individually to `GetCustomAttributes` and `GetCustomAttribute`.

The following code example takes a class as a parameter and searches for the `DeveloperAttribute` (defined previously) on the class level and on every individual method of that class.

```

public:
    static void GetAttribute(Type^ t)
    {
        DeveloperAttribute^ att;

        // Get the class-level attributes.

        // Put the instance of the attribute on the class level in the att object.
        att = (DeveloperAttribute^) Attribute::GetCustomAttribute (t, DeveloperAttribute::typeid);

        if (att == nullptr)
        {
            Console::WriteLine("No attribute in class {0}.\n", t->ToString());
        }
        else
        {
            Console::WriteLine("The Name Attribute on the class level is: {0}.", att->Name);
            Console::WriteLine("The Level Attribute on the class level is: {0}.", att->Level);
            Console::WriteLine("The Reviewed Attribute on the class level is: {0}.\n", att->Reviewed);
        }

        // Get the method-level attributes.

        // Get all methods in this class, and put them
        // in an array of System.Reflection.MemberInfo objects.
        array<MemberInfo^>^ MyMethodInfo = t->GetMethods();

        // Loop through all methods in this class that are in the
        // MyMethodInfo array.
        for (int i = 0; i < MyMethodInfo->Length; i++)
        {
            att = (DeveloperAttribute^) Attribute::GetCustomAttribute(MyMethodInfo[i],
DeveloperAttribute::typeid);
            if (att == nullptr)
            {
                Console::WriteLine("No attribute in member function {0}.\n", MyMethodInfo[i]->ToString());
            }
            else
            {
                Console::WriteLine("The Name Attribute for the {0} member is: {1}.", 
MyMethodInfo[i]->ToString(), att->Name);
                Console::WriteLine("The Level Attribute for the {0} member is: {1}.", 
MyMethodInfo[i]->ToString(), att->Level);
                Console::WriteLine("The Reviewed Attribute for the {0} member is: {1}.\n",
MyMethodInfo[i]->ToString(), att->Reviewed);
            }
        }
    }
}

```

```
public static void GetAttribute(Type t)
{
    DeveloperAttribute att;

    // Get the class-level attributes.

    // Put the instance of the attribute on the class level in the att object.
    att = (DeveloperAttribute) Attribute.GetCustomAttribute (t, typeof (DeveloperAttribute));

    if (att == null)
    {
        Console.WriteLine("No attribute in class {0}.\n", t.ToString());
    }
    else
    {
        Console.WriteLine("The Name Attribute on the class level is: {0}.", att.Name);
        Console.WriteLine("The Level Attribute on the class level is: {0}.", att.Level);
        Console.WriteLine("The Reviewed Attribute on the class level is: {0}.\n", att.Reviewed);
    }

    // Get the method-level attributes.

    // Get all methods in this class, and put them
    // in an array of System.Reflection.MemberInfo objects.
    MemberInfo[] MyMethodInfo = t.GetMethods();

    // Loop through all methods in this class that are in the
    // MyMethodInfo array.
    for (int i = 0; i < MyMethodInfo.Length; i++)
    {
        att = (DeveloperAttribute) Attribute.GetCustomAttribute(MyMethodInfo[i], typeof
(DeveloperAttribute));
        if (att == null)
        {
            Console.WriteLine("No attribute in member function {0}.\n" , MyMethodInfo[i].ToString());
        }
        else
        {
            Console.WriteLine("The Name Attribute for the {0} member is: {1}." ,
MyMethodInfo[i].ToString(), att.Name);
            Console.WriteLine("The Level Attribute for the {0} member is: {1}." ,
MyMethodInfo[i].ToString(), att.Level);
            Console.WriteLine("The Reviewed Attribute for the {0} member is: {1}.\n" ,
MyMethodInfo[i].ToString(), att.Reviewed);
        }
    }
}
```

```

Public Shared Sub GetAttribute(t As Type)
    Dim att As DeveloperAttribute

    ' Get the class-level attributes.

    ' Put the instance of the attribute on the class level in the att object.
    att = CType(Attribute.GetCustomAttribute(t, GetType(DeveloperAttribute)), DeveloperAttribute)

    If att Is Nothing
        Console.WriteLine("No attribute in class {0}.\n", t.ToString())
    Else
        Console.WriteLine("The Name Attribute on the class level is: {0}.", att.Name)
        Console.WriteLine("The Level Attribute on the class level is: {0}.", att.Level)
        Console.WriteLine("The Reviewed Attribute on the class level is: {0}.\n", att.Reviewed)
    End If

    ' Get the method-level attributes.

    ' Get all methods in this class, and put them
    ' in an array of System.Reflection.MemberInfo objects.
    Dim MyMethodInfo() As MemberInfo = t.GetMethods()

    ' Loop through all methods in this class that are in the
    ' MyMethodInfo array.
    For i As Integer = 0 To MyMethodInfo.Length - 1
        att = CType(Attribute.GetCustomAttribute(MyMethodInfo(i), _
            GetType(DeveloperAttribute)), DeveloperAttribute)
        If att Is Nothing Then
            Console.WriteLine("No attribute in member function {0}.\n", MyMethodInfo(i).ToString())
        Else
            Console.WriteLine("The Name Attribute for the {0} member is: {1}.",
                MyMethodInfo(i).ToString(), att.Name)
            Console.WriteLine("The Level Attribute for the {0} member is: {1}.",
                MyMethodInfo(i).ToString(), att.Level)
            Console.WriteLine("The Reviewed Attribute for the {0} member is: {1}.\n",
                MyMethodInfo(i).ToString(), att.Reviewed)
        End If
    Next
End Sub

```

If no instances of the `DeveloperAttribute` are found on the method level or class level, the `GetAttribute` method notifies the user that no attributes were found and displays the name of the method or class that does not contain the attribute. If an attribute is found, the `Name`, `Level`, and `Reviewed` fields are displayed to the console.

You can use the members of the `Type` class to get the individual methods and members in the passed class. This example first queries the `Type` object to get attribute information for the class level. Next, it uses `Type.GetMethods` to place instances of all methods into an array of `System.Reflection.MemberInfo` objects to retrieve attribute information for the method level. You can also use the `Type.GetProperties` method to check for attributes on the property level or `Type.GetConstructors` to check for attributes on the constructor level.

See also

- [System.Type](#)
- [Attribute.GetCustomAttribute](#)
- [Attribute.GetCustomAttributes](#)
- [Attributes](#)

Runtime libraries overview

9/20/2022 • 2 minutes to read • [Edit Online](#)

The [.NET runtime](#), which is installed on a machine for use by [framework-dependent apps](#), has an expansive standard set of class libraries, known as [runtime libraries](#), [framework libraries](#), or the [base class library \(BCL\)](#). In addition, there are extensions to the runtime libraries, provided in NuGet packages.

These libraries provide implementations for many general and app-specific types, algorithms, and utility functionality.

Runtime libraries

These libraries provide the foundational types and utility functionality and are the base of all other .NET class libraries. An example is the [System.String](#) class, which provides APIs for working with strings. Another example is the [serialization libraries](#).

Extensions to the runtime libraries

Some libraries are provided in NuGet packages rather than included in the runtime's [shared framework](#). For example:

CONCEPTUAL CONTENT	NUGET PACKAGE
Configuration	Microsoft.Extensions.Configuration
Dependency injection	Microsoft.Extensions.DependencyInjection
File globbing	Microsoft.Extensions.FileSystemGlobbing
Generic Host	Microsoft.Extensions.Hosting
HTTP	[†] Microsoft.Extensions.Http
Localization	Microsoft.Extensions.Localization
Logging	Microsoft.Extensions.Logging

[†] For some target frameworks, including `net6.0`, these libraries are part of the shared framework and don't need to be installed separately.

See also

- [Introduction to .NET](#)
- [Install .NET SDK or runtime](#)
- [Select the installed .NET SDK or runtime version to use](#)
- [Publish framework-dependent apps](#)

Overview: How to format numbers, dates, enums, and other types in .NET

9/20/2022 • 37 minutes to read • [Edit Online](#)

Formatting is the process of converting an instance of a class or structure, or an enumeration value, to a string representation. The purpose is to display the resulting string to users or to deserialize it later to restore the original data type. This article introduces the formatting mechanisms that .NET provides.

NOTE

Parsing is the inverse of formatting. A parsing operation creates an instance of a data type from its string representation. For more information, see [Parsing Strings](#). For information about serialization and deserialization, see [Serialization in .NET](#).

The basic mechanism for formatting is the default implementation of the [Object.ToString](#) method, which is discussed in the [Default Formatting Using the ToString Method](#) section later in this topic. However, .NET provides several ways to modify and extend its default formatting support. These include the following:

- Overriding the [Object.ToString](#) method to define a custom string representation of an object's value. For more information, see the [Override the ToString Method](#) section later in this topic.
- Defining format specifiers that enable the string representation of an object's value to take multiple forms. For example, the "X" format specifier in the following statement converts an integer to the string representation of a hexadecimal value.

```
int integerValue = 60312;
Console.WriteLine(integerValue.ToString("X"));    // Displays EB98.
```

```
Dim integerValue As Integer = 60312
Console.WriteLine(integerValue.ToString("X"))      ' Displays EB98.
```

For more information about format specifiers, see the [ToString Method and Format Strings](#) section.

- Using format providers to implement the formatting conventions of a specific culture. For example, the following statement displays a currency value by using the formatting conventions of the en-US culture.

```
double cost = 1632.54;
Console.WriteLine(cost.ToString("C",
                           new System.Globalization.CultureInfo("en-US")));
// The example displays the following output:
//      $1,632.54
```

```
Dim cost As Double = 1632.54
Console.WriteLine(cost.ToString("C", New System.Globalization.CultureInfo("en-US")))
' The example displays the following output:
'      $1,632.54
```

For more information about formatting with format providers, see the [Format Providers](#) section.

- Implementing the [IFormattable](#) interface to support both string conversion with the [Convert](#) class and

composite formatting. For more information, see the [IFormattable Interface](#) section.

- Using composite formatting to embed the string representation of a value in a larger string. For more information, see the [Composite Formatting](#) section.
- Using string interpolation, a more readable syntax to embed the string representation of a value in a larger string. For more information, see [String interpolation](#).
- Implementing [ICustomFormatter](#) and [IFormatProvider](#) to provide a complete custom formatting solution. For more information, see the [Custom Formatting with ICustomFormatter](#) section.

The following sections examine these methods for converting an object to its string representation.

Default formatting using the `ToString` method

Every type that is derived from [System.Object](#) automatically inherits a parameterless `ToString` method, which returns the name of the type by default. The following example illustrates the default `ToString` method. It defines a class named `Automobile` that has no implementation. When the class is instantiated and its `ToString` method is called, it displays its type name. Note that the `ToString` method is not explicitly called in the example. The [Console.WriteLine\(Object\)](#) method implicitly calls the `ToString` method of the object passed to it as an argument.

```
using System;

public class Automobile
{
    // No implementation. All members are inherited from Object.
}

public class Example9
{
    public static void Main()
    {
        Automobile firstAuto = new Automobile();
        Console.WriteLine(firstAuto);
    }
}
// The example displays the following output:
//      Automobile
```

```
Public Class Automobile
    ' No implementation. All members are inherited from Object.
End Class

Module Example9
    Public Sub Main9()
        Dim firstAuto As New Automobile()
        Console.WriteLine(firstAuto)
    End Sub
End Module
' The example displays the following output:
'      Automobile
```

WARNING

Starting with Windows 8.1, the Windows Runtime includes an `IStringable` interface with a single method, `IStringable.ToString`, which provides default formatting support. However, we recommend that managed types do not implement the `IStringable` interface. For more information, see "The Windows Runtime and the `IStringable` Interface" section on the `Object.ToString` reference page.

Because all types other than interfaces are derived from `Object`, this functionality is automatically provided to your custom classes or structures. However, the functionality offered by the default `ToString` method, is limited: Although it identifies the type, it fails to provide any information about an instance of the type. To provide a string representation of an object that provides information about that object, you must override the `ToString` method.

NOTE

Structures inherit from `ValueType`, which in turn is derived from `Object`. Although `ValueType` overrides `Object.ToString`, its implementation is identical.

Override the `ToString` method

Displaying the name of a type is often of limited use and does not allow consumers of your types to differentiate one instance from another. However, you can override the `ToString` method to provide a more useful representation of an object's value. The following example defines a `Temperature` object and overrides its `ToString` method to display the temperature in degrees Celsius.

```
public class Temperature
{
    private decimal temp;

    public Temperature(decimal temperature)
    {
        this.temp = temperature;
    }

    public override string ToString()
    {
        return this.temp.ToString("N1") + "°C";
    }
}

public class Example12
{
    public static void Main()
    {
        Temperature currentTemperature = new Temperature(23.6m);
        Console.WriteLine($"The current temperature is {currentTemperature}");
    }
}
// The example displays the following output:
//      The current temperature is 23.6°C.
```

```

Public Class Temperature
    Private temp As Decimal

    Public Sub New(temperature As Decimal)
        Me.temp = temperature
    End Sub

    Public Overrides Function ToString() As String
        Return Me.temp.ToString("N1") + "°C"
    End Function
End Class

Module Example13
    Public Sub Main13()
        Dim currentTemperature As New Temperature(23.6D)
        Console.WriteLine("The current temperature is " +
            currentTemperature.ToString())
    End Sub
End Module

' The example displays the following output:
'      The current temperature is 23.6°C.

```

In .NET, the `ToString` method of each primitive value type has been overridden to display the object's value instead of its name. The following table shows the override for each primitive type. Note that most of the overridden methods call another overload of the `ToString` method and pass it the "G" format specifier, which defines the general format for its type, and an `IFormatProvider` object that represents the current culture.

TYPE	TO STRING OVERRIDE
Boolean	Returns either <code>Boolean.TrueString</code> or <code>Boolean.FalseString</code> .
Byte	Calls <code>Byte.ToString("G", NumberFormatInfo.CurrentInfo)</code> to format the <code>Byte</code> value for the current culture.
Char	Returns the character as a string.
DateTime	Calls <code>DateTime.ToString("G", DatetimeFormatInfo.CurrentInfo)</code> to format the date and time value for the current culture.
Decimal	Calls <code>Decimal.ToString("G", NumberFormatInfo.CurrentInfo)</code> to format the <code>Decimal</code> value for the current culture.
Double	Calls <code>Double.ToString("G", NumberFormatInfo.CurrentInfo)</code> to format the <code>Double</code> value for the current culture.
Int16	Calls <code>Int16.ToString("G", NumberFormatInfo.CurrentInfo)</code> to format the <code>Int16</code> value for the current culture.
Int32	Calls <code>Int32.ToString("G", NumberFormatInfo.CurrentInfo)</code> to format the <code>Int32</code> value for the current culture.

TYPE	TO STRING OVERRIDE
Int64	Calls <code>Int64.ToString("G", NumberFormatInfo.CurrentInfo)</code> to format the <code>Int64</code> value for the current culture.
SByte	Calls <code>SByte.ToString("G", NumberFormatInfo.CurrentInfo)</code> to format the <code>SByte</code> value for the current culture.
Single	Calls <code>Single.ToString("G", NumberFormatInfo.CurrentInfo)</code> to format the <code>Single</code> value for the current culture.
UInt16	Calls <code>UInt16.ToString("G", NumberFormatInfo.CurrentInfo)</code> to format the <code>UInt16</code> value for the current culture.
UInt32	Calls <code>UInt32.ToString("G", NumberFormatInfo.CurrentInfo)</code> to format the <code>UInt32</code> value for the current culture.
UInt64	Calls <code>UInt64.ToString("G", NumberFormatInfo.CurrentInfo)</code> to format the <code>UInt64</code> value for the current culture.

The `ToString` method and format strings

Relying on the default `ToString` method or overriding `ToString` is appropriate when an object has a single string representation. However, the value of an object often has multiple representations. For example, a temperature can be expressed in degrees Fahrenheit, degrees Celsius, or kelvins. Similarly, the integer value 10 can be represented in numerous ways, including 10, 10.0, 1.0e01, or \$10.00.

To enable a single value to have multiple string representations, .NET uses format strings. A format string is a string that contains one or more predefined format specifiers, which are single characters or groups of characters that define how the `ToString` method should format its output. The format string is then passed as a parameter to the object's `ToString` method and determines how the string representation of that object's value should appear.

All numeric types, date and time types, and enumeration types in .NET support a predefined set of format specifiers. You can also use format strings to define multiple string representations of your application-defined data types.

Standard format strings

A standard format string contains a single format specifier, which is an alphabetic character that defines the string representation of the object to which it is applied, along with an optional precision specifier that affects how many digits are displayed in the result string. If the precision specifier is omitted or is not supported, a standard format specifier is equivalent to a standard format string.

.NET defines a set of standard format specifiers for all numeric types, all date and time types, and all enumeration types. For example, each of these categories supports a "G" standard format specifier, which defines a general string representation of a value of that type.

Standard format strings for enumeration types directly control the string representation of a value. The format strings passed to an enumeration value's `ToString` method determine whether the value is displayed using its

string name (the "G" and "F" format specifiers), its underlying integral value (the "D" format specifier), or its hexadecimal value (the "X" format specifier). The following example illustrates the use of standard format strings to format a [DayOfWeek](#) enumeration value.

```
DayOfWeek thisDay = DayOfWeek.Monday;
string[] formatStrings = {"G", "F", "D", "X"};

foreach (string formatString in formatStrings)
    Console.WriteLine(thisDay.ToString(formatString));
// The example displays the following output:
//      Monday
//      Monday
//      1
//      00000001
```

```
Dim thisDay As DayOfWeek = DayOfWeek.Monday
Dim formatStrings() As String = {"G", "F", "D", "X"}

For Each formatString As String In formatStrings
    Console.WriteLine(thisDay.ToString(formatString))
Next
' The example displays the following output:
'      Monday
'      Monday
'      1
'      00000001
```

For information about enumeration format strings, see [Enumeration Format Strings](#).

Standard format strings for numeric types usually define a result string whose precise appearance is controlled by one or more property values. For example, the "C" format specifier formats a number as a currency value.

When you call the `ToString` method with the "C" format specifier as the only parameter, the following property values from the current culture's `NumberFormatInfo` object are used to define the string representation of the numeric value:

- The `CurrencySymbol` property, which specifies the current culture's currency symbol.
- The `CurrencyNegativePattern` or `CurrencyPositivePattern` property, which returns an integer that determines the following:
 - The placement of the currency symbol.
 - Whether negative values are indicated by a leading negative sign, a trailing negative sign, or parentheses.
 - Whether a space appears between the numeric value and the currency symbol.
- The `CurrencyDecimalDigits` property, which defines the number of fractional digits in the result string.
- The `CurrencyDecimalSeparator` property, which defines the decimal separator symbol in the result string.
- The `CurrencyGroupSeparator` property, which defines the group separator symbol.
- The `CurrencyGroupSizes` property, which defines the number of digits in each group to the left of the decimal.
- The `NegativeSign` property, which determines the negative sign used in the result string if parentheses are not used to indicate negative values.

In addition, numeric format strings may include a precision specifier. The meaning of this specifier depends on

the format string with which it is used, but it typically indicates either the total number of digits or the number of fractional digits that should appear in the result string. For example, the following example uses the "X4" standard numeric string and a precision specifier to create a string value that has four hexadecimal digits.

```
byte[] byteValues = { 12, 163, 255 };
foreach (byte byteValue in byteValues)
    Console.WriteLine(byteValue.ToString("X4"));
// The example displays the following output:
//      000C
//      00A3
//      00FF
```

```
Dim byteValues() As Byte = {12, 163, 255}
For Each byteValue As Byte In byteValues
    Console.WriteLine(byteValue.ToString("X4"))
Next
' The example displays the following output:
'      000C
'      00A3
'      00FF
```

For more information about standard numeric formatting strings, see [Standard Numeric Format Strings](#).

Standard format strings for date and time values are aliases for custom format strings stored by a particular [DateTimeFormatInfo](#) property. For example, calling the `ToString` method of a date and time value with the "D" format specifier displays the date and time by using the custom format string stored in the current culture's [DateTimeFormatInfo.LongDatePattern](#) property. (For more information about custom format strings, see the [next section](#).) The following example illustrates this relationship.

```
using System;
using System.Globalization;

public class Example
{
    public static void Main()
    {
        DateTime date1 = new DateTime(2009, 6, 30);
        Console.WriteLine("D Format Specifier:      {0:D}", date1);
        string longPattern = CultureInfo.CurrentCulture.DateTimeFormat.LongDatePattern;
        Console.WriteLine("'{0}' custom format string:      {1}",
                        longPattern, date1.ToString(longPattern));
    }
}
// The example displays the following output when run on a system whose
// current culture is en-US:
//      D Format Specifier:      Tuesday, June 30, 2009
//      'dddd, MMMM dd, yyyy' custom format string:      Tuesday, June 30, 2009
```

```

Imports System.Globalization

Module Example
    Public Sub Main()
        Dim date1 As Date = #6/30/2009#
        Console.WriteLine("D Format Specifier: {0:D}", date1)
        Dim longPattern As String = CultureInfo.CurrentCulture.DateTimeFormat.LongDatePattern
        Console.WriteLine("{0} custom format string: {1}",
                         longPattern, date1.ToString(longPattern))
    End Sub
End Module
' The example displays the following output when run on a system whose
' current culture is en-US:
'   D Format Specifier: Tuesday, June 30, 2009
'   'ddd, MMMM dd, yyyy' custom format string: Tuesday, June 30, 2009

```

For more information about standard date and time format strings, see [Standard Date and Time Format Strings](#).

You can also use standard format strings to define the string representation of an application-defined object that is produced by the object's `ToString(String)` method. You can define the specific standard format specifiers that your object supports, and you can determine whether they are case-sensitive or case-insensitive. Your implementation of the `ToString(String)` method should support the following:

- A "G" format specifier that represents a customary or common format of the object. The parameterless overload of your object's `ToString` method should call its `ToString(String)` overload and pass it the "G" standard format string.
- Support for a format specifier that is equal to a null reference (`Nothing` in Visual Basic). A format specifier that is equal to a null reference should be considered equivalent to the "G" format specifier.

For example, a `Temperature` class can internally store the temperature in degrees Celsius and use format specifiers to represent the value of the `Temperature` object in degrees Celsius, degrees Fahrenheit, and kelvins. The following example provides an illustration.

```

using System;

public class Temperature
{
    private decimal m_Temp;

    public Temperature(decimal temperature)
    {
        this.m_Temp = temperature;
    }

    public decimal Celsius
    {
        get { return this.m_Temp; }
    }

    public decimal Kelvin
    {
        get { return this.m_Temp + 273.15m; }
    }

    public decimal Fahrenheit
    {
        get { return Math.Round(((decimal) (this.m_Temp * 9 / 5 + 32)), 2); }
    }

    public override string ToString()
    {
        return this.ToString("C");
    }
}

```

```

}

public string ToString(string format)
{
    // Handle null or empty string.
    if (String.IsNullOrEmpty(format)) format = "C";
    // Remove spaces and convert to uppercase.
    format = format.Trim().ToUpperInvariant();

    // Convert temperature to Fahrenheit and return string.
    switch (format)
    {
        // Convert temperature to Fahrenheit and return string.
        case "F":
            return this.Fahrenheit.ToString("N2") + " °F";
        // Convert temperature to Kelvin and return string.
        case "K":
            return this.Kelvin.ToString("N2") + " K";
        // return temperature in Celsius.
        case "G":
        case "C":
            return this.Celsius.ToString("N2") + " °C";
        default:
            throw new FormatException(String.Format("The '{0}' format string is not supported.", format));
    }
}

public class Example1
{
    public static void Main()
    {
        Temperature temp1 = new Temperature(0m);
        Console.WriteLine(temp1.ToString());
        Console.WriteLine(temp1.ToString("G"));
        Console.WriteLine(temp1.ToString("C"));
        Console.WriteLine(temp1.ToString("F"));
        Console.WriteLine(temp1.ToString("K"));

        Temperature temp2 = new Temperature(-40m);
        Console.WriteLine(temp2.ToString());
        Console.WriteLine(temp2.ToString("G"));
        Console.WriteLine(temp2.ToString("C"));
        Console.WriteLine(temp2.ToString("F"));
        Console.WriteLine(temp2.ToString("K"));

        Temperature temp3 = new Temperature(16m);
        Console.WriteLine(temp3.ToString());
        Console.WriteLine(temp3.ToString("G"));
        Console.WriteLine(temp3.ToString("C"));
        Console.WriteLine(temp3.ToString("F"));
        Console.WriteLine(temp3.ToString("K"));

        Console.WriteLine(String.Format("The temperature is now {0:F}.", temp3));
    }
}

// The example displays the following output:
//      0.00 °C
//      0.00 °C
//      0.00 °C
//      32.00 °F
//      273.15 K
//      -40.00 °C
//      -40.00 °C
//      -40.00 °F
//      233.15 K
//      16.00 °C
//      16.00 °C

```

```
//      -
// 16.00 °C
// 60.80 °F
// 289.15 K
// The temperature is now 16.00 °C.
```

```
Public Class Temperature
    Private m_Temp As Decimal

    Public Sub New(temperature As Decimal)
        Me.m_Temp = temperature
    End Sub

    Public ReadOnly Property Celsius() As Decimal
        Get
            Return Me.m_Temp
        End Get
    End Property

    Public ReadOnly Property Kelvin() As Decimal
        Get
            Return Me.m_Temp + 273.15D
        End Get
    End Property

    Public ReadOnly Property Fahrenheit() As Decimal
        Get
            Return Math.Round(CDec(Me.m_Temp * 9 / 5 + 32), 2)
        End Get
    End Property

    Public Overrides Function ToString() As String
        Return Me.ToString("C")
    End Function

    Public Overloads Function ToString(format As String) As String
        ' Handle null or empty string.
        If String.IsNullOrEmpty(format) Then format = "C"
        ' Remove spaces and convert to uppercase.
        format = format.Trim().ToUpperInvariant()

        Select Case format
            Case "F"
                ' Convert temperature to Fahrenheit and return string.
                Return Me.Fahrenheit.ToString("N2") & " °F"
            Case "K"
                ' Convert temperature to Kelvin and return string.
                Return Me.Kelvin.ToString("N2") & " K"
            Case "C", "G"
                ' Return temperature in Celsius.
                Return Me.Celsius.ToString("N2") & " °C"
            Case Else
                Throw New FormatException(String.Format("The '{0}' format string is not supported.", format))
        End Select
    End Function
End Class

Public Module Example1
    Public Sub Main()
        Dim temp1 As New Temperature(0D)
        Console.WriteLine(temp1.ToString())
        Console.WriteLine(temp1.ToString("G"))
        Console.WriteLine(temp1.ToString("C"))
        Console.WriteLine(temp1.ToString("F"))
        Console.WriteLine(temp1.ToString("K"))

        Dim temp2 As New Temperature(-40D)
```

```

Console.WriteLine(temp2.ToString())
Console.WriteLine(temp2.ToString("G"))
Console.WriteLine(temp2.ToString("C"))
Console.WriteLine(temp2.ToString("F"))
Console.WriteLine(temp2.ToString("K"))

Dim temp3 As New Temperature(16D)
Console.WriteLine(temp3.ToString())
Console.WriteLine(temp3.ToString("G"))
Console.WriteLine(temp3.ToString("C"))
Console.WriteLine(temp3.ToString("F"))
Console.WriteLine(temp3.ToString("K"))

Console.WriteLine(String.Format("The temperature is now {0:F}.", temp3))
End Sub
End Module
' The example displays the following output:
'   0.00 °C
'   0.00 °C
'   0.00 °C
'   32.00 °F
'   273.15 K
'   -40.00 °C
'   -40.00 °C
'   -40.00 °F
'   233.15 K
'   16.00 °C
'   16.00 °C
'   16.00 °C
'   60.80 °F
'   289.15 K
'   The temperature is now 16.00 °C.

```

Custom format strings

In addition to the standard format strings, .NET defines custom format strings for both numeric values and date and time values. A custom format string consists of one or more custom format specifiers that define the string representation of a value. For example, the custom date and time format string "yyyy/mm/dd hh:mm:ss.fffft zzz" converts a date to its string representation in the form "2008/11/15 07:45:00.0000 P -08:00" for the en-US culture. Similarly, the custom format string "0000" converts the integer value 12 to "0012". For a complete list of custom format strings, see [Custom Date and Time Format Strings](#) and [Custom Numeric Format Strings](#).

If a format string consists of a single custom format specifier, the format specifier should be preceded by the percent (%) symbol to avoid confusion with a standard format specifier. The following example uses the "M" custom format specifier to display a one-digit or two-digit number of the month of a particular date.

```

DateTime date1 = new DateTime(2009, 9, 8);
Console.WriteLine(date1.ToString("%M"));           // Displays 9

```

```

Dim date1 As Date = #09/08/2009#
Console.WriteLine(date1.ToString("%M"))           ' Displays 9

```

Many standard format strings for date and time values are aliases for custom format strings that are defined by properties of the [DateTimeFormatInfo](#) object. Custom format strings also offer considerable flexibility in providing application-defined formatting for numeric values or date and time values. You can define your own custom result strings for both numeric values and date and time values by combining multiple custom format specifiers into a single custom format string. The following example defines a custom format string that displays the day of the week in parentheses after the month name, day, and year.

```

string customFormat = "MMMM dd, yyyy (dddd)";
DateTime date1 = new DateTime(2009, 8, 28);
Console.WriteLine(date1.ToString(customFormat));
// The example displays the following output if run on a system
// whose language is English:
//      August 28, 2009 (Friday)

```

```

Dim customFormat As String = "MMMM dd, yyyy (dddd)"
Dim date1 As Date = #8/28/2009#
Console.WriteLine(date1.ToString(customFormat))
' The example displays the following output if run on a system
' whose language is English:
'      August 28, 2009 (Friday)

```

The following example defines a custom format string that displays an [Int64](#) value as a standard, seven-digit U.S. telephone number along with its area code.

```

using System;

public class Example17
{
    public static void Main()
    {
        long number = 800999999;
        string fmt = "000-000-0000";
        Console.WriteLine(number.ToString(fmt));
    }
}
// The example displays the following output:
//      800-999-9999

```

```

Module Example18
    Public Sub Main18()
        Dim number As Long = 800999999
        Dim fmt As String = "000-000-0000"
        Console.WriteLine(number.ToString(fmt))
    End Sub
End Module
' The example displays the following output:
'
' The example displays the following output:
'      800-999-9999

```

Although standard format strings can generally handle most of the formatting needs for your application-defined types, you may also define custom format specifiers to format your types.

Format strings and .NET types

All numeric types (that is, the [Byte](#), [Decimal](#), [Double](#), [Int16](#), [Int32](#), [Int64](#), [SByte](#), [Single](#), [UInt16](#), [UInt32](#), [UInt64](#), and [BigInteger](#) types), as well as the [DateTime](#), [DateTimeOffset](#), [TimeSpan](#), [Guid](#), and all enumeration types, support formatting with format strings. For information on the specific format strings supported by each type, see the following topics:

TITLE	DEFINITION
Standard Numeric Format Strings	Describes standard format strings that create commonly used string representations of numeric values.

TITLE	DEFINITION
Custom Numeric Format Strings	Describes custom format strings that create application-specific formats for numeric values.
Standard Date and Time Format Strings	Describes standard format strings that create commonly used string representations of DateTime and DateTimeOffset values.
Custom Date and Time Format Strings	Describes custom format strings that create application-specific formats for DateTime and DateTimeOffset values.
Standard TimeSpan Format Strings	Describes standard format strings that create commonly used string representations of time intervals.
Custom TimeSpan Format Strings	Describes custom format strings that create application-specific formats for time intervals.
Enumeration Format Strings	Describes standard format strings that are used to create string representations of enumeration values.
Guid.ToString(String)	Describes standard format strings for Guid values.

Culture-sensitive formatting with format providers

Although format specifiers let you customize the formatting of objects, producing a meaningful string representation of objects often requires additional formatting information. For example, formatting a number as a currency value by using either the "C" standard format string or a custom format string such as "\$ #,##0" requires, at a minimum, information about the correct currency symbol, group separator, and decimal separator to be available to include in the formatted string. In .NET, this additional formatting information is made available through the [IFormatProvider](#) interface, which is provided as a parameter to one or more overloads of the [ToString](#) method of numeric types and date and time types. [IFormatProvider](#) implementations are used in .NET to support culture-specific formatting. The following example illustrates how the string representation of an object changes when it is formatted with three [IFormatProvider](#) objects that represent different cultures.

```

using System;
using System.Globalization;

public class Example18
{
    public static void Main()
    {
        decimal value = 1603.42m;
        Console.WriteLine(value.ToString("C3", new CultureInfo("en-US")));
        Console.WriteLine(value.ToString("C3", new CultureInfo("fr-FR")));
        Console.WriteLine(value.ToString("C3", new CultureInfo("de-DE")));
    }
}
// The example displays the following output:
//      $1,603.420
//      1 603,420 €
//      1.603,420 €

```

```

Imports System.Globalization

Public Module Example11
    Public Sub Main11()
        Dim value As Decimal = 1603.42D
        Console.WriteLine(value.ToString("C3", New CultureInfo("en-US")))
        Console.WriteLine(value.ToString("C3", New CultureInfo("fr-FR")))
        Console.WriteLine(value.ToString("C3", New CultureInfo("de-DE")))
    End Sub
End Module
' The example displays the following output:
'      $1,603.420
'      1 603,420 €
'      1.603,420 €

```

The [IFormatProvider](#) interface includes one method, [GetFormat\(Type\)](#), which has a single parameter that specifies the type of object that provides formatting information. If the method can provide an object of that type, it returns it. Otherwise, it returns a null reference ([Nothing](#) in Visual Basic).

[IFormatProvider.GetFormat](#) is a callback method. When you call a [ToString](#) method overload that includes an [IFormatProvider](#) parameter, it calls the [GetFormat](#) method of that [IFormatProvider](#) object. The [GetFormat](#) method is responsible for returning an object that provides the necessary formatting information, as specified by its [formatType](#) parameter, to the [ToString](#) method.

A number of formatting or string conversion methods include a parameter of type [IFormatProvider](#), but in many cases the value of the parameter is ignored when the method is called. The following table lists some of the formatting methods that use the parameter and the type of the [Type](#) object that they pass to the [IFormatProvider.GetFormat](#) method.

METHOD	TYPE OF FORMATTYPE PARAMETER
ToString method of numeric types	System.Globalization.NumberFormatInfo
ToString method of date and time types	System.Globalization.DateTimeFormatInfo
String.Format	System.ICustomFormatter
StringBuilder.AppendFormat	System.ICustomFormatter

NOTE

The [ToString](#) methods of the numeric types and date and time types are overloaded, and only some of the overloads include an [IFormatProvider](#) parameter. If a method does not have a parameter of type [IFormatProvider](#), the object that is returned by the [CultureInfo.CurrentCulture](#) property is passed instead. For example, a call to the default [Int32.ToString\(\)](#) method ultimately results in a method call such as the following:

```
Int32.ToString("G", System.Globalization.CultureInfo.CurrentCulture) .
```

.NET provides three classes that implement [IFormatProvider](#):

- [DateTimeFormatInfo](#), a class that provides formatting information for date and time values for a specific culture. Its [IFormatProvider.GetFormat](#) implementation returns an instance of itself.
- [NumberFormatInfo](#), a class that provides numeric formatting information for a specific culture. Its [IFormatProvider.GetFormat](#) implementation returns an instance of itself.
- [CultureInfo](#). Its [IFormatProvider.GetFormat](#) implementation can return either a [NumberFormatInfo](#) object

to provide numeric formatting information or a [DateTimeFormatInfo](#) object to provide formatting information for date and time values.

You can also implement your own format provider to replace any one of these classes. However, your implementation's [GetFormat](#) method must return an object of the type listed in the previous table if it has to provide formatting information to the [ToString](#) method.

Culture-sensitive formatting of numeric values

By default, the formatting of numeric values is culture-sensitive. If you do not specify a culture when you call a formatting method, the formatting conventions of the current culture are used. This is illustrated in the following example, which changes the current culture four times and then calls the [Decimal.ToString\(String\)](#) method. In each case, the result string reflects the formatting conventions of the current culture. This is because the [ToString](#) and [ToString\(String\)](#) methods wrap calls to each numeric type's [ToString\(String, IFormatProvider\)](#) method.

```
using System.Globalization;

public class Example6
{
    public static void Main()
    {
        string[] cultureNames = { "en-US", "fr-FR", "es-MX", "de-DE" };
        Decimal value = 1043.17m;

        foreach (var cultureName in cultureNames) {
            // Change the current culture.
            CultureInfo.CurrentCulture = CultureInfo.CreateSpecificCulture(cultureName);
            Console.WriteLine($"The current culture is {CultureInfo.CurrentCulture.Name}");
            Console.WriteLine(value.ToString("C2"));
            Console.WriteLine();
        }
    }
}

// The example displays the following output:
//      The current culture is en-US
//      $1,043.17
//
//      The current culture is fr-FR
//      1 043,17 €
//
//      The current culture is es-MX
//      $1,043.17
//
//      The current culture is de-DE
//      1.043,17 €
```

```

Imports System.Globalization

Module Example6
    Public Sub Main6()
        Dim cultureNames() As String = {"en-US", "fr-FR", "es-MX", "de-DE"}
        Dim value As Decimal = 1043.17D

        For Each cultureName In cultureNames
            ' Change the current culture.
            CultureInfo.CurrentCulture = CultureInfo.CreateSpecificCulture(cultureName)
            Console.WriteLine($"The current culture is {CultureInfo.CurrentCulture.Name}")
            Console.WriteLine(value.ToString("C2"))
            Console.WriteLine()

            Next
        End Sub
    End Module
    ' The example displays the following output:
    '   The current culture is en-US
    '   $1,043.17
    '
    '   The current culture is fr-FR
    '   1 043,17 €
    '
    '   The current culture is es-MX
    '   $1,043.17
    '
    '   The current culture is de-DE
    '   1.043,17 €

```

You can also format a numeric value for a specific culture by calling a `ToString` overload that has a `IFormatProvider` parameter and passing it either of the following:

- A `CultureInfo` object that represents the culture whose formatting conventions are to be used. Its `CultureInfo.GetFormat` method returns the value of the `CultureInfo.NumberFormat` property, which is the `NumberFormatInfo` object that provides culture-specific formatting information for numeric values.
- A `NumberFormatInfo` object that defines the culture-specific formatting conventions to be used. Its `GetFormat` method returns an instance of itself.

The following example uses `NumberFormatInfo` objects that represent the English (United States) and English (Great Britain) cultures and the French and Russian neutral cultures to format a floating-point number.

```

using System.Globalization;

public class Example7
{
    public static void Main()
    {
        double value = 1043.62957;
        string[] cultureNames = { "en-US", "en-GB", "ru", "fr" };

        foreach (string? name in cultureNames)
        {
            NumberFormatInfo nfi = CultureInfo.CreateSpecificCulture(name).NumberFormat;
            Console.WriteLine("{0,-6} {1}", name + ":", value.ToString("N3", nfi));
        }
    }
    // The example displays the following output:
    //   en-US: 1,043.630
    //   en-GB: 1,043.630
    //   ru:     1 043,630
    //   fr:     1 043,630

```

```

Imports System.Globalization

Module Example7
    Public Sub Main7()
        Dim value As Double = 1043.62957
        Dim cultureNames() As String = {"en-US", "en-GB", "ru", "fr"}

        For Each name In cultureNames
            Dim nfi As NumberFormatInfo = CultureInfo.CreateSpecificCulture(name).NumberFormat
            Console.WriteLine("{0,-6} {1}", name + ":", value.ToString("N3", nfi))
        Next
    End Sub
End Module
' The example displays the following output:
'      en-US: 1,043.630
'      en-GB: 1,043.630
'      ru:     1 043,630
'      fr:     1 043,630

```

Culture-sensitive formatting of date and time values

By default, the formatting of date and time values is culture-sensitive. If you do not specify a culture when you call a formatting method, the formatting conventions of the current culture are used. This is illustrated in the following example, which changes the current culture four times and then calls the [DateTime.ToString\(String\)](#) method. In each case, the result string reflects the formatting conventions of the current culture. This is because the [DateTime.ToString\(\)](#), [DateTime.ToString\(String\)](#), [DateTimeOffset.ToString\(\)](#), and [DateTimeOffset.ToString\(String\)](#) methods wrap calls to the [DateTime.ToString\(String, IFormatProvider\)](#) and [DateTimeOffset.ToString\(String, IFormatProvider\)](#) methods.

```

using System.Globalization;

public class Example4
{
    public static void Main()
    {
        string[] cultureNames = { "en-US", "fr-FR", "es-MX", "de-DE" };
        DateTime dateToFormat = new DateTime(2012, 5, 28, 11, 30, 0);

        foreach (var cultureName in cultureNames) {
            // Change the current culture.
            CultureInfo.CurrentCulture = CultureInfo.CreateSpecificCulture(cultureName);
            Console.WriteLine($"The current culture is {CultureInfo.CurrentCulture.Name}");
            Console.WriteLine(dateToFormat.ToString("F"));
            Console.WriteLine();
        }
    }
}
// The example displays the following output:
//      The current culture is en-US
//      Monday, May 28, 2012 11:30:00 AM
//
//      The current culture is fr-FR
//      lundi 28 mai 2012 11:30:00
//
//      The current culture is es-MX
//      lunes, 28 de mayo de 2012 11:30:00 a.m.
//
//      The current culture is de-DE
//      Montag, 28. Mai 2012 11:30:00

```

```

Imports System.Globalization
Imports System.Threading

Module Example4
    Public Sub Main4()
        Dim cultureNames() As String = {"en-US", "fr-FR", "es-MX", "de-DE"}
        Dim dateToFormat As Date = #5/28/2012 11:30AM#

        For Each cultureName In cultureNames
            ' Change the current culture.
            CultureInfo.CurrentCulture = CultureInfo.CreateSpecificCulture(cultureName)
            Console.WriteLine($"The current culture is {CultureInfo.CurrentCulture.Name}")
            Console.WriteLine(dateToFormat.ToString("F"))
            Console.WriteLine()
        Next
    End Sub
End Module
' The example displays the following output:
'   The current culture is en-US
'   Monday, May 28, 2012 11:30:00 AM
'
'   The current culture is fr-FR
'   lundi 28 mai 2012 11:30:00
'
'   The current culture is es-MX
'   lunes, 28 de mayo de 2012 11:30:00 a.m.
'
'   The current culture is de-DE
'   Montag, 28. Mai 2012 11:30:00

```

You can also format a date and time value for a specific culture by calling a [DateTime.ToString](#) or [DateTimeOffset.ToString](#) overload that has a `provider` parameter and passing it either of the following:

- A [CultureInfo](#) object that represents the culture whose formatting conventions are to be used. Its [CultureInfo.GetFormat](#) method returns the value of the [CultureInfo.DateTimeFormat](#) property, which is the [DateTimeFormatInfo](#) object that provides culture-specific formatting information for date and time values.
- A [DateTimeFormatInfo](#) object that defines the culture-specific formatting conventions to be used. Its [GetFormat](#) method returns an instance of itself.

The following example uses [DateTimeFormatInfo](#) objects that represent the English (United States) and English (Great Britain) cultures and the French and Russian neutral cultures to format a date.

```

using System.Globalization;

public class Example5
{
    public static void Main()
    {
        DateTime dat1 = new(2012, 5, 28, 11, 30, 0);
        string[] cultureNames = { "en-US", "en-GB", "ru", "fr" };

        foreach (var name in cultureNames) {
            DateTimeFormatInfo dtfi = CultureInfo.CreateSpecificCulture(name).DateTimeFormat;
            Console.WriteLine($"{name}: {dat1.ToString(dtfi)}");
        }
    }
}

// The example displays the following output:
//      en-US: 5/28/2012 11:30:00 AM
//      en-GB: 28/05/2012 11:30:00
//      ru: 28.05.2012 11:30:00
//      fr: 28/05/2012 11:30:00

```

```

Imports System.Globalization

Module Example5
    Public Sub Main()
        Dim dat1 As Date = #5/28/2012 11:30AM#
        Dim cultureNames() As String = {"en-US", "en-GB", "ru", "fr"}

        For Each name In cultureNames
            Dim dtfi As DateTimeFormatInfo = CultureInfo.CreateSpecificCulture(name).DateTimeFormat
            Console.WriteLine($"{name}: {dat1.ToString(dtfi)}")
        Next
    End Sub
End Module

' The example displays the following output:
'      en-US: 5/28/2012 11:30:00 AM
'      en-GB: 28/05/2012 11:30:00
'      ru: 28.05.2012 11:30:00
'      fr: 28/05/2012 11:30:00

```

The [IFormattable](#) interface

Typically, types that overload the `ToString` method with a format string and an [IFormatProvider](#) parameter also implement the [IFormattable](#) interface. This interface has a single member, [IFormattable.ToString\(String, IFormatProvider\)](#), that includes both a format string and a format provider as parameters.

Implementing the [IFormattable](#) interface for your application-defined class offers two advantages:

- Support for string conversion by the [Convert](#) class. Calls to the [Convert.ToString\(Object\)](#) and [Convert.ToString\(Object, IFormatProvider\)](#) methods call your [IFormattable](#) implementation automatically.
- Support for composite formatting. If a format item that includes a format string is used to format your custom type, the common language runtime automatically calls your [IFormattable](#) implementation and passes it the format string. For more information about composite formatting with methods such as [String.Format](#) or [Console.WriteLine](#), see the [Composite Formatting](#) section.

The following example defines a `Temperature` class that implements the [IFormattable](#) interface. It supports the "C" or "G" format specifiers to display the temperature in Celsius, the "F" format specifier to display the temperature in Fahrenheit, and the "K" format specifier to display the temperature in Kelvin.

```

using System;
using System.Globalization;

namespace HotAndCold
{

    public class Temperature : IFormattable
    {
        private decimal m_Temp;

        public Temperature(decimal temperature)
        {
            this.m_Temp = temperature;
        }

        public decimal Celsius
        {
            get { return this.m_Temp; }
        }

        public decimal Kelvin
        {
            get { return this.m_Temp + 273.15m; }
        }

        public decimal Fahrenheit
        {
            get { return Math.Round((decimal)this.m_Temp * 9 / 5 + 32, 2); }
        }

        public override string ToString()
        {
            return this.ToString("G", null);
        }

        public string ToString(string format)
        {
            return this.ToString(format, null);
        }

        public string ToString(string format, IFormatProvider provider)
        {
            // Handle null or empty arguments.
            if (String.IsNullOrEmpty(format))
                format = "G";
            // Remove any white space and convert to uppercase.
            format = format.Trim().ToUpperInvariant();

            if (provider == null)
                provider = NumberFormatInfo.CurrentInfo;

            switch (format)
            {
                // Convert temperature to Fahrenheit and return string.
                case "F":
                    return this.Fahrenheit.ToString("N2", provider) + °F";
                // Convert temperature to Kelvin and return string.
                case "K":
                    return this.Kelvin.ToString("N2", provider) + "K";
                // Return temperature in Celsius.
                case "C":
                case "G":
                    return this.Celsius.ToString("N2", provider) + °C";
                default:
                    throw new FormatException(String.Format("The '{0}' format string is not supported.", format));
            }
        }
    }
}

```

```

Public Class Temperature : Implements IFormattable
    Private m_Temp As Decimal

    Public Sub New(temperature As Decimal)
        Me.m_Temp = temperature
    End Sub

    Public ReadOnly Property Celsius() As Decimal
        Get
            Return Me.m_Temp
        End Get
    End Property

    Public ReadOnly Property Kelvin() As Decimal
        Get
            Return Me.m_Temp + 273.15D
        End Get
    End Property

    Public ReadOnly Property Fahrenheit() As Decimal
        Get
            Return Math.Round(CDec(Me.m_Temp * 9 / 5 + 32), 2)
        End Get
    End Property

    Public Overrides Function ToString() As String
        Return Me.ToString("G", Nothing)
    End Function

    Public Overloads Function ToString(format As String) As String
        Return Me.ToString(format, Nothing)
    End Function

    Public Overloads Function ToString(format As String, provider As IFormatProvider) As String _
        Implements IFormattable.ToString

        ' Handle null or empty arguments.
        If String.IsNullOrEmpty(format) Then format = "G"
        ' Remove any white space and convert to uppercase.
        format = format.Trim().ToUpperInvariant()

        If provider Is Nothing Then provider = NumberFormatInfo.CurrentInfo

        Select Case format
            ' Convert temperature to Fahrenheit and return string.
            Case "F"
                Return Me.Fahrenheit.ToString("N2", provider) & °F"
            ' Convert temperature to Kelvin and return string.
            Case "K"
                Return Me.Kelvin.ToString("N2", provider) & "K"
            ' Return temperature in Celsius.
            Case "C", "G"
                Return Me.Celsius.ToString("N2", provider) & °C"
            Case Else
                Throw New FormatException(String.Format($"The '{format}' format string is not supported."))
        End Select
    End Function
End Class

```

The following example instantiates a `Temperature` object. It then calls the `ToString` method and uses several composite format strings to obtain different string representations of a `Temperature` object. Each of these method calls, in turn, calls the `IFormattable` implementation of the `Temperature` class.

```

public class Example11
{
    public static void Main()
    {
        CultureInfo.CurrentCulture = CultureInfo.GetCultureInfo("en-US");
        Temperature temp = new Temperature(22m);
        Console.WriteLine(Convert.ToString(temp, new CultureInfo("ja-JP")));
        Console.WriteLine("Temperature: {0:K}", temp);
        Console.WriteLine("Temperature: {0:F}", temp);
        Console.WriteLine(String.Format(new CultureInfo("fr-FR"), "Temperature: {0:F}", temp));
    }
}

// The example displays the following output:
//      22.00°C
//      Temperature: 295.15K
//      Temperature: 71.60°F
//      Temperature: 71,60°F

```

```

Public Module Example12
    Public Sub Main12()
        Dim temp As New Temperature(22D)
        CultureInfo.CurrentCulture = CultureInfo.GetCultureInfo("en-US")
        Console.WriteLine(Convert.ToString(temp, New CultureInfo("ja-JP")))
        Console.WriteLine($"Temperature: {temp:K}")
        Console.WriteLine($"Temperature: {temp:F}")
        Console.WriteLine(String.Format(New CultureInfo("fr-FR"), $"Temperature: {temp:F}"))
    End Sub
End Module

' The example displays the following output:
'      22.00°C
'      Temperature: 295.15K
'      Temperature: 71.60°F
'      Temperature: 71,60°F

```

Composite formatting

Some methods, such as [String.Format](#) and [StringBuilder.AppendFormat](#), support composite formatting. A composite format string is a kind of template that returns a single string that incorporates the string representation of zero, one, or more objects. Each object is represented in the composite format string by an indexed format item. The index of the format item corresponds to the position of the object that it represents in the method's parameter list. Indexes are zero-based. For example, in the following call to the [String.Format](#) method, the first format item, `{0:D}`, is replaced by the string representation of `thatDate`; the second format item, `{1}`, is replaced by the string representation of `item1`; and the third format item, `{2:C2}`, is replaced by the string representation of `item1.Value`.

```

result = String.Format("On {0:d}, the inventory of {1} was worth {2:C2}.",
    thatDate, item1, item1.Value);
Console.WriteLine(result);
// The example displays output like the following if run on a system
// whose current culture is en-US:
//      On 5/1/2009, the inventory of WidgetA was worth $107.44.

```

```

result = String.Format("On {0:d}, the inventory of {1} was worth {2:C2}.",
    thatDate, item1, item1.Value)
Console.WriteLine(result)
' The example displays output like the following if run on a system
' whose current culture is en-US:
'      On 5/1/2009, the inventory of WidgetA was worth $107.44.

```

In addition to replacing a format item with the string representation of its corresponding object, format items also let you control the following:

- The specific way in which an object is represented as a string, if the object implements the [IFormattable](#) interface and supports format strings. You do this by following the format item's index with a `:` (colon) followed by a valid format string. The previous example did this by formatting a date value with the "d" (short date pattern) format string (e.g., `{0:d}`) and by formatting a numeric value with the "C2" format string (e.g., `{2:c2}`) to represent the number as a currency value with two fractional decimal digits.
- The width of the field that contains the object's string representation, and the alignment of the string representation in that field. You do this by following the format item's index with a `,` (comma) followed by the field width. The string is right-aligned in the field if the field width is a positive value, and it is left-aligned if the field width is a negative value. The following example left-aligns date values in a 20-character field, and it right-aligns decimal values with one fractional digit in an 11-character field.

```
DateTime startDate = new DateTime(2015, 8, 28, 6, 0, 0);
decimal[] temps = { 73.452m, 68.98m, 72.6m, 69.24563m,
                    74.1m, 72.156m, 72.228m };
Console.WriteLine("{0,-20} {1,11}\n", "Date", "Temperature");
for (int ctr = 0; ctr < temps.Length; ctr++)
    Console.WriteLine("{0,-20:g} {1,11:N1}", startDate.AddDays(ctr), temps[ctr]);

// The example displays the following output:
//      Date                  Temperature
//
//      8/28/2015 6:00 AM      73.5
//      8/29/2015 6:00 AM      69.0
//      8/30/2015 6:00 AM      72.6
//      8/31/2015 6:00 AM      69.2
//      9/1/2015 6:00 AM       74.1
//      9/2/2015 6:00 AM       72.2
//      9/3/2015 6:00 AM       72.2
```

```
Dim startDate As New Date(2015, 8, 28, 6, 0, 0)
Dim temps() As Decimal = {73.452, 68.98, 72.6, 69.24563,
                           74.1, 72.156, 72.228}
Console.WriteLine("{0,-20} {1,11}", "Date", "Temperature")
Console.WriteLine()
For ctr As Integer = 0 To temps.Length - 1
    Console.WriteLine("{0,-20:g} {1,11:N1}", startDate.AddDays(ctr), temps(ctr))
Next
' The example displays the following output:
'      Date                  Temperature
'
'      8/28/2015 6:00 AM      73.5
'      8/29/2015 6:00 AM      69.0
'      8/30/2015 6:00 AM      72.6
'      8/31/2015 6:00 AM      69.2
'      9/1/2015 6:00 AM       74.1
'      9/2/2015 6:00 AM       72.2
'      9/3/2015 6:00 AM       72.2
```

Note that, if both the alignment string component and the format string component are present, the former precedes the latter (for example, `{0,-20:g}`).

For more information about composite formatting, see [Composite Formatting](#).

Custom formatting with [ICustomFormatter](#)

Two composite formatting methods, [String.Format\(IFormatProvider, String, Object\[\]\)](#) and

`StringBuilder.AppendFormat(IFormatProvider, String, Object[])`, include a format provider parameter that supports custom formatting. When either of these formatting methods is called, it passes a `Type` object that represents an `ICustomFormatter` interface to the format provider's `GetFormat` method. The `GetFormat` method is then responsible for returning the `ICustomFormatter` implementation that provides custom formatting.

The `ICustomFormatter` interface has a single method, `Format(String, Object, IFormatProvider)`, that is called automatically by a composite formatting method, once for each format item in a composite format string. The `Format(String, Object, IFormatProvider)` method has three parameters: a format string, which represents the `formatString` argument in a format item, an object to format, and an `IFormatProvider` object that provides formatting services. Typically, the class that implements `ICustomFormatter` also implements `IFormatProvider`, so this last parameter is a reference to the custom formatting class itself. The method returns a custom formatted string representation of the object to be formatted. If the method cannot format the object, it should return a null reference (`Nothing` in Visual Basic).

The following example provides an `ICustomFormatter` implementation named `ByteByByteFormatter` that displays integer values as a sequence of two-digit hexadecimal values followed by a space.

```
public class ByteByByteFormatter : IFormatProvider, ICustomFormatter
{
    public object GetFormat(Type formatType)
    {
        if (formatType == typeof(ICustomFormatter))
            return this;
        else
            return null;
    }

    public string Format(string format, object arg,
                         IFormatProvider formatProvider)
    {
        if (! formatProvider.Equals(this)) return null;

        // Handle only hexadecimal format string.
        if (! format.StartsWith("X")) return null;

        byte[] bytes;
        string output = null;

        // Handle only integral types.
        if (arg is Byte)
            bytes = BitConverter.GetBytes((Byte) arg);
        else if (arg is Int16)
            bytes = BitConverter.GetBytes((Int16) arg);
        else if (arg is Int32)
            bytes = BitConverter.GetBytes((Int32) arg);
        else if (arg is Int64)
            bytes = BitConverter.GetBytes((Int64) arg);
        else if (arg is SByte)
            bytes = BitConverter.GetBytes((SByte) arg);
        else if (arg is UInt16)
            bytes = BitConverter.GetBytes((UInt16) arg);
        else if (arg is UInt32)
            bytes = BitConverter.GetBytes((UInt32) arg);
        else if (arg is UInt64)
            bytes = BitConverter.GetBytes((UInt64) arg);
        else
            return null;

        for (int ctr = bytes.Length - 1; ctr >= 0; ctr--)
            output += String.Format("{0:X2} ", bytes[ctr]);

        return output.Trim();
    }
}
```

```

Public Class ByteByByteFormatter : Implements IFormatProvider, ICustomFormatter
    Public Function GetFormat(formatType As Type) As Object _
        Implements IFormatProvider.GetFormat
        If formatType Is GetType(ICustomFormatter) Then
            Return Me
        Else
            Return Nothing
        End If
    End Function

    Public Function Format(fmt As String, arg As Object,
        formatProvider As IFormatProvider) As String _
        Implements ICustomFormatter.Format

        If Not formatProvider.Equals(Me) Then Return Nothing

        ' Handle only hexadecimal format string.
        If Not fmt.StartsWith("X") Then
            Return Nothing
        End If

        ' Handle only integral types.
        If Not typeof(arg) Is Byte AndAlso
            Not typeof(arg) Is Int16 AndAlso
            Not typeof(arg) Is Int32 AndAlso
            Not typeof(arg) Is Int64 AndAlso
            Not typeof(arg) Is SByte AndAlso
            Not typeof(arg) Is UInt16 AndAlso
            Not typeof(arg) Is UInt32 AndAlso
            Not typeof(arg) Is UInt64 Then _
                Return Nothing

        Dim bytes() As Byte = BitConverter.GetBytes(arg)
        Dim output As String = Nothing

        For ctr As Integer = bytes.Length - 1 To 0 Step -1
            output += String.Format("{0:X2} ", bytes(ctr))
        Next

        Return output.Trim()
    End Function
End Class

```

The following example uses the `ByteByByteFormatter` class to format integer values. Note that the `ICustomFormatter.Format` method is called more than once in the second `String.Format(IFormatProvider, String, Object[])` method call, and that the default `NumberFormatInfo` provider is used in the third method call because the `.ByteByByteFormatter.Format` method does not recognize the "N0" format string and returns a null reference (`Nothing` in Visual Basic).

```

public class Example10
{
    public static void Main()
    {
        long value = 3210662321;
        byte value1 = 214;
        byte value2 = 19;

        Console.WriteLine(String.Format(new ByteByByteFormatter(), "{0:X}", value));
        Console.WriteLine(String.Format(new ByteByByteFormatter(), "{0:X} And {1:X} = {2:X} ({2:000})", value1, value2, value1 & value2));
        Console.WriteLine(String.Format(new ByteByByteFormatter(), "{0,10:N0}", value));
    }
}

// The example displays the following output:
//      00 00 00 00 BF 5E D1 B1
//      00 D6 And 00 13 = 00 12 (018)
//      3,210,662,321

```

```

Public Module Example10
    Public Sub Main10()
        Dim value As Long = 3210662321
        Dim value1 As Byte = 214
        Dim value2 As Byte = 19

        Console.WriteLine((String.Format(New ByteByByteFormatter(), "{0:X}", value)))
        Console.WriteLine((String.Format(New ByteByByteFormatter(), "{0:X} And {1:X} = {2:X} ({2:000})", value1, value2, value1 And value2)))
        Console.WriteLine(String.Format(New ByteByByteFormatter(), "{0,10:N0}", value))
    End Sub
End Module

' The example displays the following output:
'      00 00 00 00 BF 5E D1 B1
'      00 D6 And 00 13 = 00 12 (018)
'      3,210,662,321

```

See also

TITLE	DEFINITION
Standard Numeric Format Strings	Describes standard format strings that create commonly used string representations of numeric values.
Custom Numeric Format Strings	Describes custom format strings that create application-specific formats for numeric values.
Standard Date and Time Format Strings	Describes standard format strings that create commonly used string representations of DateTime values.
Custom Date and Time Format Strings	Describes custom format strings that create application-specific formats for DateTime values.
Standard TimeSpan Format Strings	Describes standard format strings that create commonly used string representations of time intervals.
Custom TimeSpan Format Strings	Describes custom format strings that create application-specific formats for time intervals.

TITLE	DEFINITION
Enumeration Format Strings	Describes standard format strings that are used to create string representations of enumeration values.
Composite Formatting	Describes how to embed one or more formatted values in a string. The string can subsequently be displayed on the console or written to a stream.
Parsing Strings	Describes how to initialize objects to the values described by string representations of those objects. Parsing is the inverse operation of formatting.

Reference

- [System.IFormattable](#)
- [System.IFormatProvider](#)
- [System.ICustomFormatter](#)

Standard numeric format strings

9/20/2022 • 27 minutes to read • [Edit Online](#)

Standard numeric format strings are used to format common numeric types. A standard numeric format string takes the form `[format specifier][precision specifier]`, where:

- *Format specifier* is a single alphabetic character that specifies the type of number format, for example, currency or percent. Any numeric format string that contains more than one alphabetic character, including white space, is interpreted as a custom numeric format string. For more information, see [Custom numeric format strings](#).
- *Precision specifier* is an optional integer that affects the number of digits in the resulting string. In .NET 7 and later versions, the maximum precision value is 999,999,999. In .NET 6, the maximum precision value is [Int32.MaxValue](#). In previous .NET versions, the precision can range from 0 to 99. The precision specifier controls the number of digits in the string representation of a number. It does not round the number itself. To perform a rounding operation, use the [Math.Ceiling](#), [Math.Floor](#), or [Math.Round](#) method.

When *precision specifier* controls the number of fractional digits in the result string, the result string reflects a number that is rounded to a representable result nearest to the infinitely precise result. If there are two equally near representable results:

- On [.NET Framework and .NET Core up to .NET Core 2.0](#), the runtime selects the result with the greater least significant digit (that is, using [MidpointRounding.AwayFromZero](#)).
- On [.NET Core 2.1 and later](#), the runtime selects the result with an even least significant digit (that is, using [MidpointRounding.ToEven](#)).

NOTE

The precision specifier determines the number of digits in the result string. To pad a result string with leading or trailing spaces, use the [composite formatting](#) feature and define an *alignment component* in the format item.

Standard numeric format strings are supported by:

- Some overloads of the `ToString` method of all numeric types. For example, you can supply a numeric format string to the [Int32.ToString\(String\)](#) and [Int32.ToString\(String, IFormatProvider\)](#) methods.
- The `TryFormat` method of all numeric types, for example, [Int32.TryFormat\(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider\)](#) and [Single.TryFormat\(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider\)](#).
- The .NET [composite formatting feature](#), which is used by some `Write` and `WriteLine` methods of the [Console](#) and [StreamWriter](#) classes, the [String.Format](#) method, and the [StringBuilder.AppendFormat](#) method. The composite format feature allows you to include the string representation of multiple data items in a single string, to specify field width, and to align numbers in a field. For more information, see [Composite Formatting](#).
- [Interpolated strings](#) in C# and Visual Basic, which provide a simplified syntax when compared to composite format strings.

TIP

You can download the [Formatting Utility](#), a .NET Core Windows Forms application that lets you apply format strings to either numeric or date and time values and displays the result string. Source code is available for [C#](#) and [Visual Basic](#).

Standard format specifiers

The following table describes the standard numeric format specifiers and displays sample output produced by each format specifier. See the [Notes](#) section for additional information about using standard numeric format strings, and the [Code example](#) section for a comprehensive illustration of their use.

The result of a formatted string for a specific culture might differ from the following examples. Operating system settings, user settings, environment variables, and the .NET version you're using can all affect the format. For example, starting with .NET 5, .NET tries to unify cultural formats across platforms. For more information, see [.NET globalization and ICU](#).

FORMAT SPECIFIER	NAME	DESCRIPTION	EXAMPLES
"C" or "c"	Currency	<p>Result: A currency value.</p> <p>Supported by: All numeric types.</p> <p>Precision specifier: Number of decimal digits.</p> <p>Default precision specifier: Defined by NumberFormatInfo.CurrencyDecimalDigits.</p> <p>More information: The Currency ("C") Format Specifier.</p>	<p>123.456 ("C", en-US) -> \$123.46</p> <p>123.456 ("C", fr-FR) -> 123,46 €</p> <p>123.456 ("C", ja-JP) -> ¥123</p> <p>-123.456 ("C3", en-US) -> (\$123.456)</p> <p>-123.456 ("C3", fr-FR) -> -123,456 €</p> <p>-123.456 ("C3", ja-JP) -> -¥123.456</p>
"D" or "d"	Decimal	<p>Result: Integer digits with optional negative sign.</p> <p>Supported by: Integral types only.</p> <p>Precision specifier: Minimum number of digits.</p> <p>Default precision specifier: Minimum number of digits required.</p> <p>More information: The Decimal("D") Format Specifier.</p>	<p>1234 ("D") -> 1234</p> <p>-1234 ("D6") -> -001234</p>

Format Specifier	Name	Description	Examples
"E" or "e"	Exponential (scientific)	<p>Result: Exponential notation.</p> <p>Supported by: All numeric types.</p> <p>Precision specifier: Number of decimal digits.</p> <p>Default precision specifier: 6.</p> <p>More information: The Exponential ("E") Format Specifier.</p>	1052.0329112756 ("E", en-US) -> 1.052033E+003 1052.0329112756 ("e", fr-FR) -> 1,052033e+003 -1052.0329112756 ("e2", en-US) -> -1.05e+003 -1052.0329112756 ("E2", fr-FR) -> -1,05E+003
"F" or "f"	Fixed-point	<p>Result: Integral and decimal digits with optional negative sign.</p> <p>Supported by: All numeric types.</p> <p>Precision specifier: Number of decimal digits.</p> <p>Default precision specifier: Defined by NumberFormatInfo.NumberDecimalDigits.</p> <p>More information: The Fixed-Point ("F") Format Specifier.</p>	1234.567 ("F", en-US) -> 1234.57 1234.567 ("F", de-DE) -> 1234,57 1234 ("F1", en-US) -> 1234.0 1234 ("F1", de-DE) -> 1234,0 -1234.56 ("F4", en-US) -> -1234.5600 -1234.56 ("F4", de-DE) -> -1234,5600
"G" or "g"	General	<p>Result: The more compact of either fixed-point or scientific notation.</p> <p>Supported by: All numeric types.</p> <p>Precision specifier: Number of significant digits.</p> <p>Default precision specifier: Depends on numeric type.</p> <p>More information: The General ("G") Format Specifier.</p>	-123.456 ("G", en-US) -> -123.456 -123.456 ("G", sv-SE) -> -123,456 123.4546 ("G4", en-US) -> 123.5 123.4546 ("G4", sv-SE) -> 123,5 -1.234567890e-25 ("G", en-US) -> -1.23456789E-25 -1.234567890e-25 ("G", sv-SE) -> -1,23456789E-25

Format specifier	Name	Description	Examples
"N" or "n"	Number	<p>Result: Integral and decimal digits, group separators, and a decimal separator with optional negative sign.</p> <p>Supported by: All numeric types.</p> <p>Precision specifier: Desired number of decimal places.</p> <p>Default precision specifier: Defined by NumberFormatInfo.NumberDecimalDigits.</p> <p>More information: The Numeric ("N") Format Specifier.</p>	1234.567 ("N", en-US) -> 1,234.57 1234.567 ("N", ru-RU) -> 1 234,57 1234 ("N1", en-US) -> 1,234.0 1234 ("N1", ru-RU) -> 1 234,0 -1234.56 ("N3", en-US) -> -1,234.560 -1234.56 ("N3", ru-RU) -> -1 234,560
"P" or "p"	Percent	<p>Result: Number multiplied by 100 and displayed with a percent symbol.</p> <p>Supported by: All numeric types.</p> <p>Precision specifier: Desired number of decimal places.</p> <p>Default precision specifier: Defined by NumberFormatInfo.PercentDecimalDigits.</p> <p>More information: The Percent ("P") Format Specifier.</p>	1 ("P", en-US) -> 100.00 % 1 ("P", fr-FR) -> 100,00 % -0.39678 ("P1", en-US) -> -39.7 % -0.39678 ("P1", fr-FR) -> -39,7 %
"R" or "r"	Round-trip	<p>Result: A string that can round-trip to an identical number.</p> <p>Supported by: Single, Double, and BigInteger.</p> <p>Note: Recommended for the BigInteger type only. For Double types, use "G17"; for Single types, use "G9".</p> <p>Precision specifier: Ignored.</p> <p>More information: The Round-trip ("R") Format Specifier.</p>	123456789.12345678 ("R") -> 123456789.12345678 -1234567890.12345678 ("R") -> -1234567890.12345678

Format specifier	Name	Description	Examples
"X" or "x"	Hexadecimal	<p>Result: A hexadecimal string.</p> <p>Supported by: Integral types only.</p> <p>Precision specifier: Number of digits in the result string.</p> <p>More information: The HexaDecimal ("X") Format Specifier.</p>	255 ("X") -> FF -1 ("x") -> ff 255 ("x4") -> 00ff -1 ("X4") -> 00FF
Any other single character	Unknown specifier	Result: Throws a FormatException at run time.	

Use standard numeric format strings

NOTE

The C# examples in this article run in the [Try.NET](#) inline code runner and playground. Select the **Run** button to run an example in an interactive window. Once you execute the code, you can modify it and run the modified code by selecting **Run** again. The modified code either runs in the interactive window or, if compilation fails, the interactive window displays all C# compiler error messages.

A standard numeric format string can be used to define the formatting of a numeric value in one of the following ways:

- It can be passed to the `TryFormat` method or an overload of the `ToString` method that has a `format` parameter. The following example formats a numeric value as a currency string in the current culture (in this case, the en-US culture).

```
Decimal value = static_cast<Decimal>(123.456);
Console::WriteLine(value.ToString("C2"));
// Displays $123.46
```

```
decimal value = 123.456m;
Console.WriteLine(value.ToString("C2"));
// Displays $123.46
```

```
Dim value As Decimal = 123.456d
Console.WriteLine(value.ToString("C2"))
' Displays $123.46
```

- It can be supplied as the `formatString` argument in a format item used with such methods as `String.Format`, `Console.WriteLine`, and `StringBuilder.AppendFormat`. For more information, see [Composite Formatting](#). The following example uses a format item to insert a currency value in a string.

```
Decimal value = static_cast<Decimal>(123.456);
Console::WriteLine("Your account balance is {0:C2}.", value);
// Displays "Your account balance is $123.46."
```

```
decimal value = 123.456m;
Console.WriteLine("Your account balance is {0:C2}.", value);
// Displays "Your account balance is $123.46."
```

```
Dim value As Decimal = 123.456d
Console.WriteLine("Your account balance is {0:C2}.", value)
' Displays "Your account balance is $123.46."
```

Optionally, you can supply an `alignment` argument to specify the width of the numeric field and whether its value is right- or left-aligned. The following example left-aligns a currency value in a 28-character field, and it right-aligns a currency value in a 14-character field.

```
array<Decimal>^ amounts = { static_cast<Decimal>(16305.32),
                            static_cast<Decimal>(18794.16) };
Console::WriteLine("    Beginning Balance          Ending Balance");
Console::WriteLine("    {0,-28:C2}{1,14:C2}", amounts[0], amounts[1]);
// Displays:
//      Beginning Balance          Ending Balance
//      $16,305.32                $18,794.16
```

```
decimal[] amounts = { 16305.32m, 18794.16m };
Console.WriteLine("    Beginning Balance          Ending Balance");
Console.WriteLine("    {0,-28:C2}{1,14:C2}", amounts[0], amounts[1]);
// Displays:
//      Beginning Balance          Ending Balance
//      $16,305.32                $18,794.16
```

```
Dim amounts() As Decimal = {16305.32d, 18794.16d}
Console.WriteLine("    Beginning Balance          Ending Balance")
Console.WriteLine("    {0,-28:C2}{1,14:C2}", amounts(0), amounts(1))
' Displays:
'      Beginning Balance          Ending Balance
'      $16,305.32                $18,794.16
```

- It can be supplied as the `formatString` argument in an interpolated expression item of an interpolated string. For more information, see the [String interpolation](#) article in the C# reference or the [Interpolated strings](#) article in the Visual Basic reference.

The following sections provide detailed information about each of the standard numeric format strings.

Currency format specifier (C)

The "C" (or currency) format specifier converts a number to a string that represents a currency amount. The precision specifier indicates the desired number of decimal places in the result string. If the precision specifier is omitted, the default precision is defined by the [NumberFormatInfo.CurrencyDecimalDigits](#) property.

If the value to be formatted has more than the specified or default number of decimal places, the fractional value is rounded in the result string. If the value to the right of the number of specified decimal places is 5 or greater, the last digit in the result string is rounded away from zero.

The result string is affected by the formatting information of the current [NumberFormatInfo](#) object. The following table lists the [NumberFormatInfo](#) properties that control the formatting of the returned string.

NUMBERFORMATINFO PROPERTY	DESCRIPTION
CurrencyPositivePattern	Defines the placement of the currency symbol for positive values.
CurrencyNegativePattern	Defines the placement of the currency symbol for negative values, and specifies whether the negative sign is represented by parentheses or the NegativeSign property.
NegativeSign	Defines the negative sign used if CurrencyNegativePattern indicates that parentheses are not used.
CurrencySymbol	Defines the currency symbol.
CurrencyDecimalDigits	Defines the default number of decimal digits in a currency value. This value can be overridden by using the precision specifier.
CurrencyDecimalSeparator	Defines the string that separates integral and decimal digits.
CurrencyGroupSeparator	Defines the string that separates groups of integral numbers.
CurrencyGroupSizes	Defines the number of integer digits that appear in a group.

The following example formats a [Double](#) value with the currency format specifier:

```
double value = 12345.6789;
Console::WriteLine(value.ToString("C", CultureInfo::GetCurrentCulture()));

Console::WriteLine(value.ToString("C3", CultureInfo::GetCurrentCulture()));

Console::WriteLine(value.ToString("C3",
    CultureInfo::CreateSpecificCulture("da-DK")));
// The example displays the following output on a system whose
// current culture is English (United States):
//      $12,345.68
//      $12,345.679
//      kr 12.345,679
```

```
double value = 12345.6789;
Console.WriteLine(value.ToString("C", CultureInfo.CurrentCulture));

Console.WriteLine(value.ToString("C3", CultureInfo.CurrentCulture));

Console.WriteLine(value.ToString("C3",
    CultureInfo.CreateSpecificCulture("da-DK")));
// The example displays the following output on a system whose
// current culture is English (United States):
//      $12,345.68
//      $12,345.679
//      12.345,679 kr
```

```

Dim value As Double = 12345.6789
Console.WriteLine(value.ToString("C", CultureInfo.CurrentCulture))

Console.WriteLine(value.ToString("C3", CultureInfo.CurrentCulture))

Console.WriteLine(value.ToString("C3", _
    CultureInfo.CreateSpecificCulture("da-DK")))
' The example displays the following output on a system whose
' current culture is English (United States):
'      $12,345.68
'      $12,345.679
'      kr 12.345,679

```

Decimal format specifier (D)

The "D" (or decimal) format specifier converts a number to a string of decimal digits (0-9), prefixed by a minus sign if the number is negative. This format is supported only for integral types.

The precision specifier indicates the minimum number of digits desired in the resulting string. If required, the number is padded with zeros to its left to produce the number of digits given by the precision specifier. If no precision specifier is specified, the default is the minimum value required to represent the integer without leading zeros.

The result string is affected by the formatting information of the current [NumberFormatInfo](#) object. As the following table shows, a single property affects the formatting of the result string.

NUMBERFORMATINFO PROPERTY	DESCRIPTION
NegativeSign	Defines the string that indicates that a number is negative.

The following example formats an [Int32](#) value with the decimal format specifier.

```

int value;

value = 12345;
Console::WriteLine(value.ToString("D"));
// Displays 12345
Console::WriteLine(value.ToString("D8"));
// Displays 00012345

value = -12345;
Console::WriteLine(value.ToString("D"));
// Displays -12345
Console::WriteLine(value.ToString("D8"));
// Displays -00012345

```

```

int value;

value = 12345;
Console.WriteLine(value.ToString("D"));
// Displays 12345
Console.WriteLine(value.ToString("D8"));
// Displays 00012345

value = -12345;
Console.WriteLine(value.ToString("D"));
// Displays -12345
Console.WriteLine(value.ToString("D8"));
// Displays -00012345

```

```

Dim value As Integer

value = 12345
Console.WriteLine(value.ToString("D"))
' Displays 12345
Console.WriteLine(value.ToString("D8"))
' Displays 00012345

value = -12345
Console.WriteLine(value.ToString("D"))
' Displays -12345
Console.WriteLine(value.ToString("D8"))
' Displays -00012345

```

Exponential format specifier (E)

The exponential ("E") format specifier converts a number to a string of the form "-d.ddd...E+ddd" or "-d.ddd...e+ddd", where each "d" indicates a digit (0-9). The string starts with a minus sign if the number is negative. Exactly one digit always precedes the decimal point.

The precision specifier indicates the desired number of digits after the decimal point. If the precision specifier is omitted, a default of six digits after the decimal point is used.

The case of the format specifier indicates whether to prefix the exponent with an "E" or an "e". The exponent always consists of a plus or minus sign and a minimum of three digits. The exponent is padded with zeros to meet this minimum, if required.

The result string is affected by the formatting information of the current [NumberFormatInfo](#) object. The following table lists the [NumberFormatInfo](#) properties that control the formatting of the returned string.

NUMBERFORMATINFO PROPERTY	DESCRIPTION
NegativeSign	Defines the string that indicates that a number is negative for both the coefficient and exponent.
NumberDecimalSeparator	Defines the string that separates the integral digit from decimal digits in the coefficient.
PositiveSign	Defines the string that indicates that an exponent is positive.

The following example formats a [Double](#) value with the exponential format specifier:

```

double value = 12345.6789;
Console::WriteLine(value.ToString("E", CultureInfo::InvariantCulture));
// Displays 1.234568E+004

Console::WriteLine(value.ToString("E10", CultureInfo::InvariantCulture));
// Displays 1.2345678900E+004

Console::WriteLine(value.ToString("e4", CultureInfo::InvariantCulture));
// Displays 1.2346e+004

Console::WriteLine(value.ToString("E",
    CultureInfo::CreateSpecificCulture("fr-FR")));
// Displays 1,234568E+004

```

```

double value = 12345.6789;
Console.WriteLine(value.ToString("E", CultureInfo.InvariantCulture));
// Displays 1.234568E+004

Console.WriteLine(value.ToString("E10", CultureInfo.InvariantCulture));
// Displays 1.2345678900E+004

Console.WriteLine(value.ToString("e4", CultureInfo.InvariantCulture));
// Displays 1.2346e+004

Console.WriteLine(value.ToString("E",
    CultureInfo.CreateSpecificCulture("fr-FR")));
// Displays 1,234568E+004

```

```

Dim value As Double = 12345.6789
Console.WriteLine(value.ToString("E", CultureInfo.InvariantCulture))
' Displays 1.234568E+004

Console.WriteLine(value.ToString("E10", CultureInfo.InvariantCulture))
' Displays 1.2345678900E+004

Console.WriteLine(value.ToString("e4", CultureInfo.InvariantCulture))
' Displays 1.2346e+004

Console.WriteLine(value.ToString("E", _
    CultureInfo.CreateSpecificCulture("fr-FR")))
' Displays 1,234568E+004

```

Fixed-point format specifier (F)

The fixed-point ("F") format specifier converts a number to a string of the form "-ddd.ddd..." where each "d" indicates a digit (0-9). The string starts with a minus sign if the number is negative.

The precision specifier indicates the desired number of decimal places. If the precision specifier is omitted, the current [NumberFormatInfo.NumberDecimalDigits](#) property supplies the numeric precision.

The result string is affected by the formatting information of the current [NumberFormatInfo](#) object. The following table lists the properties of the [NumberFormatInfo](#) object that control the formatting of the result string.

NUMBERFORMATINFO PROPERTY	DESCRIPTION
NegativeSign	Defines the string that indicates that a number is negative.

NUMBERFORMATINFO PROPERTY	DESCRIPTION
NumberDecimalSeparator	Defines the string that separates integral digits from decimal digits.
NumberDecimalDigits	Defines the default number of decimal digits. This value can be overridden by using the precision specifier.

The following example formats a [Double](#) and an [Int32](#) value with the fixed-point format specifier:

```

int integerNumber;
integerNumber = 17843;
Console::WriteLine(integerNumber.ToString("F",
    CultureInfo::InvariantCulture));
// Displays 17843.00

integerNumber = -29541;
Console::WriteLine(integerNumber.ToString("F3",
    CultureInfo::InvariantCulture));
// Displays -29541.000

double doubleNumber;
doubleNumber = 18934.1879;
Console::WriteLine(doubleNumber.ToString("F", CultureInfo::InvariantCulture));
// Displays 18934.19

Console::WriteLine(doubleNumber.ToString("F0", CultureInfo::InvariantCulture));
// Displays 18934

doubleNumber = -1898300.1987;
Console::WriteLine(doubleNumber.ToString("F1", CultureInfo::InvariantCulture));
// Displays -1898300.2

Console::WriteLine(doubleNumber.ToString("F3",
    CultureInfo::CreateSpecificCulture("es-ES")));
// Displays -1898300,199

```

```

int integerNumber;
integerNumber = 17843;
Console.WriteLine(integerNumber.ToString("F",
    CultureInfo.InvariantCulture));
// Displays 17843.00

integerNumber = -29541;
Console.WriteLine(integerNumber.ToString("F3",
    CultureInfo.InvariantCulture));
// Displays -29541.000

double doubleNumber;
doubleNumber = 18934.1879;
Console.WriteLine(doubleNumber.ToString("F", CultureInfo.InvariantCulture));
// Displays 18934.19

Console.WriteLine(doubleNumber.ToString("F0", CultureInfo.InvariantCulture));
// Displays 18934

doubleNumber = -1898300.1987;
Console.WriteLine(doubleNumber.ToString("F1", CultureInfo.InvariantCulture));
// Displays -1898300.2

Console.WriteLine(doubleNumber.ToString("F3",
    CultureInfo.CreateSpecificCulture("es-ES")));
// Displays -1898300,199

```

```

Dim integerNumber As Integer
integerNumber = 17843
Console.WriteLine(integerNumber.ToString("F", CultureInfo.InvariantCulture))
' Displays 17843.00

integerNumber = -29541
Console.WriteLine(integerNumber.ToString("F3", CultureInfo.InvariantCulture))
' Displays -29541.000

Dim doubleNumber As Double
doubleNumber = 18934.1879
Console.WriteLine(doubleNumber.ToString("F", CultureInfo.InvariantCulture))
' Displays 18934.19

Console.WriteLine(doubleNumber.ToString("F0", CultureInfo.InvariantCulture))
' Displays 18934

doubleNumber = -1898300.1987
Console.WriteLine(doubleNumber.ToString("F1", CultureInfo.InvariantCulture))
' Displays -1898300.2

Console.WriteLine(doubleNumber.ToString("F3", _
    CultureInfo.CreateSpecificCulture("es-ES")))
' Displays -1898300,199

```

General format specifier (G)

The general ("G") format specifier converts a number to the more compact of either fixed-point or scientific notation, depending on the type of the number and whether a precision specifier is present. The precision specifier defines the maximum number of significant digits that can appear in the result string. If the precision specifier is omitted or zero, the type of the number determines the default precision, as indicated in the following table.

NUMERIC TYPE	DEFAULT PRECISION
Byte or SByte	3 digits
Int16 or UInt16	5 digits
Int32 or UInt32	10 digits
Int64	19 digits
UInt64	20 digits
BigInteger	Unlimited (same as "R")
Half	3 digits
Single	7 digits
Double	15 digits
Decimal	29 digits

Fixed-point notation is used if the exponent that would result from expressing the number in scientific notation is greater than -5 and less than the precision specifier; otherwise, scientific notation is used. The result contains a decimal point if required, and trailing zeros after the decimal point are omitted. If the precision specifier is present and the number of significant digits in the result exceeds the specified precision, the excess trailing digits are removed by rounding.

However, if the number is a [Decimal](#) and the precision specifier is omitted, fixed-point notation is always used and trailing zeros are preserved.

If scientific notation is used, the exponent in the result is prefixed with "E" if the format specifier is "G", or "e" if the format specifier is "g". The exponent contains a minimum of two digits. This differs from the format for scientific notation that is produced by the exponential format specifier, which includes a minimum of three digits in the exponent.

When used with a [Double](#) value, the "G17" format specifier ensures that the original [Double](#) value successfully round-trips. This is because [Double](#) is an IEEE 754-2008-compliant double-precision (`binary64`) floating-point number that gives up to 17 significant digits of precision. On .NET Framework, we recommend its use instead of the ["R" format specifier](#), since in some cases "R" fails to successfully round-trip double-precision floating point values.

When used with a [Single](#) value, the "G9" format specifier ensures that the original [Single](#) value successfully round-trips. This is because [Single](#) is an IEEE 754-2008-compliant single-precision (`binary32`) floating-point number that gives up to nine significant digits of precision. For performance reasons, we recommend its use instead of the ["R" format specifier](#).

The result string is affected by the formatting information of the current [NumberFormatInfo](#) object. The following table lists the [NumberFormatInfo](#) properties that control the formatting of the result string.

NUMBERFORMATINFO PROPERTY	DESCRIPTION
NegativeSign	Defines the string that indicates that a number is negative.

NUMBERFORMATINFO PROPERTY	DESCRIPTION
NumberDecimalSeparator	Defines the string that separates integral digits from decimal digits.
PositiveSign	Defines the string that indicates that an exponent is positive.

The following example formats assorted floating-point values with the general format specifier:

```
double number;

number = 12345.6789;
Console::WriteLine(number.ToString("G", CultureInfo::InvariantCulture));
// Displays 12345.6789
Console::WriteLine(number.ToString("G",
    CultureInfo::CreateSpecificCulture("fr-FR")));
// Displays 12345,6789

Console::WriteLine(number.ToString("G7", CultureInfo::InvariantCulture));
// Displays 12345.68

number = .0000023;
Console::WriteLine(number.ToString("G", CultureInfo::InvariantCulture));
// Displays 2.3E-06
Console::WriteLine(number.ToString("G",
    CultureInfo::CreateSpecificCulture("fr-FR")));
// Displays 2,3E-06

number = .0023;
Console::WriteLine(number.ToString("G", CultureInfo::InvariantCulture));
// Displays 0.0023

number = 1234;
Console::WriteLine(number.ToString("G2", CultureInfo::InvariantCulture));
// Displays 1.2E+03

number = Math::PI;
Console::WriteLine(number.ToString("G5", CultureInfo::InvariantCulture));
// Displays 3.1416
```

```

double number;

number = 12345.6789;
Console.WriteLine(number.ToString("G", CultureInfo.InvariantCulture));
// Displays 12345.6789
Console.WriteLine(number.ToString("G",
    CultureInfo.CreateSpecificCulture("fr-FR")));
// Displays 12345,6789

Console.WriteLine(number.ToString("G7", CultureInfo.InvariantCulture));
// Displays 12345.68

number = .0000023;
Console.WriteLine(number.ToString("G", CultureInfo.InvariantCulture));
// Displays 2.3E-06
Console.WriteLine(number.ToString("G",
    CultureInfo.CreateSpecificCulture("fr-FR")));
// Displays 2,3E-06

number = .0023;
Console.WriteLine(number.ToString("G", CultureInfo.InvariantCulture));
// Displays 0.0023

number = 1234;
Console.WriteLine(number.ToString("G2", CultureInfo.InvariantCulture));
// Displays 1.2E+03

number = Math.PI;
Console.WriteLine(number.ToString("G5", CultureInfo.InvariantCulture));
// Displays 3.1416

```

```

Dim number As Double

number = 12345.6789
Console.WriteLine(number.ToString("G", CultureInfo.InvariantCulture))
' Displays 12345.6789
Console.WriteLine(number.ToString("G",
    CultureInfo.CreateSpecificCulture("fr-FR")))
' Displays 12345,6789

Console.WriteLine(number.ToString("G7", CultureInfo.InvariantCulture))
' Displays 12345.68

number = .0000023
Console.WriteLine(number.ToString("G", CultureInfo.InvariantCulture))
' Displays 2.3E-06
Console.WriteLine(number.ToString("G",
    CultureInfo.CreateSpecificCulture("fr-FR")))
' Displays 2,3E-06

number = .0023
Console.WriteLine(number.ToString("G", CultureInfo.InvariantCulture))
' Displays 0.0023

number = 1234
Console.WriteLine(number.ToString("G2", CultureInfo.InvariantCulture))
' Displays 1.2E+03

number = Math.Pi
Console.WriteLine(number.ToString("G5", CultureInfo.InvariantCulture))
' Displays 3.1416

```

Numeric format specifier (N)

The numeric ("N") format specifier converts a number to a string of the form "-d,ddd,ddd.ddd...", where "-" indicates a negative number symbol if required, "d" indicates a digit (0-9), "," indicates a group separator, and "." indicates a decimal point symbol. The precision specifier indicates the desired number of digits after the decimal point. If the precision specifier is omitted, the number of decimal places is defined by the current [NumberFormatInfo.NumberDecimalDigits](#) property.

The result string is affected by the formatting information of the current [NumberFormatInfo](#) object. The following table lists the [NumberFormatInfo](#) properties that control the formatting of the result string.

NUMBERFORMATINFO PROPERTY	DESCRIPTION
NegativeSign	Defines the string that indicates that a number is negative.
NumberNegativePattern	Defines the format of negative values, and specifies whether the negative sign is represented by parentheses or the NegativeSign property.
NumberGroupSizes	Defines the number of integral digits that appear between group separators.
NumberGroupSeparator	Defines the string that separates groups of integral numbers.
NumberDecimalSeparator	Defines the string that separates integral and decimal digits.
NumberDecimalDigits	Defines the default number of decimal digits. This value can be overridden by using a precision specifier.

The following example formats assorted floating-point values with the number format specifier:

```
double dblValue = -12445.6789;
Console::WriteLine(dblValue.ToString("N", CultureInfo::InvariantCulture));
// Displays -12,445.68
Console::WriteLine(dblValue.ToString("N1",
    CultureInfo::CreateSpecificCulture("sv-SE")));
// Displays -12 445,7

int intValue = 123456789;
Console::WriteLine(intValue.ToString("N1", CultureInfo::InvariantCulture));
// Displays 123,456,789.0
```

```
double dblValue = -12445.6789;
Console.WriteLine(dblValue.ToString("N", CultureInfo.InvariantCulture));
// Displays -12,445.68
Console.WriteLine(dblValue.ToString("N1",
    CultureInfo.CreateSpecificCulture("sv-SE")));
// Displays -12 445,7

int intValue = 123456789;
Console.WriteLine(intValue.ToString("N1", CultureInfo.InvariantCulture));
// Displays 123,456,789.0
```

```

Dim dblValue As Double = -12445.6789
Console.WriteLine(dblValue.ToString("N", CultureInfo.InvariantCulture))
' Displays -12,445.68
Console.WriteLine(dblValue.ToString("N1",
    CultureInfo.CreateSpecificCulture("sv-SE")))
' Displays -12 445,7

Dim intValue As Integer = 123456789
Console.WriteLine(intValue.ToString("N1", CultureInfo.InvariantCulture))
' Displays 123,456,789.0

```

Percent format specifier (P)

The percent ("P") format specifier multiplies a number by 100 and converts it to a string that represents a percentage. The precision specifier indicates the desired number of decimal places. If the precision specifier is omitted, the default numeric precision supplied by the current [PercentDecimalDigits](#) property is used.

The following table lists the [NumberFormatInfo](#) properties that control the formatting of the returned string.

NUMBERFORMATINFO PROPERTY	DESCRIPTION
PercentPositivePattern	Defines the placement of the percent symbol for positive values.
PercentNegativePattern	Defines the placement of the percent symbol and the negative symbol for negative values.
NegativeSign	Defines the string that indicates that a number is negative.
PercentSymbol	Defines the percent symbol.
PercentDecimalDigits	Defines the default number of decimal digits in a percentage value. This value can be overridden by using the precision specifier.
PercentDecimalSeparator	Defines the string that separates integral and decimal digits.
PercentGroupSeparator	Defines the string that separates groups of integral numbers.
PercentGroupSizes	Defines the number of integer digits that appear in a group.

The following example formats floating-point values with the percent format specifier:

```

double number = .2468013;
Console::WriteLine(number.ToString("P", CultureInfo::InvariantCulture));
// Displays 24.68 %
Console::WriteLine(number.ToString("P",
    CultureInfo::CreateSpecificCulture("hr-HR")));
// Displays 24,68%
Console::WriteLine(number.ToString("P1", CultureInfo::InvariantCulture));
// Displays 24.7 %

```

```

double number = .2468013;
Console.WriteLine(number.ToString("P", CultureInfo.InvariantCulture));
// Displays 24.68 %
Console.WriteLine(number.ToString("P",
    CultureInfo.CreateSpecificCulture("hr-HR")));
// Displays 24,68%
Console.WriteLine(number.ToString("P1", CultureInfo.InvariantCulture));
// Displays 24.7 %

```

```

Dim number As Double = .2468013
Console.WriteLine(number.ToString("P", CultureInfo.InvariantCulture))
' Displays 24.68 %
Console.WriteLine(number.ToString("P",
    CultureInfo.CreateSpecificCulture("hr-HR")))
' Displays 24,68%
Console.WriteLine(number.ToString("P1", CultureInfo.InvariantCulture))
' Displays 24.7 %

```

Round-trip format specifier (R)

The round-trip ("R") format specifier attempts to ensure that a numeric value that is converted to a string is parsed back into the same numeric value. This format is supported only for the [Half](#), [Single](#), [Double](#), and [BigInteger](#) types.

In .NET Framework and in .NET Core versions earlier than 3.0, the "R" format specifier fails to successfully round-trip [Double](#) values in some cases. For both [Double](#) and [Single](#) values, the "R" format specifier offers relatively poor performance. Instead, we recommend that you use the "[G17](#)" format specifier for [Double](#) values and the "[G9](#)" format specifier to successfully round-trip [Single](#) values.

When a [BigInteger](#) value is formatted using this specifier, its string representation contains all the significant digits in the [BigInteger](#) value.

Although you can include a precision specifier, it is ignored. Round trips are given precedence over precision when using this specifier. The result string is affected by the formatting information of the current [NumberFormatInfo](#) object. The following table lists the [NumberFormatInfo](#) properties that control the formatting of the result string.

NUMBERFORMATINFO PROPERTY	DESCRIPTION
NegativeSign	Defines the string that indicates that a number is negative.
NumberDecimalSeparator	Defines the string that separates integral digits from decimal digits.
PositiveSign	Defines the string that indicates that an exponent is positive.

The following example formats a [BigInteger](#) value with the round-trip format specifier.

```
#using <System.Numerics.dll>

using namespace System;
using namespace System::Numerics;

void main()
{
    BigInteger value = BigInteger::Pow(Int64::.MaxValue, 2);
    Console::WriteLine(value.ToString("R"));
}
// The example displays the following output:
//      85070591730234615847396907784232501249
```

```
using System;
using System.Numerics;

public class Example
{
    public static void Main()
    {
        var value = BigInteger.Pow(Int64.MaxValue, 2);
        Console.WriteLine(value.ToString("R"));
    }
}
// The example displays the following output:
//      85070591730234615847396907784232501249
```

```
Imports System.Numerics

Module Example
    Public Sub Main()
        Dim value = BigInteger.Pow(Int64.MaxValue, 2)
        Console.WriteLine(value.ToString("R"))
    End Sub
End Module
' The example displays the following output:
'      85070591730234615847396907784232501249
```

IMPORTANT

In some cases, **Double** values formatted with the "R" standard numeric format string do not successfully round-trip if compiled using the `/platform:x64` or `/platform:anycpu` switches and run on 64-bit systems. See the following paragraph for more information.

To work around the problem of **Double** values formatted with the "R" standard numeric format string not successfully round-tripping if compiled using the `/platform:x64` or `/platform:anycpu` switches and run on 64-bit systems., you can format **Double** values by using the "G17" standard numeric format string. The following example uses the "R" format string with a **Double** value that does not round-trip successfully, and also uses the "G17" format string to successfully round-trip the original value:

```

Console.WriteLine("Attempting to round-trip a Double with 'R':");
double initialValue = 0.6822871999174;
string valueString = initialValue.ToString("R",
                                         CultureInfo.InvariantCulture);
double roundTripped = double.Parse(valueString,
                                     CultureInfo.InvariantCulture);
Console.WriteLine("{0:R} = {1:R}: {2}\n",
                  initialValue, roundTripped, initialValue.Equals(roundTripped));

Console.WriteLine("Attempting to round-trip a Double with 'G17':");
string valueString17 = initialValue.ToString("G17",
                                             CultureInfo.InvariantCulture);
double roundTripped17 = double.Parse(valueString17,
                                      CultureInfo.InvariantCulture);
Console.WriteLine("{0:R} = {1:R}: {2}\n",
                  initialValue, roundTripped17, initialValue.Equals(roundTripped17));
// If compiled to an application that targets anycpu or x64 and run on an x64 system,
// the example displays the following output:
//     Attempting to round-trip a Double with 'R':
//     0.6822871999174 = 0.68228719991740006: False
//
//     Attempting to round-trip a Double with 'G17':
//     0.6822871999174 = 0.6822871999174: True

```

```

Imports System.Globalization

Module Example
    Public Sub Main()
        Console.WriteLine("Attempting to round-trip a Double with 'R':")
        Dim initialValue As Double = 0.6822871999174
        Dim valueString As String = initialValue.ToString("R",
                                                       CultureInfo.InvariantCulture)
        Dim roundTripped As Double = Double.Parse(valueString,
                                                   CultureInfo.InvariantCulture)
        Console.WriteLine("{0:R} = {1:R}: {2}",
                          initialValue, roundTripped, initialValue.Equals(roundTripped))
        Console.WriteLine()

        Console.WriteLine("Attempting to round-trip a Double with 'G17':")
        Dim valueString17 As String = initialValue.ToString("G17",
                                                          CultureInfo.InvariantCulture)
        Dim roundTripped17 As Double = double.Parse(valueString17,
                                                    CultureInfo.InvariantCulture)
        Console.WriteLine("{0:R} = {1:R}: {2}",
                          initialValue, roundTripped17, initialValue.Equals(roundTripped17))
    End Sub
End Module
' If compiled to an application that targets anycpu or x64 and run on an x64 system,
' the example displays the following output:
'     Attempting to round-trip a Double with 'R':
'     0.6822871999174 = 0.68228719991740006: False
'
'     Attempting to round-trip a Double with 'G17':
'     0.6822871999174 = 0.6822871999174: True

```

Hexadecimal format specifier (X)

The hexadecimal ("X") format specifier converts a number to a string of hexadecimal digits. The case of the format specifier indicates whether to use uppercase or lowercase characters for hexadecimal digits that are greater than 9. For example, use "X" to produce "ABCDEF", and "x" to produce "abcdef". This format is supported only for integral types.

The precision specifier indicates the minimum number of digits desired in the resulting string. If required, the number is padded with zeros to its left to produce the number of digits given by the precision specifier.

The result string is not affected by the formatting information of the current [NumberFormatInfo](#) object.

The following example formats [Int32](#) values with the hexadecimal format specifier.

```
int value;

value = 0x2045e;
Console::WriteLine(value.ToString("x"));
// Displays 2045e
Console::WriteLine(value.ToString("X"));
// Displays 2045E
Console::WriteLine(value.ToString("X8"));
// Displays 0002045E

value = 123456789;
Console::WriteLine(value.ToString("X"));
// Displays 75BCD15
Console::WriteLine(value.ToString("X2"));
// Displays 75BCD15
```

```
int value;

value = 0x2045e;
Console.WriteLine(value.ToString("x"));
// Displays 2045e
Console.WriteLine(value.ToString("X"));
// Displays 2045E
Console.WriteLine(value.ToString("X8"));
// Displays 0002045E

value = 123456789;
Console.WriteLine(value.ToString("X"));
// Displays 75BCD15
Console.WriteLine(value.ToString("X2"));
// Displays 75BCD15
```

```
Dim value As Integer

value = &h2045e
Console.WriteLine(value.ToString("x"))
' Displays 2045e
Console.WriteLine(value.ToString("X"))
' Displays 2045E
Console.WriteLine(value.ToString("X8"))
' Displays 0002045E

value = 123456789
Console.WriteLine(value.ToString("X"))
' Displays 75BCD15
Console.WriteLine(value.ToString("X2"))
' Displays 75BCD15
```

Notes

This section contains additional information about using standard numeric format strings.

Control Panel settings

The settings in the [Regional and Language Options](#) item in Control Panel influence the result string

produced by a formatting operation. Those settings are used to initialize the [NumberFormatInfo](#) object associated with the current culture, which provides values used to govern formatting. Computers that use different settings generate different result strings.

In addition, if the [CultureInfo\(String\)](#) constructor is used to instantiate a new [CultureInfo](#) object that represents the same culture as the current system culture, any customizations established by the **Regional and Language Options** item in Control Panel will be applied to the new [CultureInfo](#) object. You can use the [CultureInfo\(String, Boolean\)](#) constructor to create a [CultureInfo](#) object that does not reflect a system's customizations.

NumberFormatInfo properties

Formatting is influenced by the properties of the current [NumberFormatInfo](#) object, which is provided implicitly by the current culture or explicitly by the [IFormatProvider](#) parameter of the method that invokes formatting. Specify a [NumberFormatInfo](#) or [CultureInfo](#) object for that parameter.

NOTE

For information about customizing the patterns or strings used in formatting numeric values, see the [NumberFormatInfo](#) class topic.

Integral and floating-point numeric types

Some descriptions of standard numeric format specifiers refer to integral or floating-point numeric types. The integral numeric types are [Byte](#), [SByte](#), [Int16](#), [Int32](#), [Int64](#), [UInt16](#), [UInt32](#), [UInt64](#), and [BigInteger](#). The floating-point numeric types are [Decimal](#), [Half](#), [Single](#), and [Double](#).

Floating-point infinities and NaN

Regardless of the format string, if the value of a [Half](#), [Single](#), or [Double](#) floating-point type is positive infinity, negative infinity, or not a number (NaN), the formatted string is the value of the respective [PositiveInfinitySymbol](#), [NegativeInfinitySymbol](#), or [NaNSymbol](#) property that is specified by the currently applicable [NumberFormatInfo](#) object.

Code example

The following example formats an integral and a floating-point numeric value using the en-US culture and all the standard numeric format specifiers. This example uses two particular numeric types ([Double](#) and [Int32](#)), but would yield similar results for any of the other numeric base types ([Byte](#), [SByte](#), [Int16](#), [Int32](#), [Int64](#), [UInt16](#), [UInt32](#), [UInt64](#), [BigInteger](#), [Decimal](#), [Half](#), and [Single](#)).

```
// Display string representations of numbers for en-us culture
CultureInfo ci = new CultureInfo("en-us");

// Output floating point values
double floating = 10761.937554;
Console.WriteLine("C: {0}",
    floating.ToString("C", ci));           // Displays "C: $10,761.94"
Console.WriteLine("E: {0}",
    floating.ToString("E03", ci));         // Displays "E: 1.076E+004"
Console.WriteLine("F: {0}",
    floating.ToString("F04", ci));         // Displays "F: 10761.9376"
Console.WriteLine("G: {0}",
    floating.ToString("G", ci));          // Displays "G: 10761.937554"
Console.WriteLine("N: {0}",
    floating.ToString("N03", ci));         // Displays "N: 10,761.938"
Console.WriteLine("P: {0}",
    (floating/10000).ToString("P02", ci)); // Displays "P: 107.62 %"
Console.WriteLine("R: {0}",
    floating.ToString("R", ci));          // Displays "R: 10761.937554"
Console.WriteLine();

// Output integral values
int integral = 8395;
Console.WriteLine("C: {0}",
    integral.ToString("C", ci));           // Displays "C: $8,395.00"
Console.WriteLine("D: {0}",
    integral.ToString("D6", ci));          // Displays "D: 008395"
Console.WriteLine("E: {0}",
    integral.ToString("E03", ci));         // Displays "E: 8.395E+003"
Console.WriteLine("F: {0}",
    integral.ToString("F01", ci));          // Displays "F: 8395.0"
Console.WriteLine("G: {0}",
    integral.ToString("G", ci));           // Displays "G: 8395"
Console.WriteLine("N: {0}",
    integral.ToString("N01", ci));          // Displays "N: 8,395.0"
Console.WriteLine("P: {0}",
    (integral/10000.0).ToString("P02", ci)); // Displays "P: 83.95 %"
Console.WriteLine("X: 0x{0}",
    integral.ToString("X", ci));           // Displays "X: 0x20CB"
Console.WriteLine();
```

```

Option Strict On

Imports System.Globalization
Imports System.Threading

Module NumericFormats
    Public Sub Main()
        ' Display string representations of numbers for en-us culture
        Dim ci As New CultureInfo("en-us")

        ' Output floating point values
        Dim floating As Double = 10761.937554
        Console.WriteLine("C: {0}", _
            floating.ToString("C", ci))           ' Displays "C: $10,761.94"
        Console.WriteLine("E: {0}", _
            floating.ToString("E03", ci))         ' Displays "E: 1.076E+004"
        Console.WriteLine("F: {0}", _
            floating.ToString("F04", ci))         ' Displays "F: 10761.9376"
        Console.WriteLine("G: {0}", _
            floating.ToString("G", ci))          ' Displays "G: 10761.937554"
        Console.WriteLine("N: {0}", _
            floating.ToString("N03", ci))         ' Displays "N: 10,761.938"
        Console.WriteLine("P: {0}", _
            (floating / 10000).ToString("P02", ci)) ' Displays "P: 107.62 %"
        Console.WriteLine("R: {0}", _
            floating.ToString("R", ci))           ' Displays "R: 10761.937554"
        Console.WriteLine()

        ' Output integral values
        Dim integral As Integer = 8395
        Console.WriteLine("C: {0}", _
            integral.ToString("C", ci))           ' Displays "C: $8,395.00"
        Console.WriteLine("D: {0}", _
            integral.ToString("D6"))              ' Displays "D: 008395"
        Console.WriteLine("E: {0}", _
            integral.ToString("E03", ci))         ' Displays "E: 8.395E+003"
        Console.WriteLine("F: {0}", _
            integral.ToString("F01", ci))         ' Displays "F: 8395.0"
        Console.WriteLine("G: {0}", _
            integral.ToString("G", ci))          ' Displays "G: 8395"
        Console.WriteLine("N: {0}", _
            integral.ToString("N01", ci))         ' Displays "N: 8,395.0"
        Console.WriteLine("P: {0}", _
            (integral / 10000).ToString("P02", ci)) ' Displays "P: 83.95 %"
        Console.WriteLine("X: 0x{0}", _
            integral.ToString("X", ci))           ' Displays "X: 0x20CB"
        Console.WriteLine()

    End Sub
End Module

```

See also

- [NumberFormatInfo](#)
- [Custom Numeric Format Strings](#)
- [Formatting Types](#)
- [How to: Pad a Number with Leading Zeros](#)
- [Composite Formatting](#)
- [Sample: .NET Core WinForms Formatting Utility \(C#\)](#)
- [Sample: .NET Core WinForms Formatting Utility \(Visual Basic\)](#)

Custom numeric format strings

9/20/2022 • 22 minutes to read • [Edit Online](#)

You can create a custom numeric format string, which consists of one or more custom numeric specifiers, to define how to format numeric data. A custom numeric format string is any format string that is not a [standard numeric format string](#).

Custom numeric format strings are supported by some overloads of the `ToString` method of all numeric types. For example, you can supply a numeric format string to the `ToString(String)` and `ToString(String, IFormatProvider)` methods of the `Int32` type. Custom numeric format strings are also supported by the .NET [composite formatting feature](#), which is used by some `Write` and `WriteLine` methods of the `Console` and `StreamWriter` classes, the `String.Format` method, and the `StringBuilder.AppendFormat` method. [String interpolation](#) feature also supports custom numeric format strings.

TIP

You can download the [Formatting Utility](#), a .NET Core Windows Forms application that lets you apply format strings to either numeric or date and time values and displays the result string. Source code is available for [C#](#) and [Visual Basic](#).

The following table describes the custom numeric format specifiers and displays sample output produced by each format specifier. See the [Notes](#) section for additional information about using custom numeric format strings, and the [Example](#) section for a comprehensive illustration of their use.

FORMAT SPECIFIER	NAME	DESCRIPTION	EXAMPLES
"0"	Zero placeholder	Replaces the zero with the corresponding digit if one is present; otherwise, zero appears in the result string. More information: The "0" Custom Specifier .	1234.5678 ("00000") -> 01235 0.45678 ("0.00", en-US) -> .46 0.45678 ("0.00", fr-FR) -> ,46
"#"	Digit placeholder	Replaces the "#" symbol with the corresponding digit if one is present; otherwise, no digit appears in the result string. Note that no digit appears in the result string if the corresponding digit in the input string is a non-significant 0. For example, 0003 ("####") -> 3. More information: The "#" Custom Specifier .	1234.5678 ("#####") -> 1235 0.45678 ("#.##", en-US) -> ,46 0.45678 ("#.##", fr-FR) -> ,46

FORMAT SPECIFIER	NAME	DESCRIPTION	EXAMPLES
".."	Decimal point	Determines the location of the decimal separator in the result string. More information: The ".," Custom Specifier .	0.45678 ("0.00", en-US) -> 0.46 0.45678 ("0.00", fr-FR) -> 0,46
" , "	Group separator and number scaling	Serves as both a group separator and a number scaling specifier. As a group separator, it inserts a localized group separator character between each group. As a number scaling specifier, it divides a number by 1000 for each comma specified. More information: The ",," Custom Specifier .	Group separator specifier: 2147483647 ("##,#", en-US) -> 2,147,483,647 2147483647 ("##,#", es-ES) -> 2.147.483.647 Scaling specifier: 2147483647 ("#,##,,", en-US) -> 2,147 2147483647 ("#,##,,", es-ES) -> 2.147
"%"	Percentage placeholder	Multiplies a number by 100 and inserts a localized percentage symbol in the result string. More information: The "%" Custom Specifier .	0.3697 ("###0.00", en-US) -> %36.97 0.3697 ("###0.00", el-GR) -> %36,97 0.3697 ("##.0 %", en-US) -> 37.0 % 0.3697 ("##.0 %", el-GR) -> 37,0 %
"%o"	Per mille placeholder	Multiplies a number by 1000 and inserts a localized per mille symbol in the result string. More information: The "%o" Custom Specifier .	0.03697 ("#0.00%o", en-US) -> 36.97%o 0.03697 ("#0.00%o", ru-RU) -> 36,97%o

Format Specifier	Name	Description	Examples
"E0" "E+0" "E-0" "e0" "e+0" "e-0"	Exponential notation	If followed by at least one 0 (zero), formats the result using exponential notation. The case of "E" or "e" indicates the case of the exponent symbol in the result string. The number of zeros following the "E" or "e" character determines the minimum number of digits in the exponent. A plus sign (+) indicates that a sign character always precedes the exponent. A minus sign (-) indicates that a sign character precedes only negative exponents. More information: The "E" and "e" Custom Specifiers .	987654 ("#0.0e0") -> 98.8e4 1503.92311 ("0.0##e+00") -> 1.504e+03 1.8901385E-16 ("0.0e+00") -> 1.9e-16
"\\"	Escape character	Causes the next character to be interpreted as a literal rather than as a custom format specifier. More information: The "\" Escape Character .	987654 ("###00\#") -> #987654#
'string' "string"	Literal string delimiter	Indicates that the enclosed characters should be copied to the result string unchanged. More information: Character literals .	68 ("# 'degrees'") -> 68 degrees 68 ("#" degrees "") -> 68 degrees
;	Section separator	Defines sections with separate format strings for positive, negative, and zero numbers. More information: The ":" Section Separator .	12.345 ("#0.0#;(##0.0#);-\0-") -> 12.35 0 ("#0.0#;(##0.0#);-\0-") -> -0- -12.345 ("#0.0#;(##0.0#);-\0-") -> (12.35) 12.345 ("#0.0#;(##0.0#)") -> 12.35 0 ("#0.0#;(##0.0#)") -> 0.0 -12.345 ("#0.0#;(##0.0#)") -> (12.35)
Other	All other characters	The character is copied to the result string unchanged. More information: Character literals .	68 ("# °") -> 68 °

The following sections provide detailed information about each of the custom numeric format specifiers.

NOTE

Some of the C# examples in this article run in the [Try.NET](#) inline code runner and playground. Select the **Run** button to run an example in an interactive window. Once you execute the code, you can modify it and run the modified code by selecting **Run** again. The modified code either runs in the interactive window or, if compilation fails, the interactive window displays all C# compiler error messages.

The "0" custom specifier

The "0" custom format specifier serves as a zero-placeholder symbol. If the value that is being formatted has a digit in the position where the zero appears in the format string, that digit is copied to the result string; otherwise, a zero appears in the result string. The position of the leftmost zero before the decimal point and the rightmost zero after the decimal point determines the range of digits that are always present in the result string.

The "00" specifier causes the value to be rounded to the nearest digit preceding the decimal, where rounding away from zero is always used. For example, formatting 34.5 with "00" would result in the value 35.

The following example displays several values that are formatted by using custom format strings that include zero placeholders.

```
double value;

value = 123;
Console::WriteLine(value.ToString("00000"));
Console::WriteLine(String::Format("{0:00000}", value));
// Displays 00123

value = 1.2;
Console::WriteLine(value.ToString("0.00", CultureInfo::InvariantCulture));
Console::WriteLine(String::Format(CultureInfo::InvariantCulture,
    "{0:0.00}", value));
// Displays 1.20

Console::WriteLine(value.ToString("00.00", CultureInfo::InvariantCulture));
Console::WriteLine(String::Format(CultureInfo::InvariantCulture,
    "{0:00.00}", value));
// Displays 01.20

CultureInfo^ daDK = CultureInfo::CreateSpecificCulture("da-DK");
Console::WriteLine(value.ToString("00.00", daDK));
Console::WriteLine(String::Format(daDK, "{0:00.00}", value));
// Displays 01,20

value = .56;
Console::WriteLine(value.ToString("0.0", CultureInfo::InvariantCulture));
Console::WriteLine(String::Format(CultureInfo::InvariantCulture,
    "{0:0.0}", value));
// Displays 0.6

value = 1234567890;
Console::WriteLine(value.ToString("0,0", CultureInfo::InvariantCulture));
Console::WriteLine(String::Format(CultureInfo::InvariantCulture,
    "{0:0,0}", value));
// Displays 1,234,567,890

CultureInfo^ elGR = CultureInfo::CreateSpecificCulture("el-GR");
Console::WriteLine(value.ToString("0,0", elGR));
Console::WriteLine(String::Format(elGR, "{0:0,0}", value));
// Displays 1.234.567.890

value = 1234567890.123456;
Console::WriteLine(value.ToString("0,0.0", CultureInfo::InvariantCulture));
Console::WriteLine(String::Format(CultureInfo::InvariantCulture,
    "{0:0,0.0}", value));
// Displays 1,234,567,890.1

value = 1234.567890;
Console::WriteLine(value.ToString("0,0.00", CultureInfo::InvariantCulture));
Console::WriteLine(String::Format(CultureInfo::InvariantCulture,
    "{0:0,0.00}", value));
// Displays 1,234.57
```

```
double value;

value = 123;
Console.WriteLine(value.ToString("00000"));
Console.WriteLine(String.Format("{0:00000}", value));
// Displays 00123

value = 1.2;
Console.WriteLine(value.ToString("0.00", CultureInfo.InvariantCulture));
Console.WriteLine(String.Format(CultureInfo.InvariantCulture,
    "{0:0.00}", value));
// Displays 1.20

Console.WriteLine(value.ToString("00.00", CultureInfo.InvariantCulture));
Console.WriteLine(String.Format(CultureInfo.InvariantCulture,
    "{0:00.00}", value));
// Displays 01.20

CultureInfo daDK = CultureInfo.CreateSpecificCulture("da-DK");
Console.WriteLine(value.ToString("00.00", daDK));
Console.WriteLine(String.Format(daDK, "{0:00.00}", value));
// Displays 01,20

value = .56;
Console.WriteLine(value.ToString("0.0", CultureInfo.InvariantCulture));
Console.WriteLine(String.Format(CultureInfo.InvariantCulture,
    "{0:0.0}", value));
// Displays 0.6

value = 1234567890;
Console.WriteLine(value.ToString("0,0", CultureInfo.InvariantCulture));
Console.WriteLine(String.Format(CultureInfo.InvariantCulture,
    "{0:,0}", value));
// Displays 1,234,567,890

CultureInfo elGR = CultureInfo.CreateSpecificCulture("el-GR");
Console.WriteLine(value.ToString("0,0", elGR));
Console.WriteLine(String.Format(elGR, "{0:,0}", value));
// Displays 1.234.567.890

value = 1234567890.123456;
Console.WriteLine(value.ToString("0,0.0", CultureInfo.InvariantCulture));
Console.WriteLine(String.Format(CultureInfo.InvariantCulture,
    "{0:,0.0}", value));
// Displays 1,234,567,890.1

value = 1234.567890;
Console.WriteLine(value.ToString("0,0.00", CultureInfo.InvariantCulture));
Console.WriteLine(String.Format(CultureInfo.InvariantCulture,
    "{0:,0.00}", value));
// Displays 1,234.57
```

```

Dim value As Double

value = 123
Console.WriteLine(value.ToString("00000"))
Console.WriteLine(String.Format("{0:00000}", value))
' Displays 00123

value = 1.2
Console.WriteLine(value.ToString("0.00", CultureInfo.InvariantCulture))
Console.WriteLine(String.Format(CultureInfo.InvariantCulture,
    "{0:0.00}", value))
' Displays 1.20
Console.WriteLine(value.ToString("00.00", CultureInfo.InvariantCulture))
Console.WriteLine(String.Format(CultureInfo.InvariantCulture,
    "{0:00.00}", value))
' Displays 01.20
Dim daDK As CultureInfo = CultureInfo.CreateSpecificCulture("da-DK")
Console.WriteLine(value.ToString("00.00", daDK))
Console.WriteLine(String.Format(daDK, "{0:00.00}", value))
' Displays 01,20

value = .56
Console.WriteLine(value.ToString("0.0", CultureInfo.InvariantCulture))
Console.WriteLine(String.Format(CultureInfo.InvariantCulture,
    "{0:0.0}", value))
' Displays 0.6

value = 1234567890
Console.WriteLine(value.ToString("0,0", CultureInfo.InvariantCulture))
Console.WriteLine(String.Format(CultureInfo.InvariantCulture,
    "{0:0,0}", value))
' Displays 1,234,567,890
Dim elGR As CultureInfo = CultureInfo.CreateSpecificCulture("el-GR")
Console.WriteLine(value.ToString("0,0", elGR))
Console.WriteLine(String.Format(elGR, "{0:0,0}", value))
' Displays 1.234.567.890

value = 1234567890.123456
Console.WriteLine(value.ToString("0,0.0", CultureInfo.InvariantCulture))
Console.WriteLine(String.Format(CultureInfo.InvariantCulture,
    "{0:0,0.0}", value))
' Displays 1,234,567,890.1

value = 1234.567890
Console.WriteLine(value.ToString("0,0.00", CultureInfo.InvariantCulture))
Console.WriteLine(String.Format(CultureInfo.InvariantCulture,
    "{0:0,0.00}", value))
' Displays 1,234.57

```

[Back to table](#)

The "#" custom specifier

The "#" custom format specifier serves as a digit-placeholder symbol. If the value that is being formatted has a digit in the position where the "#" symbol appears in the format string, that digit is copied to the result string. Otherwise, nothing is stored in that position in the result string.

Note that this specifier never displays a zero that is not a significant digit, even if zero is the only digit in the string. It will display zero only if it is a significant digit in the number that is being displayed.

The "##" format string causes the value to be rounded to the nearest digit preceding the decimal, where rounding away from zero is always used. For example, formatting 34.5 with "##" would result in the value 35.

The following example displays several values that are formatted by using custom format strings that include digit placeholders.

```
double value;

value = 1.2;
Console::WriteLine(value.ToString("#.##", CultureInfo::InvariantCulture));
Console::WriteLine(String::Format(CultureInfo::InvariantCulture,
                                "{0:#.##}", value));
// Displays 1.2

value = 123;
Console::WriteLine(value.ToString("#####"));
Console::WriteLine(String::Format("{0:#####}", value));
// Displays 123

value = 123456;
Console::WriteLine(value.ToString("[##-##-##]"));
Console::WriteLine(String::Format("{0:[##-##-##]}", value));
// Displays [12-34-56]

value = 1234567890;
Console::WriteLine(value.ToString("#"));
Console::WriteLine(String::Format("{0:#}", value));
// Displays 1234567890

Console::WriteLine(value.ToString("###) ###-####"));
Console::WriteLine(String::Format("{0:(###) ###-####}", value));
// Displays (123) 456-7890
```

```
double value;

value = 1.2;
Console.WriteLine(value.ToString("#.##", CultureInfo.InvariantCulture));
Console.WriteLine(String.Format(CultureInfo.InvariantCulture,
                                "{0:#.##}", value));
// Displays 1.2

value = 123;
Console.WriteLine(value.ToString("#####"));
Console.WriteLine(String.Format("{0:#####}", value));
// Displays 123

value = 123456;
Console.WriteLine(value.ToString("[##-##-##]"));
Console.WriteLine(String.Format("{0:[##-##-##]}", value));
// Displays [12-34-56]

value = 1234567890;
Console.WriteLine(value.ToString("#"));
Console.WriteLine(String.Format("{0:#}", value));
// Displays 1234567890

Console.WriteLine(value.ToString("###) ###-####"));
Console.WriteLine(String.Format("{0:(###) ###-####}", value));
// Displays (123) 456-7890
```

```

Dim value As Double

value = 1.2
Console.WriteLine(value.ToString("#.##", CultureInfo.InvariantCulture))
Console.WriteLine(String.Format(CultureInfo.InvariantCulture,
    "{0:#.##}", value))
' Displays 1.2

value = 123
Console.WriteLine(value.ToString("#####"))
Console.WriteLine(String.Format("{0:#####}", value))
' Displays 123

value = 123456
Console.WriteLine(value.ToString("[##-##-##]"))
Console.WriteLine(String.Format("{0:[##-##-##]}", value))
' Displays [12-34-56]

value = 1234567890
Console.WriteLine(value.ToString("#"))
Console.WriteLine(String.Format("{0:#}", value))
' Displays 1234567890

Console.WriteLine(value.ToString("###) ###-####"))
Console.WriteLine(String.Format("{0:(###) ###-####}", value))
' Displays (123) 456-7890

```

To return a result string in which absent digits or leading zeroes are replaced by spaces, use the [composite formatting feature](#) and specify a field width, as the following example illustrates.

```

using namespace System;

void main()
{
    Double value = .324;
    Console::WriteLine("The value is: '{0,5:#.###}'", value);
}
// The example displays the following output if the current culture
// is en-US:
//      The value is: '.324'

```

```

using System;

public class SpaceOrDigit
{
    public static void Main()
    {
        Double value = .324;
        Console.WriteLine("The value is: '{0,5:#.###}'", value);
    }
}
// The example displays the following output if the current culture
// is en-US:
//      The value is: '.324'

```

```

Module Example
    Public Sub Main()
        Dim value As Double = .324
        Console.WriteLine("The value is: '{0,5:#.###}'", value)
    End Sub
End Module
' The example displays the following output if the current culture
' is en-US:
'     The value is: ' .324'

```

[Back to table](#)

The "." custom specifier

The "." custom format specifier inserts a localized decimal separator into the result string. The first period in the format string determines the location of the decimal separator in the formatted value; any additional periods are ignored.

The character that is used as the decimal separator in the result string is not always a period; it is determined by the [NumberDecimalSeparator](#) property of the [NumberFormatInfo](#) object that controls formatting.

The following example uses the "." format specifier to define the location of the decimal point in several result strings.

```

double value;

value = 1.2;
Console::WriteLine(value.ToString("0.00", CultureInfo::InvariantCulture));
Console::WriteLine(String::Format(CultureInfo::InvariantCulture,
                                "{0:0.00}", value));
// Displays 1.20

Console::WriteLine(value.ToString("00.00", CultureInfo::InvariantCulture));
Console::WriteLine(String::Format(CultureInfo::InvariantCulture,
                                "{0:00.00}", value));
// Displays 01.20

Console::WriteLine(value.ToString("00.00",
                               CultureInfo::CreateSpecificCulture("da-DK")));
Console::WriteLine(String::Format(CultureInfo::CreateSpecificCulture("da-DK"),
                                "{0:00.00}", value));
// Displays 01,20

value = .086;
Console::WriteLine(value.ToString("#0.##%", CultureInfo::InvariantCulture));
Console::WriteLine(String::Format(CultureInfo::InvariantCulture,
                                "{0:#0.##%}", value));
// Displays 8.6%

value = 86000;
Console::WriteLine(value.ToString("0.###E+0", CultureInfo::InvariantCulture));
Console::WriteLine(String::Format(CultureInfo::InvariantCulture,
                                "{0:0.###E+0}", value));
// Displays 8.6E+4

```

```

double value;

value = 1.2;
Console.WriteLine(value.ToString("0.00", CultureInfo.InvariantCulture));
Console.WriteLine(String.Format(CultureInfo.InvariantCulture,
    "{0:0.00}", value));
// Displays 1.20

Console.WriteLine(value.ToString("00.00", CultureInfo.InvariantCulture));
Console.WriteLine(String.Format(CultureInfo.InvariantCulture,
    "{0:00.00}", value));
// Displays 01.20

Console.WriteLine(value.ToString("00.00",
    CultureInfo.CreateSpecificCulture("da-DK")));
Console.WriteLine(String.Format(CultureInfo.CreateSpecificCulture("da-DK"),
    "{0:00.00}", value));
// Displays 01,20

value = .086;
Console.WriteLine(value.ToString("#0.##%", CultureInfo.InvariantCulture));
Console.WriteLine(String.Format(CultureInfo.InvariantCulture,
    "{0:#0.##%}", value));
// Displays 8.6%

value = 86000;
Console.WriteLine(value.ToString("0.###E+0", CultureInfo.InvariantCulture));
Console.WriteLine(String.Format(CultureInfo.InvariantCulture,
    "{0:0.###E+0}", value));
// Displays 8.6E+4

```

```

Dim value As Double

value = 1.2
Console.Writeline(value.ToString("0.00", CultureInfo.InvariantCulture))
Console.Writeline(String.Format(CultureInfo.InvariantCulture,
    "{0:0.00}", value))
' Displays 1.20

Console.WriteLine(value.ToString("00.00", CultureInfo.InvariantCulture))
Console.WriteLine(String.Format(CultureInfo.InvariantCulture,
    "{0:00.00}", value))
' Displays 01.20

Console.WriteLine(value.ToString("00.00", _
    CultureInfo.CreateSpecificCulture("da-DK")))
Console.WriteLine(String.Format(CultureInfo.CreateSpecificCulture("da-DK"),
    "{0:00.00}", value))
' Displays 01,20

value = .086
Console.WriteLine(value.ToString("#0.##%", CultureInfo.InvariantCulture))
Console.WriteLine(String.Format(CultureInfo.InvariantCulture,
    "{0:#0.##%}", value))
' Displays 8.6%

value = 86000
Console.WriteLine(value.ToString("0.###E+0", CultureInfo.InvariantCulture))
Console.WriteLine(String.Format(CultureInfo.InvariantCulture,
    "{0:0.###E+0}", value))
' Displays 8.6E+4

```

[Back to table](#)

The "," custom specifier

The "," character serves as both a group separator and a number scaling specifier.

- Group separator: If one or more commas are specified between two digit placeholders (0 or #) that format the integral digits of a number, a group separator character is inserted between each number group in the integral part of the output.

The [NumberGroupSeparator](#) and [NumberGroupSizes](#) properties of the current [NumberFormatInfo](#) object determine the character used as the number group separator and the size of each number group. For example, if the string "#,#" and the invariant culture are used to format the number 1000, the output is "1,000".

- Number scaling specifier: If one or more commas are specified immediately to the left of the explicit or implicit decimal point, the number to be formatted is divided by 1000 for each comma. For example, if the string "0,," is used to format the number 100 million, the output is "100".

You can use group separator and number scaling specifiers in the same format string. For example, if the string "#,0,," and the invariant culture are used to format the number one billion, the output is "1,000".

The following example illustrates the use of the comma as a group separator.

```
double value = 1234567890;
Console::WriteLine(value.ToString("#,#", CultureInfo::InvariantCulture));
Console::WriteLine(String::Format(CultureInfo::InvariantCulture,
                                "{0:#,#}", value));
// Displays 1,234,567,890

Console::WriteLine(value.ToString("#,##0,,, CultureInfo::InvariantCulture));
Console::WriteLine(String::Format(CultureInfo::InvariantCulture,
                                "{0:#,##0,,,}", value));
// Displays 1,235
```

```
double value = 1234567890;
Console.WriteLine(value.ToString("#,#", CultureInfo.InvariantCulture));
Console.WriteLine(String.Format(CultureInfo.InvariantCulture,
                                "{0:#,#}", value));
// Displays 1,234,567,890

Console.WriteLine(value.ToString("#,##0,,, CultureInfo.InvariantCulture));
Console.WriteLine(String.Format(CultureInfo.InvariantCulture,
                                "{0:#,##0,,,}", value));
// Displays 1,235
```

```
Dim value As Double = 1234567890
Console.WriteLine(value.ToString("#,#", CultureInfo.InvariantCulture))
Console.WriteLine(String.Format(CultureInfo.InvariantCulture,
                                "{0:#,#}", value))
' Displays 1,234,567,890

Console.WriteLine(value.ToString("#,##0,,, CultureInfo.InvariantCulture))
Console.WriteLine(String.Format(CultureInfo.InvariantCulture,
                                "{0:#,##0,,,}", value))
' Displays 1,235
```

The following example illustrates the use of the comma as a specifier for number scaling.

```

double value = 1234567890;
Console::WriteLine(value.ToString("#,,", CultureInfo::InvariantCulture));
Console::WriteLine(String::Format(CultureInfo::InvariantCulture,
                                "{0:#,,}", value));
// Displays 1235

Console::WriteLine(value.ToString("#,,,", CultureInfo::InvariantCulture));
Console::WriteLine(String::Format(CultureInfo::InvariantCulture,
                                "{0:#,,,}", value));
// Displays 1

Console::WriteLine(value.ToString("#,##0,,, CultureInfo::InvariantCulture));
Console::WriteLine(String::Format(CultureInfo::InvariantCulture,
                                "{0:#,##0,,}", value));
// Displays 1,235

```

```

double value = 1234567890;
Console.WriteLine(value.ToString("#,,", CultureInfo.InvariantCulture));
Console.WriteLine(String.Format(CultureInfo.InvariantCulture,
                               "{0:#,,}", value));
// Displays 1235

Console.WriteLine(value.ToString("#,,, CultureInfo.InvariantCulture));
Console.WriteLine(String.Format(CultureInfo.InvariantCulture,
                               "{0:#,,,}", value));
// Displays 1

Console.WriteLine(value.ToString("#,##0,,, CultureInfo.InvariantCulture));
Console.WriteLine(String.Format(CultureInfo.InvariantCulture,
                               "{0:#,##0,,}", value));
// Displays 1,235

```

```

Dim value As Double = 1234567890
Console.WriteLine(value.ToString("#,, CultureInfo.InvariantCulture))
Console.WriteLine(String.Format(CultureInfo.InvariantCulture, "{0:#,,}", value))
' Displays 1235

Console.WriteLine(value.ToString("#,,, CultureInfo.InvariantCulture))
Console.WriteLine(String.Format(CultureInfo.InvariantCulture,
                               "{0:#,,,}", value))
' Displays 1

Console.WriteLine(value.ToString("#,##0,,, CultureInfo.InvariantCulture))
Console.WriteLine(String.Format(CultureInfo.InvariantCulture,
                               "{0:#,##0,,}", value))
' Displays 1,235

```

[Back to table](#)

The "%" custom specifier

A percent sign (%) in a format string causes a number to be multiplied by 100 before it is formatted. The localized percent symbol is inserted in the number at the location where the % appears in the format string. The percent character used is defined by the [PercentSymbol](#) property of the current [NumberFormatInfo](#) object.

The following example defines several custom format strings that include the "%" custom specifier.

```

double value = .086;
Console::WriteLine(value.ToString("#0.##%", CultureInfo::InvariantCulture));
Console::WriteLine(String::Format(CultureInfo::InvariantCulture,
                                "{0:#0.##%}", value));
// Displays 8.6%

```

```

double value = .086;
Console.WriteLine(value.ToString("#0.##%", CultureInfo.InvariantCulture));
Console.WriteLine(String.Format(CultureInfo.InvariantCulture,
                               "{0:#0.##%}", value));
// Displays 8.6%

```

```

Dim value As Double = .086
Console.WriteLine(value.ToString("#0.##%", CultureInfo.InvariantCulture))
Console.WriteLine(String.Format(CultureInfo.InvariantCulture,
                               "{0:#0.##%}", value))
' Displays 8.6%

```

[Back to table](#)

The "%o" custom specifier

A per mille character (%o or \u2030) in a format string causes a number to be multiplied by 1000 before it is formatted. The appropriate per mille symbol is inserted in the returned string at the location where the %o symbol appears in the format string. The per mille character used is defined by the [NumberFormatInfo.PerMilleSymbol](#) property of the object that provides culture-specific formatting information.

The following example defines a custom format string that includes the "%o" custom specifier.

```

double value = .00354;
String^ perMilleFmt = "#0.## " + '\u2030';
Console::WriteLine(value.ToString(perMilleFmt, CultureInfo::InvariantCulture));
Console::WriteLine(String::Format(CultureInfo::InvariantCulture,
                                "{0:" + perMilleFmt + "}", value));
// Displays 3.54%

```

```

double value = .00354;
string perMilleFmt = "#0.## " + '\u2030';
Console.WriteLine(value.ToString(perMilleFmt, CultureInfo.InvariantCulture));
Console.WriteLine(String.Format(CultureInfo.InvariantCulture,
                               "{0:" + perMilleFmt + "}", value));
// Displays 3.54%

```

```

Dim value As Double = .00354
Dim perMilleFmt As String = "#0.## " & ChrW(&h2030)
Console.WriteLine(value.ToString(perMilleFmt, CultureInfo.InvariantCulture))
Console.WriteLine(String.Format(CultureInfo.InvariantCulture,
                               "{0:" + perMilleFmt + "}", value))
' Displays 3.54 %

```

[Back to table](#)

The "E" and "e" custom specifiers

If any of the strings "E", "E+", "E-", "e", "e+", or "e-" are present in the format string and are followed immediately by at least one zero, the number is formatted by using scientific notation with an "E" or "e" inserted between the number and the exponent. The number of zeros following the scientific notation indicator determines the minimum number of digits to output for the exponent. The "E+" and "e+" formats indicate that a plus sign or minus sign should always precede the exponent. The "E", "E-", "e", or "e-" formats indicate that a sign character should precede only negative exponents.

The following example formats several numeric values using the specifiers for scientific notation.

```
double value = 86000;
Console::WriteLine(value.ToString("0.###E+0", CultureInfo::InvariantCulture));
Console::WriteLine(String::Format(CultureInfo::InvariantCulture,
    "{0:0.###E+0}", value));
// Displays 8.6E+4

Console::WriteLine(value.ToString("0.###E+000", CultureInfo::InvariantCulture));
Console::WriteLine(String::Format(CultureInfo::InvariantCulture,
    "{0:0.###E+000}", value));
// Displays 8.6E+004

Console::WriteLine(value.ToString("0.###E-000", CultureInfo::InvariantCulture));
Console::WriteLine(String::Format(CultureInfo::InvariantCulture,
    "{0:0.###E-000}", value));
// Displays 8.6E004
```

```
double value = 86000;
Console.WriteLine(value.ToString("0.###E+0", CultureInfo.InvariantCulture));
Console.WriteLine(String.Format(CultureInfo.InvariantCulture,
    "{0:0.###E+0}", value));
// Displays 8.6E+4

Console.WriteLine(value.ToString("0.###E+000", CultureInfo.InvariantCulture));
Console.WriteLine(String.Format(CultureInfo.InvariantCulture,
    "{0:0.###E+000}", value));
// Displays 8.6E+004

Console.WriteLine(value.ToString("0.###E-000", CultureInfo.InvariantCulture));
Console.WriteLine(String.Format(CultureInfo.InvariantCulture,
    "{0:0.###E-000}", value));
// Displays 8.6E004
```

```
Dim value As Double = 86000
Console.WriteLine(value.ToString("0.###E+0", CultureInfo.InvariantCulture))
Console.WriteLine(String.Format(CultureInfo.InvariantCulture,
    "{0:0.###E+0}", value))
' Displays 8.6E+4

Console.WriteLine(value.ToString("0.###E+000", CultureInfo.InvariantCulture))
Console.WriteLine(String.Format(CultureInfo.InvariantCulture,
    "{0:0.###E+000}", value))
' Displays 8.6E+004

Console.WriteLine(value.ToString("0.###E-000", CultureInfo.InvariantCulture))
Console.WriteLine(String.Format(CultureInfo.InvariantCulture,
    "{0:0.###E-000}", value))
' Displays 8.6E004
```

[Back to table](#)

The "\ escape character

The "#", "0", ".", ",", "%", and "%o" symbols in a format string are interpreted as format specifiers rather than as literal characters. Depending on their position in a custom format string, the uppercase and lowercase "E" as well as the + and - symbols may also be interpreted as format specifiers.

To prevent a character from being interpreted as a format specifier, you can precede it with a backslash, which is the escape character. The escape character signifies that the following character is a character literal that should be included in the result string unchanged.

To include a backslash in a result string, you must escape it with another backslash (\\\).

NOTE

Some compilers, such as the C++ and C# compilers, may also interpret a single backslash character as an escape character. To ensure that a string is interpreted correctly when formatting, you can use the verbatim string literal character (the @ character) before the string in C#, or add another backslash character before each backslash in C# and C++. The following C# example illustrates both approaches.

The following example uses the escape character to prevent the formatting operation from interpreting the "#", "0", and "\" characters as either escape characters or format specifiers. The C# examples uses an additional backslash to ensure that a backslash is interpreted as a literal character.

```
int value = 123;
Console::WriteLine(value.ToString("\#\#\#\#\##0 dollars and \0\0 cents \#\#\#\#\#"));
Console::WriteLine(String::Format("{0:\#\#\#\#\##0 dollars and \0\0 cents \#\#\#\#\#}",
                                value));
// Displays ### 123 dollars and 00 cents ###

Console::WriteLine(value.ToString("\#\#\#\#\##0 dollars and \0\0 cents \#\#\#\#\#"));
Console::WriteLine(String::Format("{0:\#\#\#\#\##0 dollars and \0\0 cents \#\#\#\#\#}",
                                value));
// Displays ### 123 dollars and 00 cents ###

Console::WriteLine(value.ToString("\\\\\\\\\\\\\\##0 dollars and \0\0 cents \\\\\\\\\\\\\\\\\"));
Console::WriteLine(String::Format("{0:\\\\\\\\\\\\\\##0 dollars and \0\0 cents \\\\\\\\\\\\\\\\\}",
                                value));
// Displays \\ 123 dollars and 00 cents \\

Console::WriteLine(value.ToString("\\\\\\##0 dollars and \0\0 cents \\\\\\\"));
Console::WriteLine(String::Format("{0:\\\\\\##0 dollars and \0\0 cents \\\\\\\}",
                                value));
// Displays \\\ 123 dollars and 00 cents \\
```

```
int value = 123;
Console.WriteLine(value.ToString("\#\#\#\#\## dollars and \0\0 cents \#\#\#\#"));
Console.WriteLine(String.Format("{0:\#\#\#\#\## dollars and \0\0 cents \#\#\#\#}",
                               value));
// Displays ### 123 dollars and 00 cents ###

Console.WriteLine(value.ToString(@"#\#\#\##0 dollars and \0\0 cents \#\#\#\#"));
Console.WriteLine(String.Format(@"{0:#\#\#\##0 dollars and \0\0 cents \#\#\#\#}",
                               value));
// Displays ### 123 dollars and 00 cents ###

Console.WriteLine(value.ToString("\\\\\\\\\\\\\\\\##0 dollars and \0\0 cents \\\\\\\\\\\\\\\\\"));
Console.WriteLine(String.Format("{0:\\\\\\\\\\\\\\\\##0 dollars and \0\0 cents \\\\\\\\\\\\\\\\\}",
                               value));
// Displays \\ 123 dollars and 00 cents \\

Console.WriteLine(value.ToString("\\\\\\\\##0 dollars and \0\0 cents \\\\\\\\\"));
Console.WriteLine(String.Format("{0:\\\\\\\\##0 dollars and \0\0 cents \\\\\\\\\}",
                               value));
// Displays \\\ 123 dollars and 00 cents \\\
```

```
Dim value As Integer = 123
Console.WriteLine(value.ToString("###\### ##0 dollars and \0\0 cents ###"))
Console.WriteLine(String.Format("{0:###\### ##0 dollars and \0\0 cents ###}", value))
' Displays ### 123 dollars and 00 cents ###

Console.WriteLine(value.ToString("\\\\##0 dollars and \0\0 cents \\\\\""))
Console.WriteLine(String.Format("{0:\\\\##0 dollars and \0\0 cents \\\\\"}", value))
' Displays \\\ 123 dollars and 00 cents \\
```

[Back to table](#)

The ";" section separator

The semicolon (:) is a conditional format specifier that applies different formatting to a number depending on whether its value is positive, negative, or zero. To produce this behavior, a custom format string can contain up to three sections separated by semicolons. These sections are described in the following table.

NUMBER OF SECTIONS	DESCRIPTION
One section	The format string applies to all values.
Two sections	<p>The first section applies to positive values and zeros, and the second section applies to negative values.</p> <p>If the number to be formatted is negative, but becomes zero after rounding according to the format in the second section, the resulting zero is formatted according to the first section.</p>

NUMBER OF SECTIONS	DESCRIPTION
Three sections	<p>The first section applies to positive values, the second section applies to negative values, and the third section applies to zeros.</p> <p>The second section can be left empty (by having nothing between the semicolons), in which case the first section applies to all nonzero values.</p> <p>If the number to be formatted is nonzero, but becomes zero after rounding according to the format in the first or second section, the resulting zero is formatted according to the third section.</p>

Section separators ignore any preexisting formatting associated with a number when the final value is formatted. For example, negative values are always displayed without a minus sign when section separators are used. If you want the final formatted value to have a minus sign, you should explicitly include the minus sign as part of the custom format specifier.

The following example uses the ";" format specifier to format positive, negative, and zero numbers differently.

```
double posValue = 1234;
double negValue = -1234;
double zeroValue = 0;

String^ fmt2 = "##;(##)";
String^ fmt3 = "##;(##);**Zero**";

Console::WriteLine(posValue.ToString(fmt2));
Console::WriteLine(String::Format("{0:" + fmt2 + "}", posValue));
// Displays 1234

Console::WriteLine(negValue.ToString(fmt2));
Console::WriteLine(String::Format("{0:" + fmt2 + "}", negValue));
// Displays (1234)

Console::WriteLine(zeroValue.ToString(fmt3));
Console::WriteLine(String::Format("{0:" + fmt3 + "}", zeroValue));
// Displays **Zero**
```

```
double posValue = 1234;
double negValue = -1234;
double zeroValue = 0;

string fmt2 = "##;(##)";
string fmt3 = "##;(##);**Zero**";

Console.WriteLine(posValue.ToString(fmt2));
Console.WriteLine(String.Format("{0:" + fmt2 + "}", posValue));
// Displays 1234

Console.WriteLine(negValue.ToString(fmt2));
Console.WriteLine(String.Format("{0:" + fmt2 + "}", negValue));
// Displays (1234)

Console.WriteLine(zeroValue.ToString(fmt3));
Console.WriteLine(String.Format("{0:" + fmt3 + "}", zeroValue));
// Displays **Zero**
```

```

Dim posValue As Double = 1234
Dim negValue As Double = -1234
Dim zeroValue As Double = 0

Dim fmt2 As String = "##;(##)"
Dim fmt3 As String = "##;(##);**Zero**"

Console.WriteLine(posValue.ToString(fmt2))
Console.WriteLine(String.Format("{0:" + fmt2 + "}", posValue))
' Displays 1234

Console.WriteLine(negValue.ToString(fmt2))
Console.WriteLine(String.Format("{0:" + fmt2 + "}", negValue))
' Displays (1234)

Console.WriteLine(zeroValue.ToString(fmt3))
Console.WriteLine(String.Format("{0:" + fmt3 + "}", zeroValue))
' Displays **Zero**

```

[Back to table](#)

Character literals

Format specifiers that appear in a custom numeric format string are always interpreted as formatting characters and never as literal characters. This includes the following characters:

- `0`
- `#`
- `%`
- `%o`
- `'`
- `\`
- `.`
- `,`
- `E` or `e`, depending on its position in the format string.

All other characters are always interpreted as character literals and, in a formatting operation, are included in the result string unchanged. In a parsing operation, they must match the characters in the input string exactly; the comparison is case-sensitive.

The following example illustrates one common use of literal character units (in this case, thousands):

```

double n = 123.8;
Console.WriteLine($"{n:#,##0.0K}");
// The example displays the following output:
//      123.8K

```

```

Dim n As Double = 123.8
Console.WriteLine($"{n:#,##0.0K}")
' The example displays the following output:
'      123.8K

```

There are two ways to indicate that characters are to be interpreted as literal characters and not as formatting characters, so that they can be included in a result string or successfully parsed in an input string:

- By escaping a formatting character. For more information, see [The "\\" escape character](#).

- By enclosing the entire literal string in quotation apostrophes.

The following example uses both approaches to include reserved characters in a custom numeric format string.

```
double n = 9.3;
Console.WriteLine(${n:##.0%});
Console.WriteLine(${n:'##'});
Console.WriteLine(${n:\\##\\'});
Console.WriteLine();
Console.WriteLine("${n:##.0%'}");
Console.WriteLine(${n:'##'\\'});
// The example displays the following output:
//      9.3%
//      '9'
//      \9\
//
//      9.3%
//      \9\
```

```
Dim n As Double = 9.3
Console.WriteLine(${n:##.0%})
Console.WriteLine(${n:'##'})
Console.WriteLine(${n:\\##\\'})
Console.WriteLine()
Console.WriteLine("${n:##.0%'}")
Console.WriteLine(${n:'##'\\'})
' The example displays the following output:
'      9.3%
'      '9'
'      \9\
'
'      9.3%
'      \9\
```

Notes

Floating-Point infinities and NaN

Regardless of the format string, if the value of a [Half](#), [Single](#), or [Double](#) floating-point type is positive infinity, negative infinity, or not a number (NaN), the formatted string is the value of the respective [PositiveInfinitySymbol](#), [NegativeInfinitySymbol](#), or [NaNSymbol](#) property specified by the currently applicable [NumberFormatInfo](#) object.

Control Panel settings

The settings in the **Regional and Language Options** item in Control Panel influence the result string produced by a formatting operation. Those settings are used to initialize the [NumberFormatInfo](#) object associated with the current culture, and the current culture provides values used to govern formatting. Computers that use different settings generate different result strings.

In addition, if you use the [CultureInfo\(String\)](#) constructor to instantiate a new [CultureInfo](#) object that represents the same culture as the current system culture, any customizations established by the **Regional and Language Options** item in Control Panel will be applied to the new [CultureInfo](#) object. You can use the [CultureInfo\(String, Boolean\)](#) constructor to create a [CultureInfo](#) object that does not reflect a system's customizations.

Rounding and fixed-point format strings

For fixed-point format strings (that is, format strings that do not contain scientific notation format characters), numbers are rounded to as many decimal places as there are digit placeholders to the right of the decimal point. If the format string does not contain a decimal point, the number is rounded to the nearest integer. If the

number has more digits than there are digit placeholders to the left of the decimal point, the extra digits are copied to the result string immediately before the first digit placeholder.

[Back to table](#)

Example

The following example demonstrates two custom numeric format strings. In both cases, the digit placeholder (#) displays the numeric data, and all other characters are copied to the result string.

```
double number1 = 1234567890;
String^ value1 = number1.ToString("(###) ###-####");
Console::WriteLine(value1);

int number2 = 42;
String^ value2 = number2.ToString("My Number = #");
Console::WriteLine(value2);
// The example displays the following output:
//      (123) 456-7890
//      My Number = 42
```

```
double number1 = 1234567890;
string value1 = number1.ToString("(###) ###-####");
Console.WriteLine(value1);

int number2 = 42;
string value2 = number2.ToString("My Number = #");
Console.WriteLine(value2);
// The example displays the following output:
//      (123) 456-7890
//      My Number = 42
```

```
Dim number1 As Double = 1234567890
Dim value1 As String = number1.ToString("(###) ###-####")
Console.WriteLine(value1)

Dim number2 As Integer = 42
Dim value2 As String = number2.ToString("My Number = #")
Console.WriteLine(value2)
' The example displays the following output:
'      (123) 456-7890
'      My Number = 42
```

[Back to table](#)

See also

- [System.Globalization.NumberFormatInfo](#)
- [Formatting Types](#)
- [Standard Numeric Format Strings](#)
- [How to: Pad a Number with Leading Zeros](#)
- [Sample: .NET Core WinForms Formatting Utility \(C#\)](#)
- [Sample: .NET Core WinForms Formatting Utility \(Visual Basic\)](#)

Standard date and time format strings

9/20/2022 • 29 minutes to read • [Edit Online](#)

A standard date and time format string uses a single character as the format specifier to define the text representation of a [DateTime](#) or a [DateTimeOffset](#) value. Any date and time format string that contains more than one character, including white space, is interpreted as a [custom date and time format string](#). A standard or custom format string can be used in two ways:

- To define the string that results from a formatting operation.
- To define the text representation of a date and time value that can be converted to a [DateTime](#) or [DateTimeOffset](#) value by a parsing operation.

TIP

You can download the [Formatting Utility](#), a .NET Windows Forms application that lets you apply format strings to either numeric or date and time values and display the result string. Source code is available for [C#](#) and [Visual Basic](#).

NOTE

Some of the C# examples in this article run in the [Try.NET](#) inline code runner and playground. Select the **Run** button to run an example in an interactive window. Once you execute the code, you can modify it and run the modified code by selecting **Run** again. The modified code either runs in the interactive window or, if compilation fails, the interactive window displays all C# compiler error messages.

The [local time zone](#) of the [Try.NET](#) inline code runner and playground is Coordinated Universal Time, or UTC. This may affect the behavior and the output of examples that illustrate the [DateTime](#), [DateTimeOffset](#), and [TimeZoneInfo](#) types and their members.

Table of format specifiers

The following table describes the standard date and time format specifiers. Unless otherwise noted, a particular standard date and time format specifier produces an identical string representation regardless of whether it is used with a [DateTime](#) or a [DateTimeOffset](#) value. See [Control Panel Settings](#) and [DateTimeFormatInfo Properties](#) for additional information about using standard date and time format strings.

FORMAT SPECIFIER	DESCRIPTION	EXAMPLES
"d"	Short date pattern. More information: The short date ("d") format specifier.	2009-06-15T13:45:30 -> 6/15/2009 (en-US) 2009-06-15T13:45:30 -> 15/06/2009 (fr-FR) 2009-06-15T13:45:30 -> 2009/06/15 (ja-JP)

FORMAT SPECIFIER	DESCRIPTION	EXAMPLES
"D"	<p>Long date pattern.</p> <p>More information: The long date ("D") format specifier.</p>	<p>2009-06-15T13:45:30 -> Monday, June 15, 2009 (en-US)</p> <p>2009-06-15T13:45:30 -> 15 июня 2009 г. (ru-RU)</p> <p>2009-06-15T13:45:30 -> Montag, 15. Juni 2009 (de-DE)</p>
"f"	<p>Full date/time pattern (short time).</p> <p>More information: The full date short time ("f") format specifier.</p>	<p>2009-06-15T13:45:30 -> Monday, June 15, 2009 1:45 PM (en-US)</p> <p>2009-06-15T13:45:30 -> den 15 juni 2009 13:45 (sv-SE)</p> <p>2009-06-15T13:45:30 -> Δευτέρα, 15 Ιουνίου 2009 1:45 μμ (el-GR)</p>
"F"	<p>Full date/time pattern (long time).</p> <p>More information: The full date long time ("F") format specifier.</p>	<p>2009-06-15T13:45:30 -> Monday, June 15, 2009 1:45:30 PM (en-US)</p> <p>2009-06-15T13:45:30 -> den 15 juni 2009 13:45:30 (sv-SE)</p> <p>2009-06-15T13:45:30 -> Δευτέρα, 15 Ιουνίου 2009 1:45:30 μμ (el-GR)</p>
"g"	<p>General date/time pattern (short time).</p> <p>More information: The general date short time ("g") format specifier.</p>	<p>2009-06-15T13:45:30 -> 6/15/2009 1:45 PM (en-US)</p> <p>2009-06-15T13:45:30 -> 15/06/2009 13:45 (es-ES)</p> <p>2009-06-15T13:45:30 -> 2009/6/15 13:45 (zh-CN)</p>
"G"	<p>General date/time pattern (long time).</p> <p>More information: The general date long time ("G") format specifier.</p>	<p>2009-06-15T13:45:30 -> 6/15/2009 1:45:30 PM (en-US)</p> <p>2009-06-15T13:45:30 -> 15/06/2009 13:45:30 (es-ES)</p> <p>2009-06-15T13:45:30 -> 2009/6/15 13:45:30 (zh-CN)</p>
"M", "m"	<p>Month/day pattern.</p> <p>More information: The month ("M", "m") format specifier.</p>	<p>2009-06-15T13:45:30 -> June 15 (en-US)</p> <p>2009-06-15T13:45:30 -> 15. juni (da-DK)</p> <p>2009-06-15T13:45:30 -> 15 Juni (id-ID)</p>

FORMAT SPECIFIER	DESCRIPTION	EXAMPLES
"O", "o"	<p>round-trip date/time pattern.</p> <p>More information: The round-trip ("O", "o") format specifier.</p>	<p>DateTime values:</p> <p>2009-06-15T13:45:30 (DateTimeKind.Local) --> 2009-06-15T13:45:30.0000000-07:00</p> <p>2009-06-15T13:45:30 (DateTimeKind.Utc) --> 2009-06-15T13:45:30.0000000Z</p> <p>2009-06-15T13:45:30 (DateTimeKind.Unspecified) --> 2009-06-15T13:45:30.0000000</p> <p>DateTimeOffset values:</p> <p>2009-06-15T13:45:30-07:00 --> 2009-06-15T13:45:30.0000000-07:00</p>
"R", "r"	<p>RFC1123 pattern.</p> <p>More information: The RFC1123 ("R", "r") format specifier.</p>	2009-06-15T13:45:30 -> Mon, 15 Jun 2009 20:45:30 GMT
"s"	<p>Sortable date/time pattern.</p> <p>More information: The sortable ("s") format specifier.</p>	<p>2009-06-15T13:45:30 (DateTimeKind.Local) -> 2009-06-15T13:45:30</p> <p>2009-06-15T13:45:30 (DateTimeKind.Utc) -> 2009-06-15T13:45:30</p>
"t"	<p>Short time pattern.</p> <p>More information: The short time ("t") format specifier.</p>	<p>2009-06-15T13:45:30 -> 1:45 PM (en-US)</p> <p>2009-06-15T13:45:30 -> 13:45 (hr-HR)</p> <p>2009-06-15T13:45:30 -> 01:45 ρ (ar-EG)</p>
"T"	<p>Long time pattern.</p> <p>More information: The long time ("T") format specifier.</p>	<p>2009-06-15T13:45:30 -> 1:45:30 PM (en-US)</p> <p>2009-06-15T13:45:30 -> 13:45:30 (hr-HR)</p> <p>2009-06-15T13:45:30 -> 01:45:30 ρ (ar-EG)</p>
"u"	<p>Universal sortable date/time pattern.</p> <p>More information: The universal sortable ("u") format specifier.</p>	<p>With a DateTime value: 2009-06-15T13:45:30 -> 2009-06-15 13:45:30Z</p> <p>With a DateTimeOffset value: 2009-06-15T13:45:30 -> 2009-06-15 20:45:30Z</p>

Format specifier	Description	Examples
"U"	Universal full date/time pattern. More information: The universal full ("U") format specifier .	2009-06-15T13:45:30 -> Monday, June 15, 2009 8:45:30 PM (en-US) 2009-06-15T13:45:30 -> den 15 juni 2009 20:45:30 (sv-SE) 2009-06-15T13:45:30 -> Δευτέρα, 15 Ιουνίου 2009 8:45:30 μμ (el-GR)
"Y", "y"	Year month pattern. More information: The year month ("Y") format specifier .	2009-06-15T13:45:30 -> June 2009 (en-US) 2009-06-15T13:45:30 -> juni 2009 (da-DK) 2009-06-15T13:45:30 -> Juni 2009 (id-ID)
Any other single character	Unknown specifier.	Throws a run-time FormatException .

How standard format strings work

In a formatting operation, a standard format string is simply an alias for a custom format string. The advantage of using an alias to refer to a custom format string is that, although the alias remains invariant, the custom format string itself can vary. This is important because the string representations of date and time values typically vary by culture. For example, the "d" standard format string indicates that a date and time value is to be displayed using a short date pattern. For the invariant culture, this pattern is "MM/dd/yyyy". For the fr-FR culture, it is "dd/MM/yyyy". For the ja-JP culture, it is "yyyy/MM/dd".

If a standard format string in a formatting operation maps to a particular culture's custom format string, your application can define the specific culture whose custom format strings are used in one of these ways:

- You can use the default (or current) culture. The following example displays a date using the current culture's short date format. In this case, the current culture is en-US.

```
// Display using current (en-us) culture's short date format
DateTime thisDate = new DateTime(2008, 3, 15);
Console.WriteLine(thisDate.ToString("d"));           // Displays 3/15/2008
```

```
' Display using current (en-us) culture's short date format
Dim thisDate As Date = #03/15/2008#
Console.WriteLine(thisDate.ToString("d"))      ' Displays 3/15/2008
```

- You can pass a [CultureInfo](#) object representing the culture whose formatting is to be used to a method that has an [IFormatProvider](#) parameter. The following example displays a date using the short date format of the pt-BR culture.

```
// Display using pt-BR culture's short date format
DateTime thisDate = new DateTime(2008, 3, 15);
CultureInfo culture = new CultureInfo("pt-BR");
Console.WriteLine(thisDate.ToString("d", culture)); // Displays 15/3/2008
```

```
' Display using pt-BR culture's short date format
Dim thisDate As Date = #03/15/2008#
Dim culture As New CultureInfo("pt-BR")
Console.WriteLine(thisDate.ToString("d", culture))      ' Displays 15/3/2008
```

- You can pass a [DateTimeFormatInfo](#) object that provides formatting information to a method that has an [IFormatProvider](#) parameter. The following example displays a date using the short date format from a [DateTimeFormatInfo](#) object for the hr-HR culture.

```
// Display using date format information from hr-HR culture
DateTime thisDate = new DateTime(2008, 3, 15);
DateTimeFormatInfo fmt = (new CultureInfo("hr-HR")).DateTimeFormat;
Console.WriteLine(thisDate.ToString("d", fmt));      // Displays 15.3.2008
```

```
' Display using date format information from hr-HR culture
Dim thisDate As Date = #03/15/2008#
Dim fmt As DateTimeFormatInfo = (New CultureInfo("hr-HR")).DateTimeFormat
Console.WriteLine(thisDate.ToString("d", fmt))      ' Displays 15.3.2008
```

NOTE

For information about customizing the patterns or strings used in formatting date and time values, see the [NumberFormatInfo](#) class topic.

In some cases, the standard format string serves as a convenient abbreviation for a longer custom format string that is invariant. Four standard format strings fall into this category: "O" (or "o"), "R" (or "r"), "s", and "u". These strings correspond to custom format strings defined by the invariant culture. They produce string representations of date and time values that are intended to be identical across cultures. The following table provides information on these four standard date and time format strings.

STANDARD FORMAT STRING	DEFINED BY DATETIMEFORMATINFO.INVARIANTIN FO PROPERTY	CUSTOM FORMAT STRING
"O" or "o"	None	yyyy'-'MM'-'dd'T'HH':'mm':'ss'.ffffffffK
"R" or "r"	RFC1123Pattern	ddd, dd MMM yyyy HH':'mm':'ss 'GMT'
"s"	SortableDateTimePattern	yyyy'-'MM'-'dd'T'HH':'mm':'ss
"u"	UniversalSortableDateTimePattern	yyyy'-'MM'-'dd HH':'mm':'ss'Z'

Standard format strings can also be used in parsing operations with the [DateTime.ParseExact](#) or [DateTimeOffset.ParseExact](#) methods, which require an input string to exactly conform to a particular pattern for the parse operation to succeed. Many standard format strings map to multiple custom format strings, so a date and time value can be represented in a variety of formats and the parse operation will still succeed. You can determine the custom format string or strings that correspond to a standard format string by calling the [DateTimeFormatInfo.GetAllDateTimePatterns\(Char\)](#) method. The following example displays the custom format strings that map to the "d" (short date pattern) standard format string.

```

using System;
using System.Globalization;

public class Example
{
    public static void Main()
    {
        Console.WriteLine("'d' standard format string:");
        foreach (var customString in DateTimeFormatInfo.CurrentInfo.GetAllDateTimePatterns('d'))
            Console.WriteLine("  {0}", customString);
    }
}
// The example displays the following output:
//      'd' standard format string:
//      M/d/yyyy
//      M/d/yy
//      MM/dd/yy
//      MM/dd/yyyy
//      yy/MM/dd
//      yyyy-MM-dd
//      dd-MMM-yy

```

```

Imports System.Globalization

Module Example
    Public Sub Main()
        Console.WriteLine("'d' standard format string:")
        For Each customString In DateTimeFormatInfo.CurrentInfo.GetAllDateTimePatterns("d"c)
            Console.WriteLine("  {0}", customString)
        Next
    End Sub
End Module
' The example displays the following output:
'      'd' standard format string:
'      M/d/yyyy
'      M/d/yy
'      MM/dd/yy
'      MM/dd/yyyy
'      yy/MM/dd
'      yyyy-MM-dd
'      dd-MMM-yy

```

The following sections describe the standard format specifiers for [DateTime](#) and [DateTimeOffset](#) values.

Date formats

This group includes the following formats:

- [The short date \("d"\) format specifier](#)
- [The long date \("D"\) format specifier](#)

The short date ("d") format specifier

The "d" standard format specifier represents a custom date and time format string that is defined by a specific culture's [DateTimeFormatInfo.ShortDatePattern](#) property. For example, the custom format string that is returned by the [ShortDatePattern](#) property of the invariant culture is "MM/dd/yyyy".

The following table lists the [DateTimeFormatInfo](#) object properties that control the formatting of the returned string.

PROPERTY	DESCRIPTION
ShortDatePattern	Defines the overall format of the result string.
DateSeparator	Defines the string that separates the year, month, and day components of a date.

The following example uses the "d" format specifier to display a date and time value.

```
DateTime date1 = new DateTime(2008,4, 10);
Console.WriteLine(date1.ToString("d", DateTimeFormatInfo.InvariantInfo));
// Displays 04/10/2008
Console.WriteLine(date1.ToString("d",
    CultureInfo.CreateSpecificCulture("en-US")));
// Displays 4/10/2008
Console.WriteLine(date1.ToString("d",
    CultureInfo.CreateSpecificCulture("en-NZ")));
// Displays 10/04/2008
Console.WriteLine(date1.ToString("d",
    CultureInfo.CreateSpecificCulture("de-DE")));
// Displays 10.04.2008
```

```
Dim date1 As Date = #4/10/2008#
Console.WriteLine(date1.ToString("d", DateTimeFormatInfo.InvariantInfo))
' Displays 04/10/2008
Console.WriteLine(date1.ToString("d",
    CultureInfo.CreateSpecificCulture("en-US")))
' Displays 4/10/2008
Console.WriteLine(date1.ToString("d",
    CultureInfo.CreateSpecificCulture("en-NZ")))
' Displays 10/04/2008
Console.WriteLine(date1.ToString("d",
    CultureInfo.CreateSpecificCulture("de-DE")))
' Displays 10.04.2008
```

[Back to table](#)

The long date ("D") format specifier

The "D" standard format specifier represents a custom date and time format string that is defined by the current [DateTimeFormatInfo.LongDatePattern](#) property. For example, the custom format string for the invariant culture is "dddd, dd MMMM yyyy".

The following table lists the properties of the [DateTimeFormatInfo](#) object that control the formatting of the returned string.

PROPERTY	DESCRIPTION
LongDatePattern	Defines the overall format of the result string.
DayNames	Defines the localized day names that can appear in the result string.
MonthNames	Defines the localized month names that can appear in the result string.

The following example uses the "D" format specifier to display a date and time value.

```

DateTime date1 = new DateTime(2008, 4, 10);
Console.WriteLine(date1.ToString("D",
    CultureInfo.CreateSpecificCulture("en-US")));
// Displays Thursday, April 10, 2008
Console.WriteLine(date1.ToString("D",
    CultureInfo.CreateSpecificCulture("pt-BR")));
// Displays quinta-feira, 10 de abril de 2008
Console.WriteLine(date1.ToString("D",
    CultureInfo.CreateSpecificCulture("es-MX")));
// Displays jueves, 10 de abril de 2008

```

```

Dim date1 As Date = #4/10/2008#
Console.WriteLine(date1.ToString("D", _
    CultureInfo.CreateSpecificCulture("en-US")))
' Displays Thursday, April 10, 2008
Console.WriteLine(date1.ToString("D", _
    CultureInfo.CreateSpecificCulture("pt-BR")))
' Displays quinta-feira, 10 de abril de 2008
Console.WriteLine(date1.ToString("D", _
    CultureInfo.CreateSpecificCulture("es-MX")))
' Displays jueves, 10 de abril de 2008

```

[Back to table](#)

Date and time formats

This group includes the following formats:

- [The full date short time \("f"\) format specifier](#)
- [The full date long time \("F"\) format specifier](#)
- [The general date short time \("g"\) format specifier](#)
- [The general date long time \("G"\) format specifier](#)
- [The round-trip \("O", "o"\) format specifier](#)
- [The RFC1123 \("R", "r"\) format specifier](#)
- [The sortable \("s"\) format specifier](#)
- [The universal sortable \("u"\) format specifier](#)
- [The universal full \("U"\) format specifier](#)

The full date short time ("f") format specifier

The "f" standard format specifier represents a combination of the long date ("D") and short time ("t") patterns, separated by a space.

The result string is affected by the formatting information of a specific [DateTimeFormatInfo](#) object. The following table lists the [DateTimeFormatInfo](#) object properties that may control the formatting of the returned string. The custom format specifier returned by the [DateTimeFormatInfo.LongDatePattern](#) and [DateTimeFormatInfo.ShortTimePattern](#) properties of some cultures may not make use of all properties.

PROPERTY	DESCRIPTION
LongDatePattern	Defines the format of the date component of the result string.
ShortTimePattern	Defines the format of the time component of the result string.

PROPERTY	DESCRIPTION
DayNames	Defines the localized day names that can appear in the result string.
MonthNames	Defines the localized month names that can appear in the result string.
TimeSeparator	Defines the string that separates the hour, minute, and second components of a time.
AMDesignator	Defines the string that indicates times from midnight to before noon in a 12-hour clock.
PMDesignator	Defines the string that indicates times from noon to before midnight in a 12-hour clock.

The following example uses the "f" format specifier to display a date and time value.

```
DateTime date1 = new DateTime(2008, 4, 10, 6, 30, 0);
Console.WriteLine(date1.ToString("f",
    CultureInfo.CreateSpecificCulture("en-US")));
// Displays Thursday, April 10, 2008 6:30 AM
Console.WriteLine(date1.ToString("f",
    CultureInfo.CreateSpecificCulture("fr-FR")));
// Displays jeudi 10 avril 2008 06:30
```

```
Dim date1 As Date = #4/10/2008 6:30AM#
Console.WriteLine(date1.ToString("f",
    CultureInfo.CreateSpecificCulture("en-US")))
' Displays Thursday, April 10, 2008 6:30 AM
Console.WriteLine(date1.ToString("f",
    CultureInfo.CreateSpecificCulture("fr-FR")))
' Displays jeudi 10 avril 2008 06:30
```

[Back to table](#)

The full date long time ("F") format specifier

The "F" standard format specifier represents a custom date and time format string that is defined by the current [DateTimeFormatInfo.FullDateTimePattern](#) property. For example, the custom format string for the invariant culture is "dddd, dd MMMM yyyy HH:mm:ss".

The following table lists the [DateTimeFormatInfo](#) object properties that may control the formatting of the returned string. The custom format specifier that is returned by the [FullDateTimePattern](#) property of some cultures may not make use of all properties.

PROPERTY	DESCRIPTION
FullDateTimePattern	Defines the overall format of the result string.
DayNames	Defines the localized day names that can appear in the result string.
MonthNames	Defines the localized month names that can appear in the result string.

PROPERTY	DESCRIPTION
TimeSeparator	Defines the string that separates the hour, minute, and second components of a time.
AMDesignator	Defines the string that indicates times from midnight to before noon in a 12-hour clock.
PMDesignator	Defines the string that indicates times from noon to before midnight in a 12-hour clock.

The following example uses the "F" format specifier to display a date and time value.

```
DateTime date1 = new DateTime(2008, 4, 10, 6, 30, 0);
Console.WriteLine(date1.ToString("F",
    CultureInfo.CreateSpecificCulture("en-US")));
// Displays Thursday, April 10, 2008 6:30:00 AM
Console.WriteLine(date1.ToString("F",
    CultureInfo.CreateSpecificCulture("fr-FR")));
// Displays jeudi 10 avril 2008 06:30:00
```

```
Dim date1 As Date = #4/10/2008 6:30AM#
Console.WriteLine(date1.ToString("F", _
    CultureInfo.CreateSpecificCulture("en-US")))
' Displays Thursday, April 10, 2008 6:30:00 AM
Console.WriteLine(date1.ToString("F", _
    CultureInfo.CreateSpecificCulture("fr-FR")))
' Displays jeudi 10 avril 2008 06:30:00
```

[Back to table](#)

The general date short time ("g") format specifier

The "g" standard format specifier represents a combination of the short date ("d") and short time ("t") patterns, separated by a space.

The result string is affected by the formatting information of a specific [DateTimeFormatInfo](#) object. The following table lists the [DateTimeFormatInfo](#) object properties that may control the formatting of the returned string. The custom format specifier that is returned by the [DateTimeFormatInfo.ShortDatePattern](#) and [DateTimeFormatInfo.ShortTimePattern](#) properties of some cultures may not make use of all properties.

PROPERTY	DESCRIPTION
ShortDatePattern	Defines the format of the date component of the result string.
ShortTimePattern	Defines the format of the time component of the result string.
DateSeparator	Defines the string that separates the year, month, and day components of a date.
TimeSeparator	Defines the string that separates the hour, minute, and second components of a time.

PROPERTY	DESCRIPTION
AMDesignator	Defines the string that indicates times from midnight to before noon in a 12-hour clock.
PMDesignator	Defines the string that indicates times from noon to before midnight in a 12-hour clock.

The following example uses the "g" format specifier to display a date and time value.

```
DateTime date1 = new DateTime(2008, 4, 10, 6, 30, 0);
Console.WriteLine(date1.ToString("g",
    DateTimeFormatInfo.InvariantInfo));
// Displays 04/10/2008 06:30
Console.WriteLine(date1.ToString("g",
    CultureInfo.CreateSpecificCulture("en-us")));
// Displays 4/10/2008 6:30 AM
Console.WriteLine(date1.ToString("g",
    CultureInfo.CreateSpecificCulture("fr-BE")));
// Displays 10/04/2008 6:30
```

```
Dim date1 As Date = #4/10/2008 6:30AM#
Console.WriteLine(date1.ToString("g", _
    DateTimeFormatInfo.InvariantInfo))
' Displays 04/10/2008 06:30
Console.WriteLine(date1.ToString("g", _
    CultureInfo.CreateSpecificCulture("en-us")))
' Displays 4/10/2008 6:30 AM
Console.WriteLine(date1.ToString("g", _
    CultureInfo.CreateSpecificCulture("fr-BE")))
' Displays 10/04/2008 6:30
```

[Back to table](#)

The general date long time ("G") format specifier

The "G" standard format specifier represents a combination of the short date ("d") and long time ("T") patterns, separated by a space.

The result string is affected by the formatting information of a specific [DateTimeFormatInfo](#) object. The following table lists the [DateTimeFormatInfo](#) object properties that may control the formatting of the returned string. The custom format specifier that is returned by the [DateTimeFormatInfo.ShortDatePattern](#) and [DateTimeFormatInfo.LongTimePattern](#) properties of some cultures may not make use of all properties.

PROPERTY	DESCRIPTION
ShortDatePattern	Defines the format of the date component of the result string.
LongTimePattern	Defines the format of the time component of the result string.
DateSeparator	Defines the string that separates the year, month, and day components of a date.
TimeSeparator	Defines the string that separates the hour, minute, and second components of a time.

PROPERTY	DESCRIPTION
AMDesignator	Defines the string that indicates times from midnight to before noon in a 12-hour clock.
PMDesignator	Defines the string that indicates times from noon to before midnight in a 12-hour clock.

The following example uses the "G" format specifier to display a date and time value.

```
DateTime date1 = new DateTime(2008, 4, 10, 6, 30, 0);
Console.WriteLine(date1.ToString("G",
    DateTimeFormatInfo.InvariantInfo));
// Displays 04/10/2008 06:30:00
Console.WriteLine(date1.ToString("G",
    CultureInfo.CreateSpecificCulture("en-us")));
// Displays 4/10/2008 6:30:00 AM
Console.WriteLine(date1.ToString("G",
    CultureInfo.CreateSpecificCulture("nl-BE")));
// Displays 10/04/2008 6:30:00
```

```
Dim date1 As Date = #4/10/2008 6:30AM#
Console.WriteLine(date1.ToString("G",
    DateTimeFormatInfo.InvariantInfo))
' Displays 04/10/2008 06:30:00
Console.WriteLine(date1.ToString("G",
    CultureInfo.CreateSpecificCulture("en-us")))
' Displays 4/10/2008 6:30:00 AM
Console.WriteLine(date1.ToString("G",
    CultureInfo.CreateSpecificCulture("nl-BE")))
' Displays 10/04/2008 6:30:00
```

[Back to table](#)

The round-trip ("O", "o") format specifier

The "O" or "o" standard format specifier represents a custom date and time format string using a pattern that preserves time zone information and emits a result string that complies with ISO 8601. For [DateTime](#) values, this format specifier is designed to preserve date and time values along with the [DateTime.Kind](#) property in text. The formatted string can be parsed back by using the [DateTime.Parse\(String, IFormatProvider, DateTimeStyles\)](#) or [DateTime.ParseExact](#) method if the `styles` parameter is set to [DateTimeStyles.RoundtripKind](#).

The "O" or "o" standard format specifier corresponds to the "yyyy'-MM'-dd'T'HH':mm':ss'.fffffffK" custom format string for [DateTime](#) values and to the "yyyy'-MM'-dd'T'HH':mm':ss'.fffffffzzz" custom format string for [DateTimeOffset](#) values. In this string, the pairs of single quotation marks that delimit individual characters, such as the hyphens, the colons, and the letter "T", indicate that the individual character is a literal that cannot be changed. The apostrophes do not appear in the output string.

The "O" or "o" standard format specifier (and the "yyyy'-MM'-dd'T'HH':mm':ss'.fffffffK" custom format string) takes advantage of the three ways that ISO 8601 represents time zone information to preserve the [Kind](#) property of [DateTime](#) values:

- The time zone component of [DateTimeKind.Local](#) date and time values is an offset from UTC (for example, +01:00, -07:00). All [DateTimeOffset](#) values are also represented in this format.
- The time zone component of [DateTimeKind.Utc](#) date and time values uses "Z" (which stands for zero offset) to represent UTC.

- [DateTimeKind.Unspecified](#) date and time values have no time zone information.

Because the "O" or "o" standard format specifier conforms to an international standard, the formatting or parsing operation that uses the specifier always uses the invariant culture and the Gregorian calendar.

Strings that are passed to the `Parse`, `TryParse`, `ParseExact`, and `TryParseExact` methods of [DateTime](#) and [DateTimeOffset](#) can be parsed by using the "O" or "o" format specifier if they are in one of these formats. In the case of [DateTime](#) objects, the parsing overload that you call should also include a `styles` parameter with a value of [DateTimeStyles.RoundtripKind](#). Note that if you call a parsing method with the custom format string that corresponds to the "O" or "o" format specifier, you won't get the same results as "O" or "o". This is because parsing methods that use a custom format string can't parse the string representation of date and time values that lack a time zone component or use "Z" to indicate UTC.

The following example uses the "o" format specifier to display a series of [DateTime](#) values and a [DateTimeOffset](#) value on a system in the U.S. Pacific Time zone.

```
using System;

public class Example
{
    public static void Main()
    {
        DateTime dat = new DateTime(2009, 6, 15, 13, 45, 30,
                                    DateTimeKind.Unspecified);
        Console.WriteLine("{0} ({1}) --> {0:0}", dat, dat.Kind);

        DateTime uDat = new DateTime(2009, 6, 15, 13, 45, 30,
                                    DateTimeKind.Utc);
        Console.WriteLine("{0} ({1}) --> {0:0}", uDat, uDat.Kind);

        DateTime lDat = new DateTime(2009, 6, 15, 13, 45, 30,
                                    DateTimeKind.Local);
        Console.WriteLine("{0} ({1}) --> {0:0}\n", lDat, lDat.Kind);

        DateTimeOffset dto = new DateTimeOffset(lDat);
        Console.WriteLine("{0} --> {0:0}", dto);
    }
}

// The example displays the following output:
//   6/15/2009 1:45:30 PM (Unspecified) --> 2009-06-15T13:45:30.0000000
//   6/15/2009 1:45:30 PM (Utc) --> 2009-06-15T13:45:30.0000000Z
//   6/15/2009 1:45:30 PM (Local) --> 2009-06-15T13:45:30.0000000-07:00
//
//   6/15/2009 1:45:30 PM -07:00 --> 2009-06-15T13:45:30.0000000-07:00
```

```

Module Example
    Public Sub Main()
        Dim dat As New Date(2009, 6, 15, 13, 45, 30,
                           DateTimeKind.Unspecified)
        Console.WriteLine("{0} ({1}) --> {0:0}", dat, dat.Kind)

        Dim uDat As New Date(2009, 6, 15, 13, 45, 30, DateTimeKind.Utc)
        Console.WriteLine("{0} ({1}) --> {0:0}", uDat, uDat.Kind)

        Dim lDat As New Date(2009, 6, 15, 13, 45, 30, DateTimeKind.Local)
        Console.WriteLine("{0} ({1}) --> {0:0}", lDat, lDat.Kind)
        Console.WriteLine()

        Dim dto As New DateTimeOffset(lDat)
        Console.WriteLine("{0} --> {0:0}", dto)
    End Sub
End Module
' The example displays the following output:
'   6/15/2009 1:45:30 PM (Unspecified) --> 2009-06-15T13:45:30.0000000
'   6/15/2009 1:45:30 PM (Utc) --> 2009-06-15T13:45:30.0000000Z
'   6/15/2009 1:45:30 PM (Local) --> 2009-06-15T13:45:30.0000000-07:00
'
'   6/15/2009 1:45:30 PM -07:00 --> 2009-06-15T13:45:30.0000000-07:00

```

The following example uses the "o" format specifier to create a formatted string, and then restores the original date and time value by calling a date and time `Parse` method.

```

// Round-trip DateTime values.
DateTime originalDate, newDate;
string dateString;
// Round-trip a local time.
originalDate = DateTime.SpecifyKind(new DateTime(2008, 4, 10, 6, 30, 0), DateTimeKind.Local);
dateString = originalDate.ToString("o");
newDate = DateTime.Parse(dateString, null, DateTimeStyles.RoundtripKind);
Console.WriteLine("Round-tripped {0} {1} to {2} {3}.", originalDate, originalDate.Kind,
                 newDate, newDate.Kind);
// Round-trip a UTC time.
originalDate = DateTime.SpecifyKind(new DateTime(2008, 4, 12, 9, 30, 0), DateTimeKind.Utc);
dateString = originalDate.ToString("o");
newDate = DateTime.Parse(dateString, null, DateTimeStyles.RoundtripKind);
Console.WriteLine("Round-tripped {0} {1} to {2} {3}.", originalDate, originalDate.Kind,
                 newDate, newDate.Kind);
// Round-trip time in an unspecified time zone.
originalDate = DateTime.SpecifyKind(new DateTime(2008, 4, 13, 12, 30, 0), DateTimeKind.Unspecified);
dateString = originalDate.ToString("o");
newDate = DateTime.Parse(dateString, null, DateTimeStyles.RoundtripKind);
Console.WriteLine("Round-tripped {0} {1} to {2} {3}.", originalDate, originalDate.Kind,
                 newDate, newDate.Kind);

// Round-trip a DateTimeOffset value.
DateTimeOffset originalDTO = new DateTimeOffset(2008, 4, 12, 9, 30, 0, new TimeSpan(-8, 0, 0));
dateString = originalDTO.ToString("o");
DateTimeOffset newDTO = DateTimeOffset.Parse(dateString, null, DateTimeStyles.RoundtripKind);
Console.WriteLine("Round-tripped {0} to {1}.", originalDTO, newDTO);
// The example displays the following output:
//   Round-tripped 4/10/2008 6:30:00 AM Local to 4/10/2008 6:30:00 AM Local.
//   Round-tripped 4/12/2008 9:30:00 AM Utc to 4/12/2008 9:30:00 AM Utc.
//   Round-tripped 4/13/2008 12:30:00 PM Unspecified to 4/13/2008 12:30:00 PM Unspecified.
//   Round-tripped 4/12/2008 9:30:00 AM -08:00 to 4/12/2008 9:30:00 AM -08:00.

```

```

' Round-trip DateTime values.
Dim originalDate, newDate As Date
Dim dateString As String
' Round-trip a local time.
originalDate = Date.SpecifyKind(#4/10/2008 6:30AM#, DateTimeKind.Local)
dateString = originalDate.ToString("o")
newDate = Date.Parse(dateString, Nothing, DateTimeStyles.RoundtripKind)
Console.WriteLine("Round-tripped {0} {1} to {2} {3}.", originalDate, originalDate.Kind, _
    newDate, newDate.Kind)
' Round-trip a UTC time.
originalDate = Date.SpecifyKind(#4/12/2008 9:30AM#, DateTimeKind.Utc)
dateString = originalDate.ToString("o")
newDate = Date.Parse(dateString, Nothing, DateTimeStyles.RoundtripKind)
Console.WriteLine("Round-tripped {0} {1} to {2} {3}.", originalDate, originalDate.Kind, _
    newDate, newDate.Kind)
' Round-trip time in an unspecified time zone.
originalDate = Date.SpecifyKind(#4/13/2008 12:30PM#, DateTimeKind.Unspecified)
dateString = originalDate.ToString("o")
newDate = Date.Parse(dateString, Nothing, DateTimeStyles.RoundtripKind)
Console.WriteLine("Round-tripped {0} {1} to {2} {3}.", originalDate, originalDate.Kind, _
    newDate, newDate.Kind)

' Round-trip a DateTimeOffset value.
Dim originalDTO As New DateTimeOffset(#4/12/2008 9:30AM#, New TimeSpan(-8, 0, 0))
dateString = originalDTO.ToString("o")
Dim newDTO As DateTimeOffset = DateTimeOffset.Parse(dateString, Nothing, DateTimeStyles.RoundtripKind)
Console.WriteLine("Round-tripped {0} to {1}.", originalDTO, newDTO)
' The example displays the following output:
'   Round-tripped 4/10/2008 6:30:00 AM Local to 4/10/2008 6:30:00 AM Local.
'   Round-tripped 4/12/2008 9:30:00 AM Utc to 4/12/2008 9:30:00 AM Utc.
'   Round-tripped 4/13/2008 12:30:00 PM Unspecified to 4/13/2008 12:30:00 PM Unspecified.
'   Round-tripped 4/12/2008 9:30:00 AM -08:00 to 4/12/2008 9:30:00 AM -08:00.

```

[Back to table](#)

The RFC1123 ("R", "r") format specifier

The "R" or "r" standard format specifier represents a custom date and time format string that is defined by the [DateTimeFormatInfo.RFC1123Pattern](#) property. The pattern reflects a defined standard, and the property is read-only. Therefore, it is always the same, regardless of the culture used or the format provider supplied. The custom format string is "ddd, dd MMM yyyy HH':mm':ss 'GMT'". When this standard format specifier is used, the formatting or parsing operation always uses the invariant culture.

The result string is affected by the following properties of the [DateTimeFormatInfo](#) object returned by the [DateTimeFormatInfo.InvariantInfo](#) property that represents the invariant culture.

PROPERTY	DESCRIPTION
RFC1123Pattern	Defines the format of the result string.
AbbreviatedDayNames	Defines the abbreviated day names that can appear in the result string.
AbbreviatedMonthNames	Defines the abbreviated month names that can appear in the result string.

Although the RFC 1123 standard expresses a time as Coordinated Universal Time (UTC), the formatting operation does not modify the value of the [DateTime](#) object that is being formatted. Therefore, you must convert the [DateTime](#) value to UTC by calling the [DateTime.ToUniversalTime](#) method before you perform the formatting operation. In contrast, [DateTimeOffset](#) values perform this conversion automatically; there is no need

to call the [DateTimeOffset.ToUniversalTime](#) method before the formatting operation.

The following example uses the "r" format specifier to display a [DateTime](#) and a [DateTimeOffset](#) value on a system in the U.S. Pacific Time zone.

```
DateTime date1 = new DateTime(2008, 4, 10, 6, 30, 0);
DateTimeOffset dateOffset = new DateTimeOffset(date1,
                                              TimeZoneInfo.Local.GetUtcOffset(date1));
Console.WriteLine(date1.ToUniversalTime().ToString("r"));
// Displays Thu, 10 Apr 2008 13:30:00 GMT
Console.WriteLine(dateOffset.ToUniversalTime().ToString("r"));
// Displays Thu, 10 Apr 2008 13:30:00 GMT
```

```
Dim date1 As Date = #4/10/2008 6:30AM#
Dim dateOffset As New DateTimeOffset(date1, TimeZoneInfo.Local.GetUtcOffset(date1))
Console.WriteLine(date1.ToUniversalTime.ToString("r"))
' Displays Thu, 10 Apr 2008 13:30:00 GMT
Console.WriteLine(dateOffset.ToUniversalTime.ToString("r"))
' Displays Thu, 10 Apr 2008 13:30:00 GMT
```

[Back to table](#)

The sortable ("s") format specifier

The "s" standard format specifier represents a custom date and time format string that is defined by the [DateTimeFormatInfo.SortableDateTimePattern](#) property. The pattern reflects a defined standard (ISO 8601), and the property is read-only. Therefore, it is always the same, regardless of the culture used or the format provider supplied. The custom format string is "yyyy'-MM'-dd'T'HH':mm':ss".

The purpose of the "s" format specifier is to produce result strings that sort consistently in ascending or descending order based on date and time values. As a result, although the "s" standard format specifier represents a date and time value in a consistent format, the formatting operation does not modify the value of the date and time object that is being formatted to reflect its [DateTime.Kind](#) property or its [DateTimeOffset.Offset](#) value. For example, the result strings produced by formatting the date and time values 2014-11-15T18:32:17+00:00 and 2014-11-15T18:32:17+08:00 are identical.

When this standard format specifier is used, the formatting or parsing operation always uses the invariant culture.

The following example uses the "s" format specifier to display a [DateTime](#) and a [DateTimeOffset](#) value on a system in the U.S. Pacific Time zone.

```
DateTime date1 = new DateTime(2008, 4, 10, 6, 30, 0);
Console.WriteLine(date1.ToString("s"));
// Displays 2008-04-10T06:30:00
```

```
Dim date1 As Date = #4/10/2008 6:30AM#
Console.WriteLine(date1.ToString("s"))
' Displays 2008-04-10T06:30:00
```

[Back to table](#)

The universal sortable ("u") format specifier

The "u" standard format specifier represents a custom date and time format string that is defined by the [DateTimeFormatInfo.UniversalSortableDateTimePattern](#) property. The pattern reflects a defined standard, and

the property is read-only. Therefore, it is always the same, regardless of the culture used or the format provider supplied. The custom format string is "yyyy'-'MM'-'dd HH':'mm':'ss'Z'". When this standard format specifier is used, the formatting or parsing operation always uses the invariant culture.

Although the result string should express a time as Coordinated Universal Time (UTC), no conversion of the original [DateTime](#) value is performed during the formatting operation. Therefore, you must convert a [DateTime](#) value to UTC by calling the [DateTime.ToUniversalTime](#) method before formatting it. In contrast, [DateTimeOffset](#) values perform this conversion automatically; there is no need to call the [DateTimeOffset.ToUniversalTime](#) method before the formatting operation.

The following example uses the "u" format specifier to display a date and time value.

```
DateTime date1 = new DateTime(2008, 4, 10, 6, 30, 0);
Console.WriteLine(date1.ToUniversalTime().ToString("u"));
// Displays 2008-04-10 13:30:00Z
```

```
Dim date1 As Date = #4/10/2008 6:30AM#
Console.WriteLine(date1.ToUniversalTime.ToString("u"))
' Displays 2008-04-10 13:30:00Z
```

[Back to table](#)

The universal full ("U") format specifier

The "U" standard format specifier represents a custom date and time format string that is defined by a specified culture's [DateTimeFormatInfo.FullDateTimePattern](#) property. The pattern is the same as the "F" pattern. However, the [DateTime](#) value is automatically converted to UTC before it is formatted.

The following table lists the [DateTimeFormatInfo](#) object properties that may control the formatting of the returned string. The custom format specifier that is returned by the [FullDateTimePattern](#) property of some cultures may not make use of all properties.

PROPERTY	DESCRIPTION
FullDateTimePattern	Defines the overall format of the result string.
DayNames	Defines the localized day names that can appear in the result string.
MonthNames	Defines the localized month names that can appear in the result string.
TimeSeparator	Defines the string that separates the hour, minute, and second components of a time.
AMDesignator	Defines the string that indicates times from midnight to before noon in a 12-hour clock.
PMDesignator	Defines the string that indicates times from noon to before midnight in a 12-hour clock.

The "U" format specifier is not supported by the [DateTimeOffset](#) type and throws a [FormatException](#) if it is used to format a [DateTimeOffset](#) value.

The following example uses the "U" format specifier to display a date and time value.

```

DateTime date1 = new DateTime(2008, 4, 10, 6, 30, 0);
Console.WriteLine(date1.ToString("U",
    CultureInfo.CreateSpecificCulture("en-US")));
// Displays Thursday, April 10, 2008 1:30:00 PM
Console.WriteLine(date1.ToString("U",
    CultureInfo.CreateSpecificCulture("sv-FI")));
// Displays den 10 april 2008 13:30:00

```

```

Dim date1 As Date = #4/10/2008 6:30AM#
Console.WriteLine(date1.ToString("U", CultureInfo.CreateSpecificCulture("en-US")))
' Displays Thursday, April 10, 2008 1:30:00 PM
Console.WriteLine(date1.ToString("U", CultureInfo.CreateSpecificCulture("sv-FI")))
' Displays den 10 april 2008 13:30:00

```

[Back to table](#)

Time formats

This group includes the following formats:

- [The short time \("t"\) format specifier](#)
- [The long time \("T"\) format specifier](#)

The short time ("t") format specifier

The "t" standard format specifier represents a custom date and time format string that is defined by the current [DateTimeFormatInfo.ShortTimePattern](#) property. For example, the custom format string for the invariant culture is "HH:mm".

The result string is affected by the formatting information of a specific [DateTimeFormatInfo](#) object. The following table lists the [DateTimeFormatInfo](#) object properties that may control the formatting of the returned string. The custom format specifier that is returned by the [DateTimeFormatInfo.ShortTimePattern](#) property of some cultures may not make use of all properties.

PROPERTY	DESCRIPTION
ShortTimePattern	Defines the format of the time component of the result string.
TimeSeparator	Defines the string that separates the hour, minute, and second components of a time.
AMDesignator	Defines the string that indicates times from midnight to before noon in a 12-hour clock.
PMDesignator	Defines the string that indicates times from noon to before midnight in a 12-hour clock.

The following example uses the "t" format specifier to display a date and time value.

```

DateTime date1 = new DateTime(2008, 4, 10, 6, 30, 0);
Console.WriteLine(date1.ToString("t",
    CultureInfo.CreateSpecificCulture("en-us")));
// Displays 6:30 AM
Console.WriteLine(date1.ToString("t",
    CultureInfo.CreateSpecificCulture("es-ES")));
// Displays 6:30

```

```

Dim date1 As Date = #4/10/2008 6:30AM#
Console.WriteLine(date1.ToString("t", _
    CultureInfo.CreateSpecificCulture("en-us")))
' Displays 6:30 AM
Console.WriteLine(date1.ToString("t", _
    CultureInfo.CreateSpecificCulture("es-ES")))
' Displays 6:30

```

[Back to table](#)

The long time ("T") format specifier

The "T" standard format specifier represents a custom date and time format string that is defined by a specific culture's [DateTimeFormatInfo.LongTimePattern](#) property. For example, the custom format string for the invariant culture is "HH:mm:ss".

The following table lists the [DateTimeFormatInfo](#) object properties that may control the formatting of the returned string. The custom format specifier that is returned by the [DateTimeFormatInfo.LongTimePattern](#) property of some cultures may not make use of all properties.

PROPERTY	DESCRIPTION
LongTimePattern	Defines the format of the time component of the result string.
TimeSeparator	Defines the string that separates the hour, minute, and second components of a time.
AMDesignator	Defines the string that indicates times from midnight to before noon in a 12-hour clock.
PMDesignator	Defines the string that indicates times from noon to before midnight in a 12-hour clock.

The following example uses the "T" format specifier to display a date and time value.

```

DateTime date1 = new DateTime(2008, 4, 10, 6, 30, 0);
Console.WriteLine(date1.ToString("T",
    CultureInfo.CreateSpecificCulture("en-us")));
// Displays 6:30:00 AM
Console.WriteLine(date1.ToString("T",
    CultureInfo.CreateSpecificCulture("es-ES")));
// Displays 6:30:00

```

```

Dim date1 As Date = #4/10/2008 6:30AM#
Console.WriteLine(date1.ToString("T", _
    CultureInfo.CreateSpecificCulture("en-US")))
' Displays 6:30:00 AM
Console.WriteLine(date1.ToString("T", _
    CultureInfo.CreateSpecificCulture("es-ES")))
' Displays 6:30:00

```

[Back to table](#)

Partial date formats

This group includes the following formats:

- [The month \("M", "m"\) format specifier](#)
- [The year month \("Y", "y"\) format specifier](#)

The month ("M", "m") format specifier

The "M" or "m" standard format specifier represents a custom date and time format string that is defined by the current [DateTimeFormatInfo.MonthDayPattern](#) property. For example, the custom format string for the invariant culture is "MMMM dd".

The following table lists the [DateTimeFormatInfo](#) object properties that control the formatting of the returned string.

PROPERTY	DESCRIPTION
MonthDayPattern	Defines the overall format of the result string.
MonthNames	Defines the localized month names that can appear in the result string.

The following example uses the "m" format specifier to display a date and time value.

```

DateTime date1 = new DateTime(2008, 4, 10, 6, 30, 0);
Console.WriteLine(date1.ToString("m",
    CultureInfo.CreateSpecificCulture("en-US")));
// Displays April 10
Console.WriteLine(date1.ToString("m",
    CultureInfo.CreateSpecificCulture("ms-MY")));
// Displays 10 April

```

```

Dim date1 As Date = #4/10/2008 6:30AM#
Console.WriteLine(date1.ToString("m", _
    CultureInfo.CreateSpecificCulture("en-US")))
' Displays April 10
Console.WriteLine(date1.ToString("m", _
    CultureInfo.CreateSpecificCulture("ms-MY")))
' Displays 10 April

```

[Back to table](#)

The year month ("Y", "y") format specifier

The "Y" or "y" standard format specifier represents a custom date and time format string that is defined by the [DateTimeFormatInfo.YearMonthPattern](#) property of a specified culture. For example, the custom format string

for the invariant culture is "yyyy MMMM".

The following table lists the [DateTimeFormatInfo](#) object properties that control the formatting of the returned string.

PROPERTY	DESCRIPTION
YearMonthPattern	Defines the overall format of the result string.
MonthNames	Defines the localized month names that can appear in the result string.

The following example uses the "y" format specifier to display a date and time value.

```
DateTime date1 = new DateTime(2008, 4, 10, 6, 30, 0);
Console.WriteLine(date1.ToString("Y",
    CultureInfo.CreateSpecificCulture("en-US")));
// Displays April, 2008
Console.WriteLine(date1.ToString("y",
    CultureInfo.CreateSpecificCulture("af-ZA")));
// Displays April 2008
```

```
Dim date1 As Date = #4/10/2008 6:30AM#
Console.WriteLine(date1.ToString("Y", CultureInfo.CreateSpecificCulture("en-US")))
' Displays April, 2008
Console.WriteLine(date1.ToString("y", CultureInfo.CreateSpecificCulture("af-ZA")))
' Displays April 2008
```

[Back to table](#)

Control Panel settings

In Windows, the settings in the **Regional and Language Options** item in Control Panel influence the result string produced by a formatting operation. These settings are used to initialize the [DateTimeFormatInfo](#) object associated with the current culture, which provides values used to govern formatting. Computers that use different settings generate different result strings.

In addition, if you use the [CultureInfo\(String\)](#) constructor to instantiate a new [CultureInfo](#) object that represents the same culture as the current system culture, any customizations established by the **Regional and Language Options** item in Control Panel will be applied to the new [CultureInfo](#) object. You can use the [CultureInfo\(String, Boolean\)](#) constructor to create a [CultureInfo](#) object that does not reflect a system's customizations.

DateTimeFormatInfo properties

Formatting is influenced by properties of the current [DateTimeFormatInfo](#) object, which is provided implicitly by the current culture or explicitly by the [IFormatProvider](#) parameter of the method that invokes formatting. For the [IFormatProvider](#) parameter, your application should specify a [CultureInfo](#) object, which represents a culture, or a [DateTimeFormatInfo](#) object, which represents a particular culture's date and time formatting conventions. Many of the standard date and time format specifiers are aliases for formatting patterns defined by properties of the current [DateTimeFormatInfo](#) object. Your application can change the result produced by some standard date and time format specifiers by changing the corresponding date and time format patterns of the corresponding [DateTimeFormatInfo](#) property.

See also

- [System.DateTime](#)
- [System.DateTimeOffset](#)
- [Formatting Types](#)
- [Custom Date and Time Format Strings](#)
- [Sample: .NET Core WinForms Formatting Utility \(C#\)](#)
- [Sample: .NET Core WinForms Formatting Utility \(Visual Basic\)](#)

Custom date and time format strings

9/20/2022 • 57 minutes to read • [Edit Online](#)

A date and time format string defines the text representation of a [DateTime](#) or [DateTimeOffset](#) value that results from a formatting operation. It can also define the representation of a date and time value that is required in a parsing operation in order to successfully convert the string to a date and time. A custom format string consists of one or more custom date and time format specifiers. Any string that is not a [standard date and time format string](#) is interpreted as a custom date and time format string.

TIP

You can download the [Formatting Utility](#), a .NET Core Windows Forms application that lets you apply format strings to either numeric or date and time values and displays the result string. Source code is available for [C#](#) and [Visual Basic](#).

Custom date and time format strings can be used with both [DateTime](#) and [DateTimeOffset](#) values.

NOTE

Some of the C# examples in this article run in the [Try.NET](#) inline code runner and playground. Select the **Run** button to run an example in an interactive window. Once you execute the code, you can modify it and run the modified code by selecting **Run** again. The modified code either runs in the interactive window or, if compilation fails, the interactive window displays all C# compiler error messages.

The [local time zone](#) of the [Try.NET](#) inline code runner and playground is Coordinated Universal Time, or UTC. This may affect the behavior and the output of examples that illustrate the [DateTime](#), [DateTimeOffset](#), and [TimeZoneInfo](#) types and their members.

In formatting operations, custom date and time format strings can be used either with the `ToString` method of a date and time instance or with a method that supports composite formatting. The following example illustrates both uses.

```
DateTime thisDate1 = new DateTime(2011, 6, 10);
Console.WriteLine("Today is " + thisDate1.ToString("MMMM dd, yyyy") + ".");

DateTimeOffset thisDate2 = new DateTimeOffset(2011, 6, 10, 15, 24, 16,
                                              TimeSpan.Zero);
Console.WriteLine("The current date and time: {0:MM/dd/yy H:mm:ss zzz}",
                  thisDate2);
// The example displays the following output:
//   Today is June 10, 2011.
//   The current date and time: 06/10/11 15:24:16 +00:00
```

```
Dim thisDate1 As Date = #6/10/2011#
Console.WriteLine("Today is " + thisDate1.ToString("MMMM dd, yyyy") + ".") 

Dim thisDate2 As New DateTimeOffset(2011, 6, 10, 15, 24, 16, TimeSpan.Zero)
Console.WriteLine("The current date and time: {0:MM/dd/yy H:mm:ss zzz}",
                  thisDate2)
' The example displays the following output:
'   Today is June 10, 2011.
'   The current date and time: 06/10/11 15:24:16 +00:00
```

In parsing operations, custom date and time format strings can be used with the [DateTime.ParseExact](#), [DateTime.TryParseExact](#), [DateTimeOffset.ParseExact](#), and [DateTimeOffset.TryParseExact](#) methods. These methods require that an input string conforms exactly to a particular pattern for the parse operation to succeed. The following example illustrates a call to the [DateTimeOffset.ParseExact\(String, String, IFormatProvider\)](#) method to parse a date that must include a day, a month, and a two-digit year.

```
using System;
using System.Globalization;

public class Example1
{
    public static void Main()
    {
        string[] dateValues = { "30-12-2011", "12-30-2011",
                               "30-12-11", "12-30-11" };
        string pattern = "MM-dd-yy";
        DateTime parsedDate;

        foreach (var dateValue in dateValues)
        {
            if (DateTime.TryParseExact(dateValue, pattern, null,
                                      DateTimeStyles.None, out parsedDate))
                Console.WriteLine("Converted '{0}' to {1:d}.",
                                  dateValue, parsedDate);
            else
                Console.WriteLine("Unable to convert '{0}' to a date and time.",
                                  dateValue);
        }
    }
}

// The example displays the following output:
//   Unable to convert '30-12-2011' to a date and time.
//   Unable to convert '12-30-2011' to a date and time.
//   Unable to convert '30-12-11' to a date and time.
//   Converted '12-30-11' to 12/30/2011.
```

```
Imports System.Globalization

Module Example
    Public Sub Main()
        Dim dateValues() As String = {"30-12-2011", "12-30-2011",
                                      "30-12-11", "12-30-11"}
        Dim pattern As String = "MM-dd-yy"
        Dim parsedDate As Date

        For Each dateValue As String In dateValues
            If DateTime.TryParseExact(dateValue, pattern, Nothing,
                                      DateTimeStyles.None, parsedDate) Then
                Console.WriteLine("Converted '{0}' to {1:d}.",
                                  dateValue, parsedDate)
            Else
                Console.WriteLine("Unable to convert '{0}' to a date and time.",
                                  dateValue)
            End If
        Next
    End Sub
End Module

' The example displays the following output:
'   Unable to convert '30-12-2011' to a date and time.
'   Unable to convert '12-30-2011' to a date and time.
'   Unable to convert '30-12-11' to a date and time.
'   Converted '12-30-11' to 12/30/2011.
```

The following table describes the custom date and time format specifiers and displays a result string produced by each format specifier. By default, result strings reflect the formatting conventions of the en-US culture. If a particular format specifier produces a localized result string, the example also notes the culture to which the result string applies. For more information about using custom date and time format strings, see the [Notes](#) section.

FORMAT SPECIFIER	DESCRIPTION	EXAMPLES
"d"	The day of the month, from 1 through 31. More information: The "d" Custom Format Specifier .	2009-06-01T13:45:30 -> 1 2009-06-15T13:45:30 -> 15
"dd"	The day of the month, from 01 through 31. More information: The "dd" Custom Format Specifier .	2009-06-01T13:45:30 -> 01 2009-06-15T13:45:30 -> 15
"ddd"	The abbreviated name of the day of the week. More information: The "ddd" Custom Format Specifier .	2009-06-15T13:45:30 -> Mon (en-US) 2009-06-15T13:45:30 -> Пн (ru-RU) 2009-06-15T13:45:30 -> lun. (fr-FR)
"dddd"	The full name of the day of the week. More information: The "dddd" Custom Format Specifier .	2009-06-15T13:45:30 -> Monday (en-US) 2009-06-15T13:45:30 -> понедельник (ru-RU) 2009-06-15T13:45:30 -> lundi (fr-FR)
"f"	The tenths of a second in a date and time value. More information: The "f" Custom Format Specifier .	2009-06-15T13:45:30.6170000 -> 6 2009-06-15T13:45:30.05 -> 0
"ff"	The hundredths of a second in a date and time value. More information: The "ff" Custom Format Specifier .	2009-06-15T13:45:30.6170000 -> 61 2009-06-15T13:45:30.0050000 -> 00
"fff"	The milliseconds in a date and time value. More information: The "fff" Custom Format Specifier .	6/15/2009 13:45:30.617 -> 617 6/15/2009 13:45:30.0005 -> 000
"ffff"	The ten thousandths of a second in a date and time value. More information: The "ffff" Custom Format Specifier .	2009-06-15T13:45:30.6175000 -> 6175 2009-06-15T13:45:30.0000500 -> 0000

FORMAT SPECIFIER	DESCRIPTION	EXAMPLES
"ffff"	The hundred thousandths of a second in a date and time value. More information: The "ffff" Custom Format Specifier .	2009-06-15T13:45:30.6175400 -> 61754 6/15/2009 13:45:30.000005 -> 00000
"ffffff"	The millionths of a second in a date and time value. More information: The "ffffff" Custom Format Specifier .	2009-06-15T13:45:30.6175420 -> 617542 2009-06-15T13:45:30.0000005 -> 000000
"fffffff"	The ten millionths of a second in a date and time value. More information: The "fffffff" Custom Format Specifier .	2009-06-15T13:45:30.6175425 -> 6175425 2009-06-15T13:45:30.0001150 -> 0001150
"F"	If non-zero, the tenths of a second in a date and time value. More information: The "F" Custom Format Specifier .	2009-06-15T13:45:30.6170000 -> 6 2009-06-15T13:45:30.0500000 -> (no output)
"FF"	If non-zero, the hundredths of a second in a date and time value. More information: The "FF" Custom Format Specifier .	2009-06-15T13:45:30.6170000 -> 61 2009-06-15T13:45:30.0050000 -> (no output)
"FFF"	If non-zero, the milliseconds in a date and time value. More information: The "FFF" Custom Format Specifier .	2009-06-15T13:45:30.6170000 -> 617 2009-06-15T13:45:30.0005000 -> (no output)
"FFFF"	If non-zero, the ten thousandths of a second in a date and time value. More information: The "FFFF" Custom Format Specifier .	2009-06-15T13:45:30.5275000 -> 5275 2009-06-15T13:45:30.0000500 -> (no output)
"FFFFF"	If non-zero, the hundred thousandths of a second in a date and time value. More information: The "FFFFF" Custom Format Specifier .	2009-06-15T13:45:30.6175400 -> 61754 2009-06-15T13:45:30.0000050 -> (no output)
"FFFFFF"	If non-zero, the millionths of a second in a date and time value. More information: The "FFFFFF" Custom Format Specifier .	2009-06-15T13:45:30.6175420 -> 617542 2009-06-15T13:45:30.0000005 -> (no output)

FORMAT SPECIFIER	DESCRIPTION	EXAMPLES
"FFFFFF"	If non-zero, the ten millionths of a second in a date and time value. More information: The "FFFFFF" Custom Format Specifier .	2009-06-15T13:45:30.6175425 -> 6175425 2009-06-15T13:45:30.0001150 -> 000115
"g", "gg"	The period or era. More information: The "g" or "gg" Custom Format Specifier .	2009-06-15T13:45:30.6170000 -> A.D.
"h"	The hour, using a 12-hour clock from 1 to 12. More information: The "h" Custom Format Specifier .	2009-06-15T01:45:30 -> 1 2009-06-15T13:45:30 -> 1
"hh"	The hour, using a 12-hour clock from 01 to 12. More information: The "hh" Custom Format Specifier .	2009-06-15T01:45:30 -> 01 2009-06-15T13:45:30 -> 01
"H"	The hour, using a 24-hour clock from 0 to 23. More information: The "H" Custom Format Specifier .	2009-06-15T01:45:30 -> 1 2009-06-15T13:45:30 -> 13
"HH"	The hour, using a 24-hour clock from 00 to 23. More information: The "HH" Custom Format Specifier .	2009-06-15T01:45:30 -> 01 2009-06-15T13:45:30 -> 13
"K"	Time zone information. More information: The "K" Custom Format Specifier .	With DateTime values: 2009-06-15T13:45:30, Kind Unspecified -> 2009-06-15T13:45:30, Kind Utc -> Z 2009-06-15T13:45:30, Kind Local -> -07:00 (depends on local computer settings) With DateTimeOffset values: 2009-06-15T01:45:30-07:00 --> -07:00 2009-06-15T08:45:30+00:00 --> +00:00
"m"	The minute, from 0 through 59. More information: The "m" Custom Format Specifier .	2009-06-15T01:09:30 -> 9 2009-06-15T13:29:30 -> 29

FORMAT SPECIFIER	DESCRIPTION	EXAMPLES
"mm"	The minute, from 00 through 59. More information: The "mm" Custom Format Specifier .	2009-06-15T01:09:30 -> 09 2009-06-15T01:45:30 -> 45
"M"	The month, from 1 through 12. More information: The "M" Custom Format Specifier .	2009-06-15T13:45:30 -> 6
"MM"	The month, from 01 through 12. More information: The "MM" Custom Format Specifier .	2009-06-15T13:45:30 -> 06
"MMM"	The abbreviated name of the month. More information: The "MMM" Custom Format Specifier .	2009-06-15T13:45:30 -> Jun (en-US) 2009-06-15T13:45:30 -> juin (fr-FR) 2009-06-15T13:45:30 -> Jun (zu-ZA)
"MMMM"	The full name of the month. More information: The "MMMM" Custom Format Specifier .	2009-06-15T13:45:30 -> June (en-US) 2009-06-15T13:45:30 -> juni (da-DK) 2009-06-15T13:45:30 -> uJuni (zu-ZA)
"s"	The second, from 0 through 59. More information: The "s" Custom Format Specifier .	2009-06-15T13:45:09 -> 9
"ss"	The second, from 00 through 59. More information: The "ss" Custom Format Specifier .	2009-06-15T13:45:09 -> 09
"t"	The first character of the AM/PM designator. More information: The "t" Custom Format Specifier .	2009-06-15T13:45:30 -> P (en-US) 2009-06-15T13:45:30 -> 午 (ja-JP) 2009-06-15T13:45:30 -> (fr-FR)
"tt"	The AM/PM designator. More information: The "tt" Custom Format Specifier .	2009-06-15T13:45:30 -> PM (en-US) 2009-06-15T13:45:30 -> 午後 (ja-JP) 2009-06-15T13:45:30 -> (fr-FR)

FORMAT SPECIFIER	DESCRIPTION	EXAMPLES
"y"	<p>The year, from 0 to 99.</p> <p>More information: The "y" Custom Format Specifier.</p>	0001-01-01T00:00:00 -> 1 0900-01-01T00:00:00 -> 0 1900-01-01T00:00:00 -> 0 2009-06-15T13:45:30 -> 9 2019-06-15T13:45:30 -> 19
"yy"	<p>The year, from 00 to 99.</p> <p>More information: The "yy" Custom Format Specifier.</p>	0001-01-01T00:00:00 -> 01 0900-01-01T00:00:00 -> 00 1900-01-01T00:00:00 -> 00 2019-06-15T13:45:30 -> 19
"yyy"	<p>The year, with a minimum of three digits.</p> <p>More information: The "yyy" Custom Format Specifier.</p>	0001-01-01T00:00:00 -> 001 0900-01-01T00:00:00 -> 900 1900-01-01T00:00:00 -> 1900 2009-06-15T13:45:30 -> 2009
"yyyy"	<p>The year as a four-digit number.</p> <p>More information: The "yyyy" Custom Format Specifier.</p>	0001-01-01T00:00:00 -> 0001 0900-01-01T00:00:00 -> 0900 1900-01-01T00:00:00 -> 1900 2009-06-15T13:45:30 -> 2009
"yyyyy"	<p>The year as a five-digit number.</p> <p>More information: The "yyyyy" Custom Format Specifier.</p>	0001-01-01T00:00:00 -> 00001 2009-06-15T13:45:30 -> 02009
"z"	<p>Hours offset from UTC, with no leading zeros.</p> <p>More information: The "z" Custom Format Specifier.</p>	2009-06-15T13:45:30-07:00 -> -7
"zz"	<p>Hours offset from UTC, with a leading zero for a single-digit value.</p> <p>More information: The "zz" Custom Format Specifier.</p>	2009-06-15T13:45:30-07:00 -> -07
"zzz"	<p>Hours and minutes offset from UTC.</p> <p>More information: The "zzz" Custom Format Specifier.</p>	2009-06-15T13:45:30-07:00 -> -07:00

FORMAT SPECIFIER	DESCRIPTION	EXAMPLES
"."	The time separator. More information: The ":" Custom Format Specifier .	2009-06-15T13:45:30 -> : (en-US) 2009-06-15T13:45:30 -> . (it-IT) 2009-06-15T13:45:30 -> : (ja-JP)
"/"	The date separator. More Information: The "/" Custom Format Specifier .	2009-06-15T13:45:30 -> / (en-US) 2009-06-15T13:45:30 -> - (ar-DZ) 2009-06-15T13:45:30 -> . (tr-TR)
"string" 'string'	Literal string delimiter. More information: Character literals .	2009-06-15T13:45:30 ("arr:" h:m t) -> arr: 1:45 P 2009-06-15T13:45:30 ('arr:' h:m t) -> arr: 1:45 P
%	Defines the following character as a custom format specifier. More information: Using Single Custom Format Specifiers .	2009-06-15T13:45:30 (%h) -> 1
\	The escape character. More information: Character literals and Using the Escape Character .	2009-06-15T13:45:30 (h \h) -> 1 h
Any other character	The character is copied to the result string unchanged. More information: Character literals .	2009-06-15T01:45:30 (arr hh:mm t) -> arr 01:45 A

The following sections provide additional information about each custom date and time format specifier. Unless otherwise noted, each specifier produces an identical string representation regardless of whether it's used with a [DateTime](#) value or a [DateTimeOffset](#) value.

Day "d" format specifier

The "d" custom format specifier

The "d" custom format specifier represents the day of the month as a number from 1 through 31. A single-digit day is formatted without a leading zero.

If the "d" format specifier is used without other custom format specifiers, it's interpreted as the "d" standard date and time format specifier. For more information about using a single format specifier, see [Using Single Custom Format Specifiers](#) later in this article.

The following example includes the "d" custom format specifier in several format strings.

```

DateTime date1 = new DateTime(2008, 8, 29, 19, 27, 15);

Console.WriteLine(date1.ToString("d, M",
                               CultureInfo.InvariantCulture));
// Displays 29, 8

Console.WriteLine(date1.ToString("d MMMM",
                               CultureInfo.CreateSpecificCulture("en-US")));
// Displays 29 August
Console.WriteLine(date1.ToString("d MMMM",
                               CultureInfo.CreateSpecificCulture("es-MX")));
// Displays 29 agosto

```

```

Dim date1 As Date = #08/29/2008 7:27:15PM#

Console.WriteLine(date1.ToString("d, M",
                               CultureInfo.InvariantCulture))
' Displays 29, 8

Console.WriteLine(date1.ToString("d MMMM",
                               CultureInfo.CreateSpecificCulture("en-US")))
' Displays 29 August
Console.WriteLine(date1.ToString("d MMMM",
                               CultureInfo.CreateSpecificCulture("es-MX")))
' Displays 29 agosto

```

[Back to table](#)

The "dd" custom format specifier

The "dd" custom format string represents the day of the month as a number from 01 through 31. A single-digit day is formatted with a leading zero.

The following example includes the "dd" custom format specifier in a custom format string.

```

DateTime date1 = new DateTime(2008, 1, 2, 6, 30, 15);

Console.WriteLine(date1.ToString("dd, MM",
                               CultureInfo.InvariantCulture));
// 02, 01

```

```

Dim date1 As Date = #1/2/2008 6:30:15AM#

Console.WriteLine(date1.ToString("dd, MM",
                               CultureInfo.InvariantCulture))
' 02, 01

```

[Back to table](#)

The "ddd" custom format specifier

The "ddd" custom format specifier represents the abbreviated name of the day of the week. The localized abbreviated name of the day of the week is retrieved from the [DateTimeFormatInfo.AbbreviatedDayNames](#) property of the current or specified culture.

The following example includes the "ddd" custom format specifier in a custom format string.

```

DateTime date1 = new DateTime(2008, 8, 29, 19, 27, 15);

Console.WriteLine(date1.ToString("ddd d MMM",
                               CultureInfo.CreateSpecificCulture("en-US")));
// Displays Fri 29 Aug
Console.WriteLine(date1.ToString("ddd d MMM",
                               CultureInfo.CreateSpecificCulture("fr-FR")));
// Displays ven. 29 août

```

```

Dim date1 As Date = #08/29/2008 7:27:15PM#

Console.WriteLine(date1.ToString("ddd d MMM",
                               CultureInfo.CreateSpecificCulture("en-US")))
' Displays Fri 29 Aug
Console.WriteLine(date1.ToString("ddd d MMM",
                               CultureInfo.CreateSpecificCulture("fr-FR")))
' Displays ven. 29 août

```

[Back to table](#)

The "dddd" custom format specifier

The "dddd" custom format specifier (plus any number of additional "d" specifiers) represents the full name of the day of the week. The localized name of the day of the week is retrieved from the [DateTimeFormatInfo.DayNames](#) property of the current or specified culture.

The following example includes the "dddd" custom format specifier in a custom format string.

```

DateTime date1 = new DateTime(2008, 8, 29, 19, 27, 15);

Console.WriteLine(date1.ToString("dddd dd MMMM",
                               CultureInfo.CreateSpecificCulture("en-US")));
// Displays Friday 29 August
Console.WriteLine(date1.ToString("dddd dd MMMM",
                               CultureInfo.CreateSpecificCulture("it-IT")));
// Displays venerdì 29 agosto

```

```

Dim date1 As Date = #08/29/2008 7:27:15PM#

Console.WriteLine(date1.ToString("dddd dd MMMM",
                               CultureInfo.CreateSpecificCulture("en-US")))
' Displays Friday 29 August
Console.WriteLine(date1.ToString("dddd dd MMMM",
                               CultureInfo.CreateSpecificCulture("it-IT")))
' Displays venerdì 29 agosto

```

[Back to table](#)

Lowercase seconds "f" fraction specifier

The "f" custom format specifier

The "f" custom format specifier represents the most significant digit of the seconds fraction; that is, it represents the tenths of a second in a date and time value.

If the "f" format specifier is used without other format specifiers, it's interpreted as the "f" standard date and time format specifier. For more information about using a single format specifier, see [Using Single Custom Format Specifiers](#) later in this article.

When you use "f" format specifiers as part of a format string supplied to the [ParseExact](#), [TryParseExact](#), [ParseExact](#), or [TryParseExact](#) method, the number of "f" format specifiers indicates the number of most significant digits of the seconds fraction that must be present to successfully parse the string.

The following example includes the "f" custom format specifier in a custom format string.

```
DateTime date1 = new DateTime(2008, 8, 29, 19, 27, 15, 18);
CultureInfo ci = CultureInfo.InvariantCulture;

Console.WriteLine(date1.ToString("hh:mm:ss.f", ci));
// Displays 07:27:15.0
Console.WriteLine(date1.ToString("hh:mm:ss.F", ci));
// Displays 07:27:15
Console.WriteLine(date1.ToString("hh:mm:ss.ff", ci));
// Displays 07:27:15.01
Console.WriteLine(date1.ToString("hh:mm:ss.FF", ci));
// Displays 07:27:15.01
Console.WriteLine(date1.ToString("hh:mm:ss.fff", ci));
// Displays 07:27:15.018
Console.WriteLine(date1.ToString("hh:mm:ss.FFF", ci));
// Displays 07:27:15.018
```

```
Dim date1 As New Date(2008, 8, 29, 19, 27, 15, 018)
Dim ci As CultureInfo = CultureInfo.InvariantCulture

Console.WriteLine(date1.ToString("hh:mm:ss.f", ci))
' Displays 07:27:15.0
Console.WriteLine(date1.ToString("hh:mm:ss.F", ci))
' Displays 07:27:15
Console.WriteLine(date1.ToString("hh:mm:ss.ff", ci))
' Displays 07:27:15.01
Console.WriteLine(date1.ToString("hh:mm:ss.FF", ci))
' Displays 07:27:15.01
Console.WriteLine(date1.ToString("hh:mm:ss.fff", ci))
' Displays 07:27:15.018
Console.WriteLine(date1.ToString("hh:mm:ss.FFF", ci))
' Displays 07:27:15.018
```

[Back to table](#)

The "ff" custom format specifier

The "ff" custom format specifier represents the two most significant digits of the seconds fraction; that is, it represents the hundredths of a second in a date and time value.

The following example includes the "ff" custom format specifier in a custom format string.

```
DateTime date1 = new DateTime(2008, 8, 29, 19, 27, 15, 18);
CultureInfo ci = CultureInfo.InvariantCulture;

Console.WriteLine(date1.ToString("hh:mm:ss.f", ci));
// Displays 07:27:15.0
Console.WriteLine(date1.ToString("hh:mm:ss.F", ci));
// Displays 07:27:15
Console.WriteLine(date1.ToString("hh:mm:ss.ff", ci));
// Displays 07:27:15.01
Console.WriteLine(date1.ToString("hh:mm:ss.FF", ci));
// Displays 07:27:15.01
Console.WriteLine(date1.ToString("hh:mm:ss.fff", ci));
// Displays 07:27:15.018
Console.WriteLine(date1.ToString("hh:mm:ss.FFF", ci));
// Displays 07:27:15.018
```

```

Dim date1 As New Date(2008, 8, 29, 19, 27, 15, 018)
Dim ci As CultureInfo = CultureInfo.InvariantCulture

Console.WriteLine(date1.ToString("hh:mm:ss.f", ci))
' Displays 07:27:15.0
Console.WriteLine(date1.ToString("hh:mm:ss.F", ci))
' Displays 07:27:15
Console.WriteLine(date1.ToString("hh:mm:ss.ff", ci))
' Displays 07:27:15.01
Console.WriteLine(date1.ToString("hh:mm:ss.FF", ci))
' Displays 07:27:15.01
Console.WriteLine(date1.ToString("hh:mm:ss.fff", ci))
' Displays 07:27:15.018
Console.WriteLine(date1.ToString("hh:mm:ss.FFF", ci))
' Displays 07:27:15.018

```

[Back to table](#)

The "fff" custom format specifier

The "fff" custom format specifier represents the three most significant digits of the seconds fraction; that is, it represents the milliseconds in a date and time value.

The following example includes the "fff" custom format specifier in a custom format string.

```

DateTime date1 = new DateTime(2008, 8, 29, 19, 27, 15, 18);
CultureInfo ci = CultureInfo.InvariantCulture;

Console.WriteLine(date1.ToString("hh:mm:ss.f", ci));
// Displays 07:27:15.0
Console.WriteLine(date1.ToString("hh:mm:ss.F", ci));
// Displays 07:27:15
Console.WriteLine(date1.ToString("hh:mm:ss.ff", ci));
// Displays 07:27:15.01
Console.WriteLine(date1.ToString("hh:mm:ss.FF", ci));
// Displays 07:27:15.01
Console.WriteLine(date1.ToString("hh:mm:ss.fff", ci));
// Displays 07:27:15.018
Console.WriteLine(date1.ToString("hh:mm:ss.FFF", ci));
// Displays 07:27:15.018

```

```

Dim date1 As New Date(2008, 8, 29, 19, 27, 15, 018)
Dim ci As CultureInfo = CultureInfo.InvariantCulture

Console.WriteLine(date1.ToString("hh:mm:ss.f", ci))
' Displays 07:27:15.0
Console.WriteLine(date1.ToString("hh:mm:ss.F", ci))
' Displays 07:27:15
Console.WriteLine(date1.ToString("hh:mm:ss.ff", ci))
' Displays 07:27:15.01
Console.WriteLine(date1.ToString("hh:mm:ss.FF", ci))
' Displays 07:27:15.01
Console.WriteLine(date1.ToString("hh:mm:ss.fff", ci))
' Displays 07:27:15.018
Console.WriteLine(date1.ToString("hh:mm:ss.FFF", ci))
' Displays 07:27:15.018

```

[Back to table](#)

The "ffff" custom format specifier

The "ffff" custom format specifier represents the four most significant digits of the seconds fraction; that is, it represents the ten thousandths of a second in a date and time value.

Although it's possible to display the ten thousandths of a second component of a time value, that value may not be meaningful. The precision of date and time values depends on the resolution of the system clock. On the Windows NT version 3.5 (and later) and Windows Vista operating systems, the clock's resolution is approximately 10-15 milliseconds.

[Back to table](#)

The "fffff" custom format specifier

The "fffff" custom format specifier represents the five most significant digits of the seconds fraction; that is, it represents the hundred thousandths of a second in a date and time value.

Although it's possible to display the hundred thousandths of a second component of a time value, that value may not be meaningful. The precision of date and time values depends on the resolution of the system clock. On the Windows NT 3.5 (and later) and Windows Vista operating systems, the clock's resolution is approximately 10-15 milliseconds.

[Back to table](#)

The "ffffff" custom format specifier

The "ffffff" custom format specifier represents the six most significant digits of the seconds fraction; that is, it represents the millionths of a second in a date and time value.

Although it's possible to display the millionths of a second component of a time value, that value may not be meaningful. The precision of date and time values depends on the resolution of the system clock. On the Windows NT 3.5 (and later) and Windows Vista operating systems, the clock's resolution is approximately 10-15 milliseconds.

[Back to table](#)

The "fffffff" custom format specifier

The "fffffff" custom format specifier represents the seven most significant digits of the seconds fraction; that is, it represents the ten millionths of a second in a date and time value.

Although it's possible to display the ten millionths of a second component of a time value, that value may not be meaningful. The precision of date and time values depends on the resolution of the system clock. On the Windows NT 3.5 (and later) and Windows Vista operating systems, the clock's resolution is approximately 10-15 milliseconds.

[Back to table](#)

Uppercase seconds "F" fraction specifier

The "F" custom format specifier

The "F" custom format specifier represents the most significant digit of the seconds fraction; that is, it represents the tenths of a second in a date and time value. Nothing is displayed if the digit is zero, and the decimal point that follows the number of seconds is also not displayed.

If the "F" format specifier is used without other format specifiers, it's interpreted as the "F" standard date and time format specifier. For more information about using a single format specifier, see [Using Single Custom Format Specifiers](#) later in this article.

The number of "F" format specifiers used with the `ParseExact`, `TryParseExact`, `ParseExact`, or `TryParseExact` method indicates the maximum number of most significant digits of the seconds fraction that can be present to successfully parse the string.

The following example includes the "F" custom format specifier in a custom format string.

```

DateTime date1 = new DateTime(2008, 8, 29, 19, 27, 15, 18);
CultureInfo ci = CultureInfo.InvariantCulture;

Console.WriteLine(date1.ToString("hh:mm:ss.f", ci));
// Displays 07:27:15.0
Console.WriteLine(date1.ToString("hh:mm:ss.F", ci));
// Displays 07:27:15
Console.WriteLine(date1.ToString("hh:mm:ss.ff", ci));
// Displays 07:27:15.01
Console.WriteLine(date1.ToString("hh:mm:ss.FF", ci));
// Displays 07:27:15.011
Console.WriteLine(date1.ToString("hh:mm:ss.fff", ci));
// Displays 07:27:15.018
Console.WriteLine(date1.ToString("hh:mm:ss.FFF", ci));
// Displays 07:27:15.018

```

```

Dim date1 As New Date(2008, 8, 29, 19, 27, 15, 018)
Dim ci As CultureInfo = CultureInfo.InvariantCulture

Console.WriteLine(date1.ToString("hh:mm:ss.f", ci))
' Displays 07:27:15.0
Console.WriteLine(date1.ToString("hh:mm:ss.F", ci))
' Displays 07:27:15
Console.WriteLine(date1.ToString("hh:mm:ss.ff", ci));
' Displays 07:27:15.01
Console.WriteLine(date1.ToString("hh:mm:ss.FF", ci));
' Displays 07:27:15.011
Console.WriteLine(date1.ToString("hh:mm:ss.fff", ci));
' Displays 07:27:15.018
Console.WriteLine(date1.ToString("hh:mm:ss.FFF", ci));
' Displays 07:27:15.018

```

[Back to table](#)

The "FF" custom format specifier

The "FF" custom format specifier represents the two most significant digits of the seconds fraction; that is, it represents the hundredths of a second in a date and time value. Trailing zeros aren't displayed. Nothing is displayed if the two significant digits are zero, and in that case the decimal point that follows the number of seconds is also not displayed.

The following example includes the "FF" custom format specifier in a custom format string.

```

DateTime date1 = new DateTime(2008, 8, 29, 19, 27, 15, 18);
CultureInfo ci = CultureInfo.InvariantCulture;

Console.WriteLine(date1.ToString("hh:mm:ss.f", ci));
// Displays 07:27:15.0
Console.WriteLine(date1.ToString("hh:mm:ss.F", ci));
// Displays 07:27:15
Console.WriteLine(date1.ToString("hh:mm:ss.ff", ci));
// Displays 07:27:15.01
Console.WriteLine(date1.ToString("hh:mm:ss.FF", ci));
// Displays 07:27:15.011
Console.WriteLine(date1.ToString("hh:mm:ss.fff", ci));
// Displays 07:27:15.018
Console.WriteLine(date1.ToString("hh:mm:ss.FFF", ci));
// Displays 07:27:15.018

```

```

Dim date1 As New Date(2008, 8, 29, 19, 27, 15, 018)
Dim ci As CultureInfo = CultureInfo.InvariantCulture

Console.WriteLine(date1.ToString("hh:mm:ss.f", ci))
' Displays 07:27:15.0
Console.WriteLine(date1.ToString("hh:mm:ss.F", ci))
' Displays 07:27:15
Console.WriteLine(date1.ToString("hh:mm:ss.ff", ci))
' Displays 07:27:15.01
Console.WriteLine(date1.ToString("hh:mm:ss.FF", ci))
' Displays 07:27:15.01
Console.WriteLine(date1.ToString("hh:mm:ss.fff", ci))
' Displays 07:27:15.018
Console.WriteLine(date1.ToString("hh:mm:ss.FFF", ci))
' Displays 07:27:15.018

```

[Back to table](#)

The "FFF" custom format specifier

The "FFF" custom format specifier represents the three most significant digits of the seconds fraction; that is, it represents the milliseconds in a date and time value. Trailing zeros aren't displayed. Nothing is displayed if the three significant digits are zero, and in that case the decimal point that follows the number of seconds is also not displayed.

The following example includes the "FFF" custom format specifier in a custom format string.

```

DateTime date1 = new DateTime(2008, 8, 29, 19, 27, 15, 18);
CultureInfo ci = CultureInfo.InvariantCulture;

Console.WriteLine(date1.ToString("hh:mm:ss.f", ci));
// Displays 07:27:15.0
Console.WriteLine(date1.ToString("hh:mm:ss.F", ci));
// Displays 07:27:15
Console.WriteLine(date1.ToString("hh:mm:ss.ff", ci));
// Displays 07:27:15.01
Console.WriteLine(date1.ToString("hh:mm:ss.FF", ci));
// Displays 07:27:15.01
Console.WriteLine(date1.ToString("hh:mm:ss.fff", ci));
// Displays 07:27:15.018
Console.WriteLine(date1.ToString("hh:mm:ss.FFF", ci));
// Displays 07:27:15.018

```

```

Dim date1 As New Date(2008, 8, 29, 19, 27, 15, 018)
Dim ci As CultureInfo = CultureInfo.InvariantCulture

Console.WriteLine(date1.ToString("hh:mm:ss.f", ci))
' Displays 07:27:15.0
Console.WriteLine(date1.ToString("hh:mm:ss.F", ci))
' Displays 07:27:15
Console.WriteLine(date1.ToString("hh:mm:ss.ff", ci))
' Displays 07:27:15.01
Console.WriteLine(date1.ToString("hh:mm:ss.FF", ci))
' Displays 07:27:15.01
Console.WriteLine(date1.ToString("hh:mm:ss.fff", ci))
' Displays 07:27:15.018
Console.WriteLine(date1.ToString("hh:mm:ss.FFF", ci))
' Displays 07:27:15.018

```

[Back to table](#)

The "FFFF" custom format specifier

The "FFFF" custom format specifier represents the four most significant digits of the seconds fraction; that is, it represents the ten thousandths of a second in a date and time value. Trailing zeros aren't displayed. Nothing is displayed if the four significant digits are zero, and in that case the decimal point that follows the number of seconds is also not displayed.

Although it's possible to display the ten thousandths of a second component of a time value, that value may not be meaningful. The precision of date and time values depends on the resolution of the system clock. On the Windows NT 3.5 (and later) and Windows Vista operating systems, the clock's resolution is approximately 10-15 milliseconds.

[Back to table](#)

The "FFFFF" custom format specifier

The "FFFFF" custom format specifier represents the five most significant digits of the seconds fraction; that is, it represents the hundred thousandths of a second in a date and time value. Trailing zeros aren't displayed. Nothing is displayed if the five significant digits are zero, and in that case the decimal point that follows the number of seconds is also not displayed.

Although it's possible to display the hundred thousandths of a second component of a time value, that value may not be meaningful. The precision of date and time values depends on the resolution of the system clock. On the Windows NT 3.5 (and later) and Windows Vista operating systems, the clock's resolution is approximately 10-15 milliseconds.

[Back to table](#)

The "FFFFFF" custom format specifier

The "FFFFFF" custom format specifier represents the six most significant digits of the seconds fraction; that is, it represents the millionths of a second in a date and time value. Trailing zeros aren't displayed. Nothing is displayed if the six significant digits are zero, and in that case the decimal point that follows the number of seconds is also not displayed.

Although it's possible to display the millionths of a second component of a time value, that value may not be meaningful. The precision of date and time values depends on the resolution of the system clock. On the Windows NT 3.5 (and later) and Windows Vista operating systems, the clock's resolution is approximately 10-15 milliseconds.

[Back to table](#)

The "FFFFFFF" custom format specifier

The "FFFFFFF" custom format specifier represents the seven most significant digits of the seconds fraction; that is, it represents the ten millionths of a second in a date and time value. Trailing zeros aren't displayed. Nothing is displayed if the seven significant digits are zero, and in that case the decimal point that follows the number of seconds is also not displayed.

Although it's possible to display the ten millionths of a second component of a time value, that value may not be meaningful. The precision of date and time values depends on the resolution of the system clock. On the Windows NT 3.5 (and later) and Windows Vista operating systems, the clock's resolution is approximately 10-15 milliseconds.

[Back to table](#)

Era "g" format specifier

The "g" or "gg" custom format specifier

The "g" or "gg" custom format specifiers (plus any number of additional "g" specifiers) represents the period or era, such as A.D. The formatting operation ignores this specifier if the date to be formatted doesn't have an

associated period or era string.

If the "g" format specifier is used without other custom format specifiers, it's interpreted as the "g" standard date and time format specifier. For more information about using a single format specifier, see [Using Single Custom Format Specifiers](#) later in this article.

The following example includes the "g" custom format specifier in a custom format string.

```
DateTime date1 = new DateTime(70, 08, 04);

Console.WriteLine(date1.ToString("MM/dd/yyyy g",
    CultureInfo.InvariantCulture));
// Displays 08/04/0070 A.D.

Console.WriteLine(date1.ToString("MM/dd/yyyy g",
    CultureInfo.CreateSpecificCulture("fr-FR")));
// Displays 08/04/0070 ap. J.-C.
```

```
Dim date1 As Date = #08/04/0070#

Console.WriteLine(date1.ToString("MM/dd/yyyy g", _
    CultureInfo.InvariantCulture))
' Displays 08/04/0070 A.D.

Console.WriteLine(date1.ToString("MM/dd/yyyy g", _
    CultureInfo.CreateSpecificCulture("fr-FR")))
' Displays 08/04/0070 ap. J.-C.
```

[Back to table](#)

Lowercase hour "h" format specifier

The "h" custom format specifier

The "h" custom format specifier represents the hour as a number from 1 through 12; that is, the hour is represented by a 12-hour clock that counts the whole hours since midnight or noon. A particular hour after midnight is indistinguishable from the same hour after noon. The hour is not rounded, and a single-digit hour is formatted without a leading zero. For example, given a time of 5:43 in the morning or afternoon, this custom format specifier displays "5".

If the "h" format specifier is used without other custom format specifiers, it's interpreted as a standard date and time format specifier and throws a [FormatException](#). For more information about using a single format specifier, see [Using Single Custom Format Specifiers](#) later in this article.

The following example includes the "h" custom format specifier in a custom format string.

```
DateTime date1;
date1 = new DateTime(2008, 1, 1, 18, 9, 1);
Console.WriteLine(date1.ToString("h:m:s.F t",
    CultureInfo.InvariantCulture));
// Displays 6:9:1 P

Console.WriteLine(date1.ToString("h:m:s.F t",
    CultureInfo.CreateSpecificCulture("el-GR")));
// Displays 6:9:1 μ

date1 = new DateTime(2008, 1, 1, 18, 9, 1, 500);
Console.WriteLine(date1.ToString("h:m:s.F t",
    CultureInfo.InvariantCulture));
// Displays 6:9:1.5 P

Console.WriteLine(date1.ToString("h:m:s.F t",
    CultureInfo.CreateSpecificCulture("el-GR")));
// Displays 6:9:1.5 μ
```

```

Dim date1 As Date
date1 = #6:09:01PM#
Console.WriteLine(date1.ToString("h:m:s.F t",
                               CultureInfo.InvariantCulture))
' Displays 6:9:1 P
Console.WriteLine(date1.ToString("h:m:s.F t",
                               CultureInfo.CreateSpecificCulture("el-GR")))
' Displays 6:9:1 μ
date1 = New Date(2008, 1, 1, 18, 9, 1, 500)
Console.WriteLine(date1.ToString("h:m:s.F t",
                               CultureInfo.InvariantCulture))
' Displays 6:9:1.5 P
Console.WriteLine(date1.ToString("h:m:s.F t",
                               CultureInfo.CreateSpecificCulture("el-GR")))
' Displays 6:9:1.5 μ

```

[Back to table](#)

The "hh" custom format specifier

The "hh" custom format specifier (plus any number of additional "h" specifiers) represents the hour as a number from 01 through 12; that is, the hour is represented by a 12-hour clock that counts the whole hours since midnight or noon. A particular hour after midnight is indistinguishable from the same hour after noon. The hour is not rounded, and a single-digit hour is formatted with a leading zero. For example, given a time of 5:43 in the morning or afternoon, this format specifier displays "05".

The following example includes the "hh" custom format specifier in a custom format string.

```

DateTime date1;
date1 = new DateTime(2008, 1, 1, 18, 9, 1);
Console.WriteLine(date1.ToString("hh:mm:ss tt",
                               CultureInfo.InvariantCulture));
// Displays 06:09:01 PM
Console.WriteLine(date1.ToString("hh:mm:ss tt",
                               CultureInfo.CreateSpecificCulture("hu-HU")));
// Displays 06:09:01 du.
date1 = new DateTime(2008, 1, 1, 18, 9, 1, 500);
Console.WriteLine(date1.ToString("hh:mm:ss.ff tt",
                               CultureInfo.InvariantCulture));
// Displays 06:09:01.50 PM
Console.WriteLine(date1.ToString("hh:mm:ss.ff tt",
                               CultureInfo.CreateSpecificCulture("hu-HU")));
// Displays 06:09:01.50 du.

```

```

Dim date1 As Date
date1 = #6:09:01PM#
Console.WriteLine(date1.ToString("hh:mm:ss tt",
                               CultureInfo.InvariantCulture))
' Displays 06:09:01 PM
Console.WriteLine(date1.ToString("hh:mm:ss tt",
                               CultureInfo.CreateSpecificCulture("hu-HU")))
' Displays 06:09:01 du.
date1 = New Date(2008, 1, 1, 18, 9, 1, 500)
Console.WriteLine(date1.ToString("hh:mm:ss.ff tt",
                               CultureInfo.InvariantCulture))
' Displays 06:09:01.50 PM
Console.WriteLine(date1.ToString("hh:mm:ss.ff tt",
                               CultureInfo.CreateSpecificCulture("hu-HU")))
' Displays 06:09:01.50 du.

```

[Back to table](#)

Uppercase hour "H" format specifier

The "H" custom format specifier

The "H" custom format specifier represents the hour as a number from 0 through 23; that is, the hour is represented by a zero-based 24-hour clock that counts the hours since midnight. A single-digit hour is formatted without a leading zero.

If the "H" format specifier is used without other custom format specifiers, it's interpreted as a standard date and time format specifier and throws a [FormatException](#). For more information about using a single format specifier, see [Using Single Custom Format Specifiers](#) later in this article.

The following example includes the "H" custom format specifier in a custom format string.

```
DateTime date1 = new DateTime(2008, 1, 1, 6, 9, 1);
Console.WriteLine(date1.ToString("H:mm:ss",
                               CultureInfo.InvariantCulture));
// Displays 6:09:01
```

```
Dim date1 As Date = #6:09:01AM#
Console.WriteLine(date1.ToString("H:mm:ss",
                               CultureInfo.InvariantCulture))
' Displays 6:09:01
```

[Back to table](#)

The "HH" custom format specifier

The "HH" custom format specifier (plus any number of additional "H" specifiers) represents the hour as a number from 00 through 23; that is, the hour is represented by a zero-based 24-hour clock that counts the hours since midnight. A single-digit hour is formatted with a leading zero.

The following example includes the "HH" custom format specifier in a custom format string.

```
DateTime date1 = new DateTime(2008, 1, 1, 6, 9, 1);
Console.WriteLine(date1.ToString("HH:mm:ss",
                               CultureInfo.InvariantCulture));
// Displays 06:09:01
```

```
Dim date1 As Date = #6:09:01AM#
Console.WriteLine(date1.ToString("HH:mm:ss",
                               CultureInfo.InvariantCulture))
' Displays 06:09:01
```

[Back to table](#)

Time zone "K" format specifier

The "K" custom format specifier

The "K" custom format specifier represents the time zone information of a date and time value. When this format specifier is used with [DateTime](#) values, the result string is defined by the value of the [DateTime.Kind](#) property:

- For the local time zone (a [DateTime.Kind](#) property value of [DateTimeKind.Local](#)), this specifier produces a result string containing the local offset from Coordinated Universal Time (UTC); for example, "-07:00".
- For a UTC time (a [DateTime.Kind](#) property value of [DateTimeKind.Utc](#)), the result string includes a "Z"

character to represent a UTC date.

- For a time from an unspecified time zone (a time whose `DateTime.Kind` property equals `DateTimeKind.Unspecified`), the result is equivalent to `String.Empty`.

For `DateTimeOffset` values, the "K" format specifier is equivalent to the "zzz" format specifier, and produces a result string containing the `DateTimeOffset` value's offset from UTC.

If the "K" format specifier is used without other custom format specifiers, it's interpreted as a standard date and time format specifier and throws a `FormatException`. For more information about using a single format specifier, see [Using Single Custom Format Specifiers](#) later in this article.

The following example displays the string that results from using the "K" custom format specifier with various `DateTime` and `DateTimeOffset` values on a system in the U.S. Pacific Time zone.

```
Console.WriteLine(DateTime.Now.ToString("%K"));
// Displays -07:00
Console.WriteLine(DateTime.UtcNow.ToString("%K"));
// Displays Z
Console.WriteLine("{0}",
    DateTime.SpecifyKind(DateTime.Now,
        DateTimeKind.Unspecified).ToString("%K"));
// Displays ''
Console.WriteLine(DateTimeOffset.Now.ToString("%K"));
// Displays -07:00
Console.WriteLine(DateTimeOffset.UtcNow.ToString("%K"));
// Displays +00:00
Console.WriteLine(new DateTimeOffset(2008, 5, 1, 6, 30, 0,
    new TimeSpan(5, 0, 0)).ToString("%K"));
// Displays +05:00
```

```
Console.WriteLine(Date.Now.ToString("%K"))
' Displays -07:00
Console.WriteLine(Date.UtcNow.ToString("%K"))
' Displays Z
Console.WriteLine("{0}", _
    Date.SpecifyKind(Date.Now, _
        DateTimeKind.Unspecified). _
    ToString("%K"))
' Displays ''
Console.WriteLine(DateTimeOffset.Now.ToString("%K"))
' Displays -07:00
Console.WriteLine(DateTimeOffset.UtcNow.ToString("%K"))
' Displays +00:00
Console.WriteLine(New DateTimeOffset(2008, 5, 1, 6, 30, 0, _
    New TimeSpan(5, 0, 0)). _
    ToString("%K"))
' Displays +05:00
```

[Back to table](#)

Minute "m" format specifier

The "m" custom format specifier

The "m" custom format specifier represents the minute as a number from 0 through 59. The minute represents whole minutes that have passed since the last hour. A single-digit minute is formatted without a leading zero.

If the "m" format specifier is used without other custom format specifiers, it's interpreted as the "m" standard date and time format specifier. For more information about using a single format specifier, see [Using Single Custom Format Specifiers](#) later in this article.

The following example includes the "m" custom format specifier in a custom format string.

```
DateTime date1;
date1 = new DateTime(2008, 1, 1, 18, 9, 1);
Console.WriteLine(date1.ToString("h:m:s.F t",
    CultureInfo.InvariantCulture));
// Displays 6:9:1 P
Console.WriteLine(date1.ToString("h:m:s.F t",
    CultureInfo.CreateSpecificCulture("el-GR")));
// Displays 6:9:1 μ
date1 = new DateTime(2008, 1, 1, 18, 9, 1, 500);
Console.WriteLine(date1.ToString("h:m:s.F t",
    CultureInfo.InvariantCulture));
// Displays 6:9:1.5 P
Console.WriteLine(date1.ToString("h:m:s.F t",
    CultureInfo.CreateSpecificCulture("el-GR")));
// Displays 6:9:1.5 μ
```

```
Dim date1 As Date
date1 = #6:09:01PM#
Console.WriteLine(date1.ToString("h:m:s.F t", _
    CultureInfo.InvariantCulture))
' Displays 6:9:1 P
Console.WriteLine(date1.ToString("h:m:s.F t", _
    CultureInfo.CreateSpecificCulture("el-GR")))
' Displays 6:9:1 μ
date1 = New Date(2008, 1, 1, 18, 9, 1, 500)
Console.WriteLine(date1.ToString("h:m:s.F t", _
    CultureInfo.InvariantCulture))
' Displays 6:9:1.5 P
Console.WriteLine(date1.ToString("h:m:s.F t", _
    CultureInfo.CreateSpecificCulture("el-GR")))
' Displays 6:9:1.5 μ
```

[Back to table](#)

The "mm" custom format specifier

The "mm" custom format specifier (plus any number of additional "m" specifiers) represents the minute as a number from 00 through 59. The minute represents whole minutes that have passed since the last hour. A single-digit minute is formatted with a leading zero.

The following example includes the "mm" custom format specifier in a custom format string.

```
DateTime date1;
date1 = new DateTime(2008, 1, 1, 18, 9, 1);
Console.WriteLine(date1.ToString("hh:mm:ss tt",
    CultureInfo.InvariantCulture));
// Displays 06:09:01 PM
Console.WriteLine(date1.ToString("hh:mm:ss tt",
    CultureInfo.CreateSpecificCulture("hu-HU")));
// Displays 06:09:01 du.
date1 = new DateTime(2008, 1, 1, 18, 9, 1, 500);
Console.WriteLine(date1.ToString("hh:mm:ss.ff tt",
    CultureInfo.InvariantCulture));
// Displays 06:09:01.50 PM
Console.WriteLine(date1.ToString("hh:mm:ss.ff tt",
    CultureInfo.CreateSpecificCulture("hu-HU")));
// Displays 06:09:01.50 du.
```

```

Dim date1 As Date
date1 = #6:09:01PM#
Console.WriteLine(date1.ToString("hh:mm:ss tt", _
    CultureInfo.InvariantCulture))
' Displays 06:09:01 PM
Console.WriteLine(date1.ToString("hh:mm:ss tt", _
    CultureInfo.CreateSpecificCulture("hu-HU")))
' Displays 06:09:01 du.
date1 = New Date(2008, 1, 1, 18, 9, 1, 500)
Console.WriteLine(date1.ToString("hh:mm:ss.ff tt", _
    CultureInfo.InvariantCulture))
' Displays 06:09:01.50 PM
Console.WriteLine(date1.ToString("hh:mm:ss.ff tt", _
    CultureInfo.CreateSpecificCulture("hu-HU")))
' Displays 06:09:01.50 du.

```

[Back to table](#)

Month "M" format specifier

The "M" custom format specifier

The "M" custom format specifier represents the month as a number from 1 through 12 (or from 1 through 13 for calendars that have 13 months). A single-digit month is formatted without a leading zero.

If the "M" format specifier is used without other custom format specifiers, it's interpreted as the "M" standard date and time format specifier. For more information about using a single format specifier, see [Using Single Custom Format Specifiers](#) later in this article.

The following example includes the "M" custom format specifier in a custom format string.

```

DateTime date1 = new DateTime(2008, 8, 18);
Console.WriteLine(date1.ToString("(M) MMM, MMMM",
    CultureInfo.CreateSpecificCulture("en-US")));
// Displays (8) Aug, August
Console.WriteLine(date1.ToString("(M) MMM, MMMM",
    CultureInfo.CreateSpecificCulture("nl-NL")));
// Displays (8) aug, augustus
Console.WriteLine(date1.ToString("(M) MMM, MMMM",
    CultureInfo.CreateSpecificCulture("lv-LV")));
// Displays (8) Aug, augusts

```

```

Dim date1 As Date = #8/18/2008#
Console.WriteLine(date1.ToString("(M) MMM, MMMM",
    CultureInfo.CreateSpecificCulture("en-US")))
' Displays (8) Aug, August
Console.WriteLine(date1.ToString("(M) MMM, MMMM",
    CultureInfo.CreateSpecificCulture("nl-NL")))
' Displays (8) aug, augustus
Console.WriteLine(date1.ToString("(M) MMM, MMMM",
    CultureInfo.CreateSpecificCulture("lv-LV")))
' Displays (8) Aug, augusts

```

[Back to table](#)

The "MM" custom format specifier

The "MM" custom format specifier represents the month as a number from 01 through 12 (or from 1 through 13 for calendars that have 13 months). A single-digit month is formatted with a leading zero.

The following example includes the "MM" custom format specifier in a custom format string.

```

DateTime date1 = new DateTime(2008, 1, 2, 6, 30, 15);

Console.WriteLine(date1.ToString("dd, MM",
                               CultureInfo.InvariantCulture));
// 02, 01

```

```

Dim date1 As Date = #1/2/2008 6:30:15AM#

Console.WriteLine(date1.ToString("dd, MM",
                               CultureInfo.InvariantCulture))
' 02, 01

```

[Back to table](#)

The "MMM" custom format specifier

The "MMM" custom format specifier represents the abbreviated name of the month. The localized abbreviated name of the month is retrieved from the [DateTimeFormatInfo.AbbreviatedMonthNames](#) property of the current or specified culture.

The following example includes the "MMM" custom format specifier in a custom format string.

```

DateTime date1 = new DateTime(2008, 8, 29, 19, 27, 15);

Console.WriteLine(date1.ToString("ddd d MMM",
                               CultureInfo.CreateSpecificCulture("en-US")));
// Displays Fri 29 Aug
Console.WriteLine(date1.ToString("ddd d MMM",
                               CultureInfo.CreateSpecificCulture("fr-FR")));
// Displays ven. 29 août

```

```

Dim date1 As Date = #08/29/2008 7:27:15PM#

Console.WriteLine(date1.ToString("ddd d MMM",
                               CultureInfo.CreateSpecificCulture("en-US")))
' Displays Fri 29 Aug
Console.WriteLine(date1.ToString("ddd d MMM",
                               CultureInfo.CreateSpecificCulture("fr-FR")))
' Displays ven. 29 août

```

[Back to table](#)

The "MMMM" custom format specifier

The "MMMM" custom format specifier represents the full name of the month. The localized name of the month is retrieved from the [DateTimeFormatInfo.MonthNames](#) property of the current or specified culture.

The following example includes the "MMMM" custom format specifier in a custom format string.

```

DateTime date1 = new DateTime(2008, 8, 29, 19, 27, 15);

Console.WriteLine(date1.ToString("dddd dd MMMM",
                               CultureInfo.CreateSpecificCulture("en-US")));
// Displays Friday 29 August
Console.WriteLine(date1.ToString("dddd dd MMMM",
                               CultureInfo.CreateSpecificCulture("it-IT")));
// Displays venerdì 29 agosto

```

```

Dim date1 As Date = #08/29/2008 7:27:15PM#
Console.WriteLine(date1.ToString("dddd dd MMMM",
                               CultureInfo.CreateSpecificCulture("en-US")))
' Displays Friday 29 August
Console.WriteLine(date1.ToString("dddd dd MMMM",
                               CultureInfo.CreateSpecificCulture("it-IT")))
' Displays venerdì 29 agosto

```

[Back to table](#)

Seconds "s" format specifier

The "s" custom format specifier

The "s" custom format specifier represents the seconds as a number from 0 through 59. The result represents whole seconds that have passed since the last minute. A single-digit second is formatted without a leading zero.

If the "s" format specifier is used without other custom format specifiers, it's interpreted as the "s" standard date and time format specifier. For more information about using a single format specifier, see [Using Single Custom Format Specifiers](#) later in this article.

The following example includes the "s" custom format specifier in a custom format string.

```

DateTime date1;
date1 = new DateTime(2008, 1, 1, 18, 9, 1);
Console.WriteLine(date1.ToString("h:m:s.F t",
                               CultureInfo.InvariantCulture));
// Displays 6:9:1 P
Console.WriteLine(date1.ToString("h:m:s.F t",
                               CultureInfo.CreateSpecificCulture("el-GR")));
// Displays 6:9:1 μ
date1 = new DateTime(2008, 1, 1, 18, 9, 1, 500);
Console.WriteLine(date1.ToString("h:m:s.F t",
                               CultureInfo.InvariantCulture));
// Displays 6:9:1.5 P
Console.WriteLine(date1.ToString("h:m:s.F t",
                               CultureInfo.CreateSpecificCulture("el-GR")));
// Displays 6:9:1.5 μ

```

```

Dim date1 As Date
date1 = #6:09:01PM#
Console.WriteLine(date1.ToString("h:m:s.F t",
                               CultureInfo.InvariantCulture))
' Displays 6:9:1 P
Console.WriteLine(date1.ToString("h:m:s.F t",
                               CultureInfo.CreateSpecificCulture("el-GR")))
' Displays 6:9:1 μ
date1 = New Date(2008, 1, 1, 18, 9, 1, 500)
Console.WriteLine(date1.ToString("h:m:s.F t",
                               CultureInfo.InvariantCulture))
' Displays 6:9:1.5 P
Console.WriteLine(date1.ToString("h:m:s.F t",
                               CultureInfo.CreateSpecificCulture("el-GR")))
' Displays 6:9:1.5 μ

```

[Back to table](#)

The "ss" custom format specifier

The "ss" custom format specifier (plus any number of additional "s" specifiers) represents the seconds as a

number from 00 through 59. The result represents whole seconds that have passed since the last minute. A single-digit second is formatted with a leading zero.

The following example includes the "ss" custom format specifier in a custom format string.

```
DateTime date1;
date1 = new DateTime(2008, 1, 1, 18, 9, 1);
Console.WriteLine(date1.ToString("hh:mm:ss tt",
    CultureInfo.InvariantCulture));
// Displays 06:09:01 PM
Console.WriteLine(date1.ToString("hh:mm:ss tt",
    CultureInfo.CreateSpecificCulture("hu-HU")));
// Displays 06:09:01 du.
date1 = new DateTime(2008, 1, 1, 18, 9, 1, 500);
Console.WriteLine(date1.ToString("hh:mm:ss.ff tt",
    CultureInfo.InvariantCulture));
// Displays 06:09:01.50 PM
Console.WriteLine(date1.ToString("hh:mm:ss.ff tt",
    CultureInfo.CreateSpecificCulture("hu-HU")));
// Displays 06:09:01.50 du.
```

```
Dim date1 As Date
date1 = #6:09:01PM#
Console.WriteLine(date1.ToString("hh:mm:ss tt", _
    CultureInfo.InvariantCulture))
' Displays 06:09:01 PM
Console.WriteLine(date1.ToString("hh:mm:ss tt", _
    CultureInfo.CreateSpecificCulture("hu-HU")))
' Displays 06:09:01 du.
date1 = New Date(2008, 1, 1, 18, 9, 1, 500)
Console.WriteLine(date1.ToString("hh:mm:ss.ff tt", _
    CultureInfo.InvariantCulture))
' Displays 06:09:01.50 PM
Console.WriteLine(date1.ToString("hh:mm:ss.ff tt", _
    CultureInfo.CreateSpecificCulture("hu-HU")))
' Displays 06:09:01.50 du.
```

[Back to table](#)

Meridiem "t" format specifier

The "t" custom format specifier

The "t" custom format specifier represents the first character of the AM/PM designator. The appropriate localized designator is retrieved from the [DateTimeFormatInfo.AMDesignator](#) or [DateTimeFormatInfo.PMDesignator](#) property of the current or specific culture. The AM designator is used for all times from 0:00:00 (midnight) to 11:59:59.999. The PM designator is used for all times from 12:00:00 (noon) to 23:59:59.999.

If the "t" format specifier is used without other custom format specifiers, it's interpreted as the "t" standard date and time format specifier. For more information about using a single format specifier, see [Using Single Custom Format Specifiers](#) later in this article.

The following example includes the "t" custom format specifier in a custom format string.

```

DateTime date1;
date1 = new DateTime(2008, 1, 1, 18, 9, 1);
Console.WriteLine(date1.ToString("h:m:s.F t",
                               CultureInfo.InvariantCulture));
// Displays 6:9:1 P
Console.WriteLine(date1.ToString("h:m:s.F t",
                               CultureInfo.CreateSpecificCulture("el-GR")));
// Displays 6:9:1 μ
date1 = new DateTime(2008, 1, 1, 18, 9, 1, 500);
Console.WriteLine(date1.ToString("h:m:s.F t",
                               CultureInfo.InvariantCulture));
// Displays 6:9:1.5 P
Console.WriteLine(date1.ToString("h:m:s.F t",
                               CultureInfo.CreateSpecificCulture("el-GR")));
// Displays 6:9:1.5 μ

```

```

Dim date1 As Date
date1 = #6:09:01PM#
Console.WriteLine(date1.ToString("h:m:s.F t", _
                               CultureInfo.InvariantCulture))
' Displays 6:9:1 P
Console.WriteLine(date1.ToString("h:m:s.F t", _
                               CultureInfo.CreateSpecificCulture("el-GR")))
' Displays 6:9:1 μ
date1 = New Date(2008, 1, 1, 18, 9, 1, 500)
Console.WriteLine(date1.ToString("h:m:s.F t", _
                               CultureInfo.InvariantCulture))
' Displays 6:9:1.5 P
Console.WriteLine(date1.ToString("h:m:s.F t", _
                               CultureInfo.CreateSpecificCulture("el-GR")))
' Displays 6:9:1.5 μ

```

[Back to table](#)

The "tt" custom format specifier

The "tt" custom format specifier (plus any number of additional "t" specifiers) represents the entire AM/PM designator. The appropriate localized designator is retrieved from the [DateTimeFormatInfo.AMDesignator](#) or [DateTimeFormatInfo.PMDesignator](#) property of the current or specific culture. The AM designator is used for all times from 0:00:00 (midnight) to 11:59:59.999. The PM designator is used for all times from 12:00:00 (noon) to 23:59:59.999.

Make sure to use the "tt" specifier for languages for which it's necessary to maintain the distinction between AM and PM. An example is Japanese, for which the AM and PM designators differ in the second character instead of the first character.

The following example includes the "tt" custom format specifier in a custom format string.

```

DateTime date1;
date1 = new DateTime(2008, 1, 1, 18, 9, 1);
Console.WriteLine(date1.ToString("hh:mm:ss tt",
                               CultureInfo.InvariantCulture));
// Displays 06:09:01 PM
Console.WriteLine(date1.ToString("hh:mm:ss tt",
                               CultureInfo.CreateSpecificCulture("hu-HU")));
// Displays 06:09:01 du.
date1 = new DateTime(2008, 1, 1, 18, 9, 1, 500);
Console.WriteLine(date1.ToString("hh:mm:ss.ff tt",
                               CultureInfo.InvariantCulture));
// Displays 06:09:01.50 PM
Console.WriteLine(date1.ToString("hh:mm:ss.ff tt",
                               CultureInfo.CreateSpecificCulture("hu-HU")));
// Displays 06:09:01.50 du.

```

```

Dim date1 As Date
date1 = #6:09:01PM#
Console.WriteLine(date1.ToString("hh:mm:ss tt", _
                               CultureInfo.InvariantCulture))
' Displays 06:09:01 PM
Console.WriteLine(date1.ToString("hh:mm:ss tt", _
                               CultureInfo.CreateSpecificCulture("hu-HU")))
' Displays 06:09:01 du.
date1 = New Date(2008, 1, 1, 18, 9, 1, 500)
Console.WriteLine(date1.ToString("hh:mm:ss.ff tt", _
                               CultureInfo.InvariantCulture))
' Displays 06:09:01.50 PM
Console.WriteLine(date1.ToString("hh:mm:ss.ff tt", _
                               CultureInfo.CreateSpecificCulture("hu-HU")))
' Displays 06:09:01.50 du.

```

[Back to table](#)

Year "y" format specifier

The "y" custom format specifier

The "y" custom format specifier represents the year as a one-digit or two-digit number. If the year has more than two digits, only the two low-order digits appear in the result. If the first digit of a two-digit year begins with a zero (for example, 2008), the number is formatted without a leading zero.

If the "y" format specifier is used without other custom format specifiers, it's interpreted as the "y" standard date and time format specifier. For more information about using a single format specifier, see [Using Single Custom Format Specifiers](#) later in this article.

The following example includes the "y" custom format specifier in a custom format string.

```

DateTime date1 = new DateTime(1, 12, 1);
DateTime date2 = new DateTime(2010, 1, 1);
Console.WriteLine(date1.ToString("%y"));
// Displays 1
Console.WriteLine(date1.ToString("yy"));
// Displays 01
Console.WriteLine(date1.ToString("yyy"));
// Displays 001
Console.WriteLine(date1.ToString("yyyy"));
// Displays 0001
Console.WriteLine(date1.ToString("yyyyy"));
// Displays 00001
Console.WriteLine(date2.ToString("%y"));
// Displays 10
Console.WriteLine(date2.ToString("yy"));
// Displays 10
Console.WriteLine(date2.ToString("yyy"));
// Displays 2010
Console.WriteLine(date2.ToString("yyyy"));
// Displays 2010
Console.WriteLine(date2.ToString("yyyyy"));
// Displays 02010

```

```

Dim date1 As Date = #12/1/0001#
Dim date2 As Date = #1/1/2010#
Console.WriteLine(date1.ToString("%y"))
' Displays 1
Console.WriteLine(date1.ToString("yy"))
' Displays 01
Console.WriteLine(date1.ToString("yyy"))
' Displays 001
Console.WriteLine(date1.ToString("yyyy"))
' Displays 0001
Console.WriteLine(date1.ToString("yyyyy"))
' Displays 00001
Console.WriteLine(date2.ToString("%y"))
' Displays 10
Console.WriteLine(date2.ToString("yy"))
' Displays 10
Console.WriteLine(date2.ToString("yyy"))
' Displays 2010
Console.WriteLine(date2.ToString("yyyy"))
' Displays 2010
Console.WriteLine(date2.ToString("yyyyy"))
' Displays 02010

```

[Back to table](#)

The "yy" custom format specifier

The "yy" custom format specifier represents the year as a two-digit number. If the year has more than two digits, only the two low-order digits appear in the result. If the two-digit year has fewer than two significant digits, the number is padded with leading zeros to produce two digits.

In a parsing operation, a two-digit year that is parsed using the "yy" custom format specifier is interpreted based on the [Calendar.TwoDigitYearMax](#) property of the format provider's current calendar. The following example parses the string representation of a date that has a two-digit year by using the default Gregorian calendar of the en-US culture, which, in this case, is the current culture. It then changes the current culture's [CultureInfo](#) object to use a [GregorianCalendar](#) object whose [TwoDigitYearMax](#) property has been modified.

```
using System;
using System.Globalization;
using System.Threading;

public class Example7
{
    public static void Main()
    {
        string fmt = "dd-MMM-yy";
        string value = "24-Jan-49";

        Calendar cal = (Calendar)CultureInfo.CurrentCulture.Calendar.Clone();
        Console.WriteLine("Two Digit Year Range: {0} - {1}",
            cal.TwoDigitYearMax - 99, cal.TwoDigitYearMax);

        Console.WriteLine("{0:d}", DateTime.ParseExact(value, fmt, null));
        Console.WriteLine();

        cal.TwoDigitYearMax = 2099;
        CultureInfo culture = (CultureInfo)CultureInfo.CurrentCulture.Clone();
        culture.DateTimeFormat.Calendar = cal;
        Thread.CurrentThread.CurrentCulture = culture;

        Console.WriteLine("Two Digit Year Range: {0} - {1}",
            cal.TwoDigitYearMax - 99, cal.TwoDigitYearMax);
        Console.WriteLine("{0:d}", DateTime.ParseExact(value, fmt, null));
    }
}

// The example displays the following output:
//      Two Digit Year Range: 1930 - 2029
//      1/24/1949
//
//      Two Digit Year Range: 2000 - 2099
//      1/24/2049
```

```

Imports System.Globalization
Imports System.Threading

Module Example
    Public Sub Main()
        Dim fmt As String = "dd-MMM-yy"
        Dim value As String = "24-Jan-49"

        Dim cal As Calendar = CType(CultureInfo.CurrentCulture.Calendar.Clone(), Calendar)
        Console.WriteLine("Two Digit Year Range: {0} - {1}",
                         cal.TwoDigitYearMax - 99, cal.TwoDigitYearMax)

        Console.WriteLine("{0:d}", DateTime.ParseExact(value, fmt, Nothing))
        Console.WriteLine()

        cal.TwoDigitYearMax = 2099
        Dim culture As CultureInfo = CType(CultureInfo.CurrentCulture.Clone(), CultureInfo)
        culture.DateTimeFormat.Calendar = cal
        Thread.CurrentThread.CurrentCulture = culture

        Console.WriteLine("Two Digit Year Range: {0} - {1}",
                         cal.TwoDigitYearMax - 99, cal.TwoDigitYearMax)
        Console.WriteLine("{0:d}", DateTime.ParseExact(value, fmt, Nothing))
    End Sub
End Module
' The example displays the following output:
'   Two Digit Year Range: 1930 - 2029
'   1/24/1949
'
'   Two Digit Year Range: 2000 - 2099
'   1/24/2049

```

The following example includes the "yy" custom format specifier in a custom format string.

```

DateTime date1 = new DateTime(1, 12, 1);
DateTime date2 = new DateTime(2010, 1, 1);
Console.WriteLine(date1.ToString("%y"));
// Displays 1
Console.WriteLine(date1.ToString("yy"));
// Displays 01
Console.WriteLine(date1.ToString("yyy"));
// Displays 001
Console.WriteLine(date1.ToString("yyyy"));
// Displays 0001
Console.WriteLine(date2.ToString("%y"));
// Displays 10
Console.WriteLine(date2.ToString("yy"));
// Displays 10
Console.WriteLine(date2.ToString("yyy"));
// Displays 2010
Console.WriteLine(date2.ToString("yyyy"));
// Displays 2010
Console.WriteLine(date2.ToString("yyyyy"));
// Displays 02010

```

```

Dim date1 As Date = #12/1/0001#
Dim date2 As Date = #1/1/2010#
Console.WriteLine(date1.ToString("%y"))
' Displays 1
Console.WriteLine(date1.ToString("yy"))
' Displays 01
Console.WriteLine(date1.ToString("yyy"))
' Displays 001
Console.WriteLine(date1.ToString("yyyy"))
' Displays 0001
Console.WriteLine(date1.ToString("yyyyy"))
' Displays 00001
Console.WriteLine(date2.ToString("%y"))
' Displays 10
Console.WriteLine(date2.ToString("yy"))
' Displays 10
Console.WriteLine(date2.ToString("yyy"))
' Displays 2010
Console.WriteLine(date2.ToString("yyyy"))
' Displays 2010
Console.WriteLine(date2.ToString("yyyyy"))
' Displays 02010

```

[Back to table](#)

The "yyy" custom format specifier

The "yyy" custom format specifier represents the year with a minimum of three digits. If the year has more than three significant digits, they are included in the result string. If the year has fewer than three digits, the number is padded with leading zeros to produce three digits.

NOTE

For the Thai Buddhist calendar, which can have five-digit years, this format specifier displays all significant digits.

The following example includes the "yyy" custom format specifier in a custom format string.

```

DateTime date1 = new DateTime(1, 12, 1);
DateTime date2 = new DateTime(2010, 1, 1);
Console.WriteLine(date1.ToString("%y"));
// Displays 1
Console.WriteLine(date1.ToString("yy"));
// Displays 01
Console.WriteLine(date1.ToString("yyy"));
// Displays 001
Console.WriteLine(date1.ToString("yyyy"));
// Displays 0001
Console.WriteLine(date1.ToString("yyyyy"));
// Displays 00001
Console.WriteLine(date2.ToString("%y"));
// Displays 10
Console.WriteLine(date2.ToString("yy"));
// Displays 10
Console.WriteLine(date2.ToString("yyy"));
// Displays 2010
Console.WriteLine(date2.ToString("yyyy"));
// Displays 2010
Console.WriteLine(date2.ToString("yyyyy"));
// Displays 02010

```

```

Dim date1 As Date = #12/1/0001#
Dim date2 As Date = #1/1/2010#
Console.WriteLine(date1.ToString("%y"))
' Displays 1
Console.WriteLine(date1.ToString("yy"))
' Displays 01
Console.WriteLine(date1.ToString("yyy"))
' Displays 001
Console.WriteLine(date1.ToString("yyyy"))
' Displays 0001
Console.WriteLine(date1.ToString("yyyyy"))
' Displays 00001
Console.WriteLine(date2.ToString("%y"))
' Displays 10
Console.WriteLine(date2.ToString("yy"))
' Displays 10
Console.WriteLine(date2.ToString("yyy"))
' Displays 2010
Console.WriteLine(date2.ToString("yyyy"))
' Displays 2010
Console.WriteLine(date2.ToString("yyyyy"))
' Displays 02010

```

[Back to table](#)

The "yyyy" custom format specifier

The "yyyy" custom format specifier represents the year with a minimum of four digits. If the year has more than four significant digits, they are included in the result string. If the year has fewer than four digits, the number is padded with leading zeros to produce four digits.

NOTE

For the Thai Buddhist calendar, which can have five-digit years, this format specifier displays a minimum of four digits.

The following example includes the "yyyy" custom format specifier in a custom format string.

```

DateTime date1 = new DateTime(1, 12, 1);
DateTime date2 = new DateTime(2010, 1, 1);
Console.WriteLine(date1.ToString("%y"));
// Displays 1
Console.WriteLine(date1.ToString("yy"));
// Displays 01
Console.WriteLine(date1.ToString("yyy"));
// Displays 001
Console.WriteLine(date1.ToString("yyyy"));
// Displays 0001
Console.WriteLine(date1.ToString("yyyyy"));
// Displays 00001
Console.WriteLine(date2.ToString("%y"));
// Displays 10
Console.WriteLine(date2.ToString("yy"));
// Displays 10
Console.WriteLine(date2.ToString("yyy"));
// Displays 2010
Console.WriteLine(date2.ToString("yyyy"));
// Displays 2010
Console.WriteLine(date2.ToString("yyyyy"));
// Displays 02010

```

```

Dim date1 As Date = #12/1/0001#
Dim date2 As Date = #1/1/2010#
Console.WriteLine(date1.ToString("%y"))
' Displays 1
Console.WriteLine(date1.ToString("yy"))
' Displays 01
Console.WriteLine(date1.ToString("yyy"))
' Displays 001
Console.WriteLine(date1.ToString("yyyy"))
' Displays 0001
Console.WriteLine(date1.ToString("yyyyy"))
' Displays 00001
Console.WriteLine(date2.ToString("%y"))
' Displays 10
Console.WriteLine(date2.ToString("yy"))
' Displays 10
Console.WriteLine(date2.ToString("yyy"))
' Displays 2010
Console.WriteLine(date2.ToString("yyyy"))
' Displays 2010
Console.WriteLine(date2.ToString("yyyyy"))
' Displays 02010

```

[Back to table](#)

The "yyyy" custom format specifier

The "yyyy" custom format specifier (plus any number of additional "y" specifiers) represents the year with a minimum of five digits. If the year has more than five significant digits, they are included in the result string. If the year has fewer than five digits, the number is padded with leading zeros to produce five digits.

If there are additional "y" specifiers, the number is padded with as many leading zeros as necessary to produce the number of "y" specifiers.

The following example includes the "yyyy" custom format specifier in a custom format string.

```

DateTime date1 = new DateTime(1, 12, 1);
DateTime date2 = new DateTime(2010, 1, 1);
Console.WriteLine(date1.ToString("%y"));
// Displays 1
Console.WriteLine(date1.ToString("yy"));
// Displays 01
Console.WriteLine(date1.ToString("yyy"));
// Displays 001
Console.WriteLine(date1.ToString("yyyy"));
// Displays 0001
Console.WriteLine(date1.ToString("yyyyy"));
// Displays 00001
Console.WriteLine(date2.ToString("%y"));
// Displays 10
Console.WriteLine(date2.ToString("yy"));
// Displays 10
Console.WriteLine(date2.ToString("yyy"));
// Displays 2010
Console.WriteLine(date2.ToString("yyyy"));
// Displays 2010
Console.WriteLine(date2.ToString("yyyyy"));
// Displays 02010

```

```

Dim date1 As Date = #12/1/0001#
Dim date2 As Date = #1/1/2010#
Console.WriteLine(date1.ToString("%y"))
' Displays 1
Console.WriteLine(date1.ToString("yy"))
' Displays 01
Console.WriteLine(date1.ToString("yyy"))
' Displays 001
Console.WriteLine(date1.ToString("yyyy"))
' Displays 0001
Console.WriteLine(date1.ToString("yyyyy"))
' Displays 00001
Console.WriteLine(date2.ToString("%y"))
' Displays 10
Console.WriteLine(date2.ToString("yy"))
' Displays 10
Console.WriteLine(date2.ToString("yyy"))
' Displays 2010
Console.WriteLine(date2.ToString("yyyy"))
' Displays 2010
Console.WriteLine(date2.ToString("yyyyy"))
' Displays 02010

```

[Back to table](#)

Offset "z" format specifier

The "z" custom format specifier

With [DateTime](#) values, the "z" custom format specifier represents the signed offset of the specified time zone from Coordinated Universal Time (UTC), measured in hours. The offset is always displayed with a leading sign. A plus sign (+) indicates hours ahead of UTC, and a minus sign (-) indicates hours behind UTC. A single-digit offset is formatted *without* a leading zero.

The following table shows how the offset value changes depending on [DateTimeKind](#).

DATETIMEKIND VALUE	OFFSET VALUE
Local	The signed offset of the local operating system's time zone from UTC.
Unspecified	The signed offset of the local operating system's time zone from UTC.
Utc	+0 on .NET Core and .NET 5+. On .NET Framework, the signed offset of the local operating system's time zone from UTC.

With [DateTimeOffset](#) values, this format specifier represents the [DateTimeOffset](#) value's offset from UTC in hours.

If the "z" format specifier is used without other custom format specifiers, it's interpreted as a standard date and time format specifier and throws a [FormatException](#). For more information about using a single format specifier, see [Using Single Custom Format Specifiers](#) later in this article.

The following example includes the "z" custom format specifier in a custom format string.

```

DateTime date1 = DateTime.UtcNow;
Console.WriteLine(String.Format("{0:%z}, {0:zz}, {0:zzz}", 
    date1));
// Displays -7, -07, -07:00 on .NET Framework
// Displays +0, +00, +00:00 on .NET Core and .NET 5+

DateTimeOffset date2 = new DateTimeOffset(2008, 8, 1, 0, 0, 0,
    new TimeSpan(6, 0, 0));
Console.WriteLine(String.Format("{0:%z}, {0:zz}, {0:zzz}", 
    date2));
// Displays +6, +06, +06:00

```

```

Dim date1 As Date = Date.UtcNow
Console.WriteLine(String.Format("{0:%z}, {0:zz}, {0:zzz}", _ 
    date1))
' Displays -7, -07, -07:00 on .NET Framework
' Displays +0, +00, +00:00 on .NET Core and .NET 5+

Dim date2 As New DateTimeOffset(2008, 8, 1, 0, 0, 0, _ 
    New Timespan(6, 0, 0))
Console.WriteLine(String.Format("{0:%z}, {0:zz}, {0:zzz}", _ 
    date2))
' Displays +6, +06, +06:00

```

[Back to table](#)

The "zz" custom format specifier

With [DateTime](#) values, the "zz" custom format specifier represents the signed offset of the specified time zone from UTC, measured in hours. The offset is always displayed with a leading sign. A plus sign (+) indicates hours ahead of UTC, and a minus sign (-) indicates hours behind UTC. A single-digit offset is formatted *with* a leading zero.

The following table shows how the offset value changes depending on [DateTimeKind](#).

DATETIMEKIND VALUE	OFFSET VALUE
Local	The signed offset of the local operating system's time zone from UTC.
Unspecified	The signed offset of the local operating system's time zone from UTC.
Utc	+00 on .NET Core and .NET 5+. On .NET Framework, the signed offset of the local operating system's time zone from UTC.

With [DateTimeOffset](#) values, this format specifier represents the [DateTimeOffset](#) value's offset from UTC in hours.

The following example includes the "zz" custom format specifier in a custom format string.

```

DateTime date1 = DateTime.UtcNow;
Console.WriteLine(String.Format("{0:%z}, {0:zz}, {0:zzz}",
    date1));
// Displays -7, -07, -07:00 on .NET Framework
// Displays +0, +00, +00:00 on .NET Core and .NET 5+

DateTimeOffset date2 = new DateTimeOffset(2008, 8, 1, 0, 0, 0,
    new TimeSpan(6, 0, 0));
Console.WriteLine(String.Format("{0:%z}, {0:zz}, {0:zzz}",
    date2));
// Displays +6, +06, +06:00

```

```

Dim date1 As Date = Date.UtcNow
Console.WriteLine(String.Format("{0:%z}, {0:zz}, {0:zzz}", _
    date1))
' Displays -7, -07, -07:00 on .NET Framework
' Displays +0, +00, +00:00 on .NET Core and .NET 5+

Dim date2 As New DateTimeOffset(2008, 8, 1, 0, 0, 0, _
    New Timespan(6, 0, 0))
Console.WriteLine(String.Format("{0:%z}, {0:zz}, {0:zzz}", _
    date2))
' Displays +6, +06, +06:00

```

[Back to table](#)

The "zzz" custom format specifier

With [DateTime](#) values, the "zzz" custom format specifier represents the signed offset of the specified time zone from UTC, measured in hours and minutes. The offset is always displayed with a leading sign. A plus sign (+) indicates hours ahead of UTC, and a minus sign (-) indicates hours behind UTC. A single-digit offset is formatted with a leading zero.

The following table shows how the offset value changes depending on [DateTimeKind](#).

DATETIMEKIND VALUE	OFFSET VALUE
Local	The signed offset of the local operating system's time zone from UTC.
Unspecified	The signed offset of the local operating system's time zone from UTC.
Utc	+00:00 on .NET Core and .NET 5+. On .NET Framework, the signed offset of the local operating system's time zone from UTC.

With [DateTimeOffset](#) values, this format specifier represents the [DateTimeOffset](#) value's offset from UTC in hours and minutes.

The following example includes the "zzz" custom format specifier in a custom format string.

```

DateTime date1 = DateTime.UtcNow;
Console.WriteLine(String.Format("{0:%z}, {0:zz}, {0:zzz}", 
    date1));
// Displays -7, -07, -07:00 on .NET Framework
// Displays +0, +00, +00:00 on .NET Core and .NET 5+

DateTimeOffset date2 = new DateTimeOffset(2008, 8, 1, 0, 0, 0,
    new TimeSpan(6, 0, 0));
Console.WriteLine(String.Format("{0:%z}, {0:zz}, {0:zzz}", 
    date2));
// Displays +6, +06, +06:00

```

```

Dim date1 As Date = Date.UtcNow
Console.WriteLine(String.Format("{0:%z}, {0:zz}, {0:zzz}", _ 
    date1))
' Displays -7, -07, -07:00 on .NET Framework
' Displays +0, +00, +00:00 on .NET Core and .NET 5+

Dim date2 As New DateTimeOffset(2008, 8, 1, 0, 0, 0, _ 
    New Timespan(6, 0, 0))
Console.WriteLine(String.Format("{0:%z}, {0:zz}, {0:zzz}", _ 
    date2))
' Displays +6, +06, +06:00

```

[Back to table](#)

Date and time separator specifiers

The ":" custom format specifier

The ":" custom format specifier represents the time separator, which is used to differentiate hours, minutes, and seconds. The appropriate localized time separator is retrieved from the [DateTimeFormatInfo.TimeSeparator](#) property of the current or specified culture.

NOTE

To change the time separator for a particular date and time string, specify the separator character within a literal string delimiter. For example, the custom format string `hh'_'dd'_'ss` produces a result string in which "_" (an underscore) is always used as the time separator. To change the time separator for all dates for a culture, either change the value of the [DateTimeFormatInfo.TimeSeparator](#) property of the current culture, or instantiate a [DateTimeFormatInfo](#) object, assign the character to its [TimeSeparator](#) property, and call an overload of the formatting method that includes an [IFormatProvider](#) parameter.

If the ":" format specifier is used without other custom format specifiers, it's interpreted as a standard date and time format specifier and throws a [FormatException](#). For more information about using a single format specifier, see [Using Single Custom Format Specifiers](#) later in this article.

[Back to table](#)

The "/" custom format specifier

The "/" custom format specifier represents the date separator, which is used to differentiate years, months, and days. The appropriate localized date separator is retrieved from the [DateTimeFormatInfo.DateSeparator](#) property of the current or specified culture.

NOTE

To change the date separator for a particular date and time string, specify the separator character within a literal string delimiter. For example, the custom format string `mm'/'dd'/'yyyy` produces a result string in which "/" is always used as the date separator. To change the date separator for all dates for a culture, either change the value of the [DateTimeFormatInfo.DateSeparator](#) property of the current culture, or instantiate a [DateTimeFormatInfo](#) object, assign the character to its [DateSeparator](#) property, and call an overload of the formatting method that includes an [IFormatProvider](#) parameter.

If the "/" format specifier is used without other custom format specifiers, it's interpreted as a standard date and time format specifier and throws a [FormatException](#). For more information about using a single format specifier, see [Using Single Custom Format Specifiers](#) later in this article.

[Back to table](#)

Character literals

The following characters in a custom date and time format string are reserved and are always interpreted as formatting characters or, in the case of `"`, `'`, `/`, and `\`, as special characters.

- `F`
- `H`
- `K`
- `M`
- `d`
- `f`
- `g`
- `h`
- `m`
- `s`
- `t`
- `y`
- `z`
- `%`
- `:`
- `/`
- `"`
- `'`
- `\`

All other characters are always interpreted as character literals and, in a formatting operation, are included in the result string unchanged. In a parsing operation, they must match the characters in the input string exactly; the comparison is case-sensitive.

The following example includes the literal characters "PST" (for Pacific Standard Time) and "PDT" (for Pacific Daylight Time) to represent the local time zone in a format string. Note that the string is included in the result string, and that a string that includes the local time zone string also parses successfully.

```

using System;
using System.Globalization;

public class Example5
{
    public static void Main()
    {
        String[] formats = { "dd MMM yyyy hh:mm tt PST",
                            "dd MMM yyyy hh:mm tt PDT" };
        var dat = new DateTime(2016, 8, 18, 16, 50, 0);
        // Display the result string.
        Console.WriteLine(dat.ToString(formats[1]));

        // Parse a string.
        String value = "25 Dec 2016 12:00 pm PST";
        DateTime newDate;
        if (DateTime.TryParseExact(value, formats, null,
                                  DateTimeStyles.None, out newDate))
            Console.WriteLine(newDate);
        else
            Console.WriteLine("Unable to parse '{0}'", value);
    }
}
// The example displays the following output:
//      18 Aug 2016 04:50 PM PDT
//      12/25/2016 12:00:00 PM

```

```

Imports System.Globalization

Module Example
    Public Sub Main()
        Dim formats() As String = {"dd MMM yyyy hh:mm tt PST",
                                   "dd MMM yyyy hh:mm tt PDT"}
        Dim dat As New Date(2016, 8, 18, 16, 50, 0)
        ' Display the result string.
        Console.WriteLine(dat.ToString(formats(1)))

        ' Parse a string.
        Dim value As String = "25 Dec 2016 12:00 pm PST"
        Dim newDate As Date
        If Date.TryParseExact(value, formats, Nothing,
                              DateTimeStyles.None, newDate) Then
            Console.WriteLine(newDate)
        Else
            Console.WriteLine("Unable to parse '{0}'", value)
        End If
    End Sub
End Module
' The example displays the following output:
'      18 Aug 2016 04:50 PM PDT
'      12/25/2016 12:00:00 PM

```

There are two ways to indicate that characters are to be interpreted as literal characters and not as reserve characters, so that they can be included in a result string or successfully parsed in an input string:

- By escaping each reserved character. For more information, see [Using the Escape Character](#).

The following example includes the literal characters "pst" (for Pacific Standard time) to represent the local time zone in a format string. Because both "s" and "t" are custom format strings, both characters must be escaped to be interpreted as character literals.

```

using System;
using System.Globalization;

public class Example3
{
    public static void Main()
    {
        String format = "dd MMM yyyy hh:mm tt p\\s\\t";
        var dat = new DateTime(2016, 8, 18, 16, 50, 0);
        // Display the result string.
        Console.WriteLine(dat.ToString(format));

        // Parse a string.
        String value = "25 Dec 2016 12:00 pm pst";
        DateTime newDate;
        if (DateTime.TryParseExact(value, format, null,
            DateTimeStyles.None, out newDate))
            Console.WriteLine(newDate);
        else
            Console.WriteLine("Unable to parse '{0}'", value);
    }
}

// The example displays the following output:
//      18 Aug 2016 04:50 PM pst
//      12/25/2016 12:00:00 PM

```

```

Imports System.Globalization

Module Example
    Public Sub Main()
        Dim fmt As String = "dd MMM yyyy hh:mm tt p\s\t"
        Dim dat As New Date(2016, 8, 18, 16, 50, 0)
        ' Display the result string.
        Console.WriteLine(dat.ToString(fmt))

        ' Parse a string.
        Dim value As String = "25 Dec 2016 12:00 pm pst"
        Dim newDate As Date
        If Date.TryParseExact(value, fmt, Nothing,
            DateTimeStyles.None, newDate) Then
            Console.WriteLine(newDate)
        Else
            Console.WriteLine("Unable to parse '{0}'", value)
        End If
    End Sub
End Module

' The example displays the following output:
'      18 Aug 2016 04:50 PM pst
'      12/25/2016 12:00:00 PM

```

- By enclosing the entire literal string in quotation marks or apostrophes. The following example is like the previous one, except that "pst" is enclosed in quotation marks to indicate that the entire delimited string should be interpreted as character literals.

```

using System;
using System.Globalization;

public class Example6
{
    public static void Main()
    {
        String format = "dd MMM yyyy hh:mm tt \"pst\"";
        var dat = new DateTime(2016, 8, 18, 16, 50, 0);
        // Display the result string.
        Console.WriteLine(dat.ToString(format));

        // Parse a string.
        String value = "25 Dec 2016 12:00 pm pst";
        DateTime newDate;
        if (DateTime.TryParseExact(value, format, null,
            DateTimeStyles.None, out newDate))
            Console.WriteLine(newDate);
        else
            Console.WriteLine("Unable to parse '{0}'", value);
    }
}

// The example displays the following output:
//      18 Aug 2016 04:50 PM pst
//      12/25/2016 12:00:00 PM

```

```

Imports System.Globalization

Module Example
    Public Sub Main()
        Dim fmt As String = "dd MMM yyyy hh:mm tt ""pst"""
        Dim dat As New Date(2016, 8, 18, 16, 50, 0)
        ' Display the result string.
        Console.WriteLine(dat.ToString(fmt))

        ' Parse a string.
        Dim value As String = "25 Dec 2016 12:00 pm pst"
        Dim newDate As Date
        If Date.TryParseExact(value, fmt, Nothing,
            DateTimeStyles.None, newDate) Then
            Console.WriteLine(newDate)
        Else
            Console.WriteLine("Unable to parse '{0}'", value)
        End If
    End Sub
End Module

' The example displays the following output:
'      18 Aug 2016 04:50 PM pst
'      12/25/2016 12:00:00 PM

```

Notes

Using single custom format specifiers

A custom date and time format string consists of two or more characters. Date and time formatting methods interpret any single-character string as a standard date and time format string. If they don't recognize the character as a valid format specifier, they throw a [FormatException](#). For example, a format string that consists only of the specifier "h" is interpreted as a standard date and time format string. However, in this particular case, an exception is thrown because there is no "h" standard date and time format specifier.

To use any of the custom date and time format specifiers as the only specifier in a format string (that is, to use the "d", "f", "F", "g", "h", "H", "K", "m", "M", "s", "t", "y", "z", ":", or "/" custom format specifier by itself), include a

space before or after the specifier, or include a percent ("%) format specifier before the single custom date and time specifier.

For example, "`%h`" is interpreted as a custom date and time format string that displays the hour represented by the current date and time value. You can also use the "h" or "h" format string, although this includes a space in the result string along with the hour. The following example illustrates these three format strings.

```
DateTime dat1 = new DateTime(2009, 6, 15, 13, 45, 0);

Console.WriteLine("{0:%h}", dat1);
Console.WriteLine("{0: h}", dat1);
Console.WriteLine("{0:h }", dat1);
// The example displays the following output:
//      '1'
//      ' 1'
//      '1 '
```

```
Dim dat1 As Date = #6/15/2009 1:45PM#

Console.WriteLine("{0:%h}", dat1)
Console.WriteLine("{0: h}", dat1)
Console.WriteLine("{0:h }", dat1)
' The example displays the following output:
'      '1'
'      ' 1'
'      '1 '
```

Using the Escape character

The "d", "f", "F", "g", "h", "H", "K", "m", "M", "s", "t", "y", "z", ":", or "/" characters in a format string are interpreted as custom format specifiers rather than as literal characters. To prevent a character from being interpreted as a format specifier, you can precede it with a backslash (\), which is the escape character. The escape character signifies that the following character is a character literal that should be included in the result string unchanged.

To include a backslash in a result string, you must escape it with another backslash (`\\"\\`).

NOTE

Some compilers, such as the C++ and C# compilers, may also interpret a single backslash character as an escape character. To ensure that a string is interpreted correctly when formatting, you can use the verbatim string literal character (the @ character) before the string in C#, or add another backslash character before each backslash in C# and C++. The following C# example illustrates both approaches.

The following example uses the escape character to prevent the formatting operation from interpreting the "h" and "m" characters as format specifiers.

```
DateTime date = new DateTime(2009, 06, 15, 13, 45, 30, 90);
string fmt1 = "h \\h m \\m";
string fmt2 = @"h \h m \m";

Console.WriteLine("{0} ({1}) -> {2}", date, fmt1, date.ToString(fmt1));
Console.WriteLine("{0} ({1}) -> {2}", date, fmt2, date.ToString(fmt2));
// The example displays the following output:
//      6/15/2009 1:45:30 PM (h \h m \m) -> 1 h 45 m
//      6/15/2009 1:45:30 PM (h \h m \m) -> 1 h 45 m
```

```
Dim date1 As Date = #6/15/2009 13:45#
Dim fmt As String = "h \h m \m"

Console.WriteLine("{0} ({1}) -> {2}", date1, fmt, date1.ToString(fmt))
' The example displays the following output:
'       6/15/2009 1:45:00 PM (h \h m \m) -> 1 h 45 m
```

Control Panel settings

The **Regional and Language Options** settings in Control Panel influence the result string produced by a formatting operation that includes many of the custom date and time format specifiers. These settings are used to initialize the [DateTimeFormatInfo](#) object associated with the current culture, which provides values used to govern formatting. Computers that use different settings generate different result strings.

In addition, if you use the [CultureInfo\(String\)](#) constructor to instantiate a new [CultureInfo](#) object that represents the same culture as the current system culture, any customizations established by the **Regional and Language Options** item in Control Panel will be applied to the new [CultureInfo](#) object. You can use the [CultureInfo\(String, Boolean\)](#) constructor to create a [CultureInfo](#) object that doesn't reflect a system's customizations.

DateTimeFormatInfo properties

Formatting is influenced by properties of the current [DateTimeFormatInfo](#) object, which is provided implicitly by the current culture or explicitly by the [IFormatProvider](#) parameter of the method that invokes formatting. For the [IFormatProvider](#) parameter, you should specify a [CultureInfo](#) object, which represents a culture, or a [DateTimeFormatInfo](#) object.

The result string produced by many of the custom date and time format specifiers also depends on properties of the current [DateTimeFormatInfo](#) object. Your application can change the result produced by some custom date and time format specifiers by changing the corresponding [DateTimeFormatInfo](#) property. For example, the "ddd" format specifier adds an abbreviated weekday name found in the [AbbreviatedDayNames](#) string array to the result string. Similarly, the "MMMM" format specifier adds a full month name found in the [MonthNames](#) string array to the result string.

See also

- [System.DateTime](#)
- [System.IFormatProvider](#)
- [Formatting types](#)
- [Standard Date and Time format strings](#)
- [Sample: .NET Core WinForms formatting utility \(C#\)](#)
- [Sample: .NET Core WinForms formatting utility \(Visual Basic\)](#)

Standard TimeSpan format strings

9/20/2022 • 9 minutes to read • [Edit Online](#)

A standard [TimeSpan](#) format string uses a single format specifier to define the text representation of a [TimeSpan](#) value that results from a formatting operation. Any format string that contains more than one character, including white space, is interpreted as a custom [TimeSpan](#) format string. For more information, see [Custom TimeSpan format strings](#).

The string representations of [TimeSpan](#) values are produced by calls to the overloads of the [TimeSpan.ToString](#) method, as well as by methods that support composite formatting, such as [String.Format](#). For more information, see [Formatting Types](#) and [Composite Formatting](#). The following example illustrates the use of standard format strings in formatting operations.

```
using System;

public class Example
{
    public static void Main()
    {
        TimeSpan duration = new TimeSpan(1, 12, 23, 62);
        string output = "Time of Travel: " + duration.ToString("c");
        Console.WriteLine(output);

        Console.WriteLine("Time of Travel: {0:c}", duration);
    }
}

// The example displays the following output:
//      Time of Travel: 1.12:24:02
//      Time of Travel: 1.12:24:02
```

```
Module Example
    Public Sub Main()
        Dim duration As New TimeSpan(1, 12, 23, 62)
        Dim output As String = "Time of Travel: " + duration.ToString("c")
        Console.WriteLine(output)

        Console.WriteLine("Time of Travel: {0:c}", duration)
    End Sub
End Module

' The example displays the following output:
'      Time of Travel: 1.12:24:02
'      Time of Travel: 1.12:24:02
```

Standard [TimeSpan](#) format strings are also used by the [TimeSpan.ParseExact](#) and [TimeSpan.TryParseExact](#) methods to define the required format of input strings for parsing operations. (Parsing converts the string representation of a value to that value.) The following example illustrates the use of standard format strings in parsing operations.

```

using System;

public class Example
{
    public static void Main()
    {
        string value = "1.03:14:56.1667";
        TimeSpan interval;
        try {
            interval = TimeSpan.ParseExact(value, "c", null);
            Console.WriteLine("Converted '{0}' to {1}", value, interval);
        }
        catch (FormatException) {
            Console.WriteLine("{0}: Bad Format", value);
        }
        catch (OverflowException) {
            Console.WriteLine("{0}: Out of Range", value);
        }

        if (TimeSpan.TryParseExact(value, "c", null, out interval))
            Console.WriteLine("Converted '{0}' to {1}", value, interval);
        else
            Console.WriteLine("Unable to convert {0} to a time interval.",
                             value);
    }
}

// The example displays the following output:
//      Converted '1.03:14:56.1667' to 1.03:14:56.1667000
//      Converted '1.03:14:56.1667' to 1.03:14:56.1667000

```

```

Module Example
Public Sub Main()
    Dim value As String = "1.03:14:56.1667"
    Dim interval As TimeSpan
    Try
        interval = TimeSpan.ParseExact(value, "c", Nothing)
        Console.WriteLine("Converted '{0}' to {1}", value, interval)
    Catch e As FormatException
        Console.WriteLine("{0}: Bad Format", value)
    Catch e As OverflowException
        Console.WriteLine("{0}: Out of Range", value)
    End Try

    If TimeSpan.TryParseExact(value, "c", Nothing, interval) Then
        Console.WriteLine("Converted '{0}' to {1}", value, interval)
    Else
        Console.WriteLine("Unable to convert {0} to a time interval.",
                         value)
    End If
End Sub
End Module

' The example displays the following output:
'      Converted '1.03:14:56.1667' to 1.03:14:56.1667000
'      Converted '1.03:14:56.1667' to 1.03:14:56.1667000

```

The following table lists the standard time interval format specifiers.

FORMAT SPECIFIER	NAME	DESCRIPTION	EXAMPLES
------------------	------	-------------	----------

Format Specifier	Name	Description	Examples
"c"	Constant (invariant) format	<p>This specifier is not culture-sensitive. It takes the form</p> <pre>[-] [d'.]hh':mm':ss[.fffffff]</pre> <p>(The "t" and "T" format strings produce the same results.)</p> <p>More information: The Constant ("c") Format Specifier.</p>	<pre>TimeSpan.Zero -> 00:00:00</pre> <pre>New TimeSpan(0, 0, 30, 0)</pre> <pre>-> 00:30:00</pre> <pre>New TimeSpan(3, 17, 25, 30, 500)</pre> <pre>-> 3.17:25:30.5000000</pre>
"g"	General short format	<p>This specifier outputs only what is needed. It is culture-sensitive and takes the form</p> <pre>[-] [d':]hh':mm':ss[.FFFFFF]</pre> <p>More information: The General Short ("g") Format Specifier.</p>	<pre>New TimeSpan(1, 3, 16, 50, 500)</pre> <pre>-> 1:3:16:50.5 (en-US)</pre> <pre>New TimeSpan(1, 3, 16, 50, 500)</pre> <pre>-> 1:3:16:50,5 (fr-FR)</pre> <pre>New TimeSpan(1, 3, 16, 50, 599)</pre> <pre>-> 1:3:16:50.599 (en-US)</pre> <pre>New TimeSpan(1, 3, 16, 50, 599)</pre> <pre>-> 1:3:16:50,599 (fr-FR)</pre>
"G"	General long format	<p>This specifier always outputs days and seven fractional digits. It is culture-sensitive and takes the form</p> <pre>[-]d':hh':mm':ss.fffffff</pre> <p>More information: The General Long ("G") Format Specifier.</p>	<pre>New TimeSpan(18, 30, 0)</pre> <pre>-> 0:18:30:00.0000000 (en-US)</pre> <pre>New TimeSpan(18, 30, 0)</pre> <pre>-> 0:18:30:00,0000000 (fr-FR)</pre>

The Constant ("c") Format Specifier

The "c" format specifier returns the string representation of a [TimeSpan](#) value in the following form:

```
[ - ][d.]hh:mm:ss[.fffffff]
```

Elements in square brackets ([and]) are optional. The period (.) and colon (:) are literal symbols. The following table describes the remaining elements.

Element	Description
-	An optional negative sign, which indicates a negative time interval.
d	The optional number of days, with no leading zeros.

ELEMENT	DESCRIPTION
<i>hh</i>	The number of hours, which ranges from "00" to "23".
<i>mm</i>	The number of minutes, which ranges from "00" to "59".
<i>ss</i>	The number of seconds, which ranges from "0" to "59".
<i>fffffff</i>	The optional fractional portion of a second. Its value can range from "0000001" (one tick, or one ten-millionth of a second) to "9999999" (9,999,999 ten-millionths of a second, or one second less one tick).

Unlike the "g" and "G" format specifiers, the "c" format specifier is not culture-sensitive. It produces the string representation of a [TimeSpan](#) value that is invariant and that's common to versions prior to .NET Framework 4. "c" is the default [TimeSpan.ToString\(\)](#) method formats a time interval value by using the "c" format string.

NOTE

[TimeSpan](#) also supports the "t" and "T" standard format strings, which are identical in behavior to the "c" standard format string.

The following example instantiates two [TimeSpan](#) objects, uses them to perform arithmetic operations, and displays the result. In each case, it uses composite formatting to display the [TimeSpan](#) value by using the "c" format specifier.

```
using System;

public class Example
{
    public static void Main()
    {
        TimeSpan interval1, interval2;
        interval1 = new TimeSpan(7, 45, 16);
        interval2 = new TimeSpan(18, 12, 38);

        Console.WriteLine("{0:c} - {1:c} = {2:c}", interval1,
                          interval2, interval1 - interval2);
        Console.WriteLine("{0:c} + {1:c} = {2:c}", interval1,
                          interval2, interval1 + interval2);

        interval1 = new TimeSpan(0, 0, 1, 14, 365);
        interval2 = TimeSpan.FromTicks(2143756);
        Console.WriteLine("{0:c} + {1:c} = {2:c}", interval1,
                          interval2, interval1 + interval2);
    }
}
// The example displays the following output:
//      07:45:16 - 18:12:38 = -10:27:22
//      07:45:16 + 18:12:38 = 1.01:57:54
//      00:01:14.3650000 + 00:00:00.2143756 = 00:01:14.5793756
```

```

Module Example
    Public Sub Main()
        Dim interval1, interval2 As TimeSpan
        interval1 = New TimeSpan(7, 45, 16)
        interval2 = New TimeSpan(18, 12, 38)

        Console.WriteLine("{0:c} - {1:c} = {2:c}", interval1,
                          interval2, interval1 - interval2)
        Console.WriteLine("{0:c} + {1:c} = {2:c}", interval1,
                          interval2, interval1 + interval2)

        interval1 = New TimeSpan(0, 0, 1, 14, 365)
        interval2 = TimeSpan.FromTicks(2143756)
        Console.WriteLine("{0:c} + {1:c} = {2:c}", interval1,
                          interval2, interval1 + interval2)
    End Sub
End Module
' The example displays the following output:
'   07:45:16 - 18:12:38 = -10:27:22
'   07:45:16 + 18:12:38 = 1.01:57:54
'   00:01:14.365000 + 00:00:00.2143756 = 00:01:14.5793756

```

The General Short ("g") Format Specifier

The "g" [TimeSpan](#) format specifier returns the string representation of a [TimeSpan](#) value in a compact form by including only the elements that are necessary. It has the following form:

`[-][d:]h:mm:ss[.FFFFFFF]`

Elements in square brackets ([and]) are optional. The colon (:) is a literal symbol. The following table describes the remaining elements.

ELEMENT	DESCRIPTION
-	An optional negative sign, which indicates a negative time interval.
d	The optional number of days, with no leading zeros.
h	The number of hours, which ranges from "0" to "23", with no leading zeros.
mm	The number of minutes, which ranges from "00" to "59".
ss	The number of seconds, which ranges from "00" to "59".
.	The fractional seconds separator. It is equivalent to the specified culture's NumberDecimalSeparator property without user overrides.
FFFFFFF	The fractional seconds. As few digits as possible are displayed.

Like the "G" format specifier, the "g" format specifier is localized. Its fractional seconds separator is based on either the current culture or a specified culture's [NumberDecimalSeparator](#) property.

The following example instantiates two [TimeSpan](#) objects, uses them to perform arithmetic operations, and displays the result. In each case, it uses composite formatting to display the [TimeSpan](#) value by using the "g"

format specifier. In addition, it formats the [TimeSpan](#) value by using the formatting conventions of the current system culture (which, in this case, is English - United States or en-US) and the French - France (fr-FR) culture.

```
using System;
using System.Globalization;

public class Example
{
    public static void Main()
    {
        TimeSpan interval1, interval2;
        interval1 = new TimeSpan(7, 45, 16);
        interval2 = new TimeSpan(18, 12, 38);

        Console.WriteLine("{0:g} - {1:g} = {2:g}", interval1,
                          interval2, interval1 - interval2);
        Console.WriteLine(String.Format(new CultureInfo("fr-FR"),
                                         "{0:g} + {1:g} = {2:g}", interval1,
                                         interval2, interval1 + interval2));

        interval1 = new TimeSpan(0, 0, 1, 14, 36);
        interval2 = TimeSpan.FromTicks(2143756);
        Console.WriteLine("{0:g} + {1:g} = {2:g}", interval1,
                          interval2, interval1 + interval2);
    }
}

// The example displays the following output:
//      7:45:16 - 18:12:38 = -10:27:22
//      7:45:16 + 18:12:38 = 1:1:57:54
//      0:01:14.036 + 0:00:00.2143756 = 0:01:14.2503756
```

```
Imports System.Globalization

Module Example
    Public Sub Main()
        Dim interval1, interval2 As TimeSpan
        interval1 = New TimeSpan(7, 45, 16)
        interval2 = New TimeSpan(18, 12, 38)

        Console.WriteLine("{0:g} - {1:g} = {2:g}", interval1,
                          interval2, interval1 - interval2)
        Console.WriteLine(String.Format(New CultureInfo("fr-FR"),
                                         "{0:g} + {1:g} = {2:g}", interval1,
                                         interval2, interval1 + interval2))

        interval1 = New TimeSpan(0, 0, 1, 14, 36)
        interval2 = TimeSpan.FromTicks(2143756)
        Console.WriteLine("{0:g} + {1:g} = {2:g}", interval1,
                          interval2, interval1 + interval2)
    End Sub
End Module

' The example displays the following output:
'      7:45:16 - 18:12:38 = -10:27:22
'      7:45:16 + 18:12:38 = 1:1:57:54
'      0:01:14.036 + 0:00:00.2143756 = 0:01:14.2503756
```

The General Long ("G") Format Specifier

The "G" [TimeSpan](#) format specifier returns the string representation of a [TimeSpan](#) value in a long form that always includes both days and fractional seconds. The string that results from the "G" standard format specifier has the following form:

```
[-]d:hh:mm:ss.fffffff
```

Elements in square brackets ([and]) are optional. The colon (:) is a literal symbol. The following table describes the remaining elements.

ELEMENT	DESCRIPTION
-	An optional negative sign, which indicates a negative time interval.
d	The number of days, with no leading zeros.
hh	The number of hours, which ranges from "00" to "23".
mm	The number of minutes, which ranges from "00" to "59".
ss	The number of seconds, which ranges from "00" to "59".
.	The fractional seconds separator. It is equivalent to the specified culture's NumberDecimalSeparator property without user overrides.
fffffff	The fractional seconds.

Like the "G" format specifier, the "g" format specifier is localized. Its fractional seconds separator is based on either the current culture or a specified culture's [NumberDecimalSeparator](#) property.

The following example instantiates two [TimeSpan](#) objects, uses them to perform arithmetic operations, and displays the result. In each case, it uses composite formatting to display the [TimeSpan](#) value by using the "G" format specifier. In addition, it formats the [TimeSpan](#) value by using the formatting conventions of the current system culture (which, in this case, is English - United States or en-US) and the French - France (fr-FR) culture.

```
using System;
using System.Globalization;

public class Example
{
    public static void Main()
    {
        TimeSpan interval1, interval2;
        interval1 = new TimeSpan(7, 45, 16);
        interval2 = new TimeSpan(18, 12, 38);

        Console.WriteLine("{0:G} - {1:G} = {2:G}", interval1,
                          interval2, interval1 - interval2);
        Console.WriteLine(String.Format(new CultureInfo("fr-FR"),
                                         "{0:G} + {1:G} = {2:G}", interval1,
                                         interval2, interval1 + interval2));

        interval1 = new TimeSpan(0, 0, 1, 14, 36);
        interval2 = TimeSpan.FromTicks(2143756);
        Console.WriteLine("{0:G} + {1:G} = {2:G}", interval1,
                         interval2, interval1 + interval2);
    }
}
// The example displays the following output:
//      0:07:45:16.0000000 - 0:18:12:38.0000000 = -0:10:27:22.0000000
//      0:07:45:16,0000000 + 0:18:12:38,0000000 = 1:01:57:54,0000000
//      0:00:01:14.0360000 + 0:00:00:00.2143756 = 0:00:01:14.2503756
```

```
Imports System.Globalization

Module Example
    Public Sub Main()
        Dim interval1, interval2 As TimeSpan
        interval1 = New TimeSpan(7, 45, 16)
        interval2 = New TimeSpan(18, 12, 38)

        Console.WriteLine("{0:G} - {1:G} = {2:G}", interval1,
                          interval2, interval1 - interval2)
        Console.WriteLine(String.Format(New CultureInfo("fr-FR"),
                                         "{0:G} + {1:G} = {2:G}", interval1,
                                         interval2, interval1 + interval2))

        interval1 = New TimeSpan(0, 0, 1, 14, 36)
        interval2 = TimeSpan.FromTicks(2143756)
        Console.WriteLine("{0:G} + {1:G} = {2:G}", interval1,
                          interval2, interval1 + interval2)
    End Sub
End Module
' The example displays the following output:
'     0:07:45:16.0000000 - 0:18:12:38.0000000 = -0:10:27:22.0000000
'     0:07:45:16,0000000 + 0:18:12:38,0000000 = 1:01:57:54,0000000
'     0:00:01:14.0360000 + 0:00:00:00.2143756 = 0:00:01:14.2503756
```

See also

- [Formatting Types](#)
- [Custom TimeSpan Format Strings](#)
- [Parsing Strings](#)

Custom TimeSpan format strings

9/20/2022 • 43 minutes to read • [Edit Online](#)

A [TimeSpan](#) format string defines the string representation of a [TimeSpan](#) value that results from a formatting operation. A custom format string consists of one or more custom [TimeSpan](#) format specifiers along with any number of literal characters. Any string that isn't a [Standard TimeSpan format string](#) is interpreted as a custom [TimeSpan](#) format string.

IMPORTANT

The custom [TimeSpan](#) format specifiers don't include placeholder separator symbols, such as the symbols that separate days from hours, hours from minutes, or seconds from fractional seconds. Instead, these symbols must be included in the custom format string as string literals. For example, `"dd\.hh\:mm"` defines a period (.) as the separator between days and hours, and a colon (:) as the separator between hours and minutes.

Custom [TimeSpan](#) format specifiers also don't include a sign symbol that enables you to differentiate between negative and positive time intervals. To include a sign symbol, you have to construct a format string by using conditional logic. The [Other characters](#) section includes an example.

The string representations of [TimeSpan](#) values are produced by calls to the overloads of the [TimeSpan.ToString](#) method, and by methods that support composite formatting, such as [String.Format](#). For more information, see [Formatting Types](#) and [Composite Formatting](#). The following example illustrates the use of custom format strings in formatting operations.

```
using System;

public class Example
{
    public static void Main()
    {
        TimeSpan duration = new TimeSpan(1, 12, 23, 62);

        string output = null;
        output = "Time of Travel: " + duration.ToString("%d") + " days";
        Console.WriteLine(output);
        output = "Time of Travel: " + duration.ToString(@"dd\.hh\:mm\:ss");
        Console.WriteLine(output);

        Console.WriteLine("Time of Travel: {0:%d} day(s)", duration);
        Console.WriteLine("Time of Travel: {0:dd\\.hh\\\:mm\\\:ss} days", duration);
    }
}

// The example displays the following output:
//      Time of Travel: 1 days
//      Time of Travel: 01.12:24:02
//      Time of Travel: 1 day(s)
//      Time of Travel: 01.12:24:02 days
```

```

Module Example
    Public Sub Main()
        Dim duration As New TimeSpan(1, 12, 23, 62)

        Dim output As String = Nothing
        output = "Time of Travel: " + duration.ToString("%d") + " days"
        Console.WriteLine(output)
        output = "Time of Travel: " + duration.ToString("dd\.hh\:mm\:ss")
        Console.WriteLine(output)

        Console.WriteLine("Time of Travel: {0:%d} day(s)", duration)
        Console.WriteLine("Time of Travel: {0:dd\.hh\:mm\:ss} days", duration)
    End Sub
End Module
' The example displays the following output:
'   Time of Travel: 1 days
'   Time of Travel: 01.12:24:02
'   Time of Travel: 1 day(s)
'   Time of Travel: 01.12:24:02 days

```

Custom [TimeSpan](#) format strings are also used by the [TimeSpan.ParseExact](#) and [TimeSpan.TryParseExact](#) methods to define the required format of input strings for parsing operations. (Parsing converts the string representation of a value to that value.) The following example illustrates the use of standard format strings in parsing operations.

```

using System;

public class Example
{
    public static void Main()
    {
        string value = null;
        TimeSpan interval;

        value = "6";
        if (TimeSpan.TryParseExact(value, "%d", null, out interval))
            Console.WriteLine("{0} --> {1}", value, interval.ToString("c"));
        else
            Console.WriteLine("Unable to parse '{0}'", value);

        value = "16:32.05";
        if (TimeSpan.TryParseExact(value, @"mm\:ss\.\ff", null, out interval))
            Console.WriteLine("{0} --> {1}", value, interval.ToString("c"));
        else
            Console.WriteLine("Unable to parse '{0}'", value);

        value= "12.035";
        if (TimeSpan.TryParseExact(value, "ss\.\fff", null, out interval))
            Console.WriteLine("{0} --> {1}", value, interval.ToString("c"));
        else
            Console.WriteLine("Unable to parse '{0}'", value);
    }
}

// The example displays the following output:
//   6 --> 6.00:00:00
//   16:32.05 --> 00:16:32.0500000
//   12.035 --> 00:00:12.0350000

```

```

Module Example
    Public Sub Main()
        Dim value As String = Nothing
        Dim interval As TimeSpan

        value = "6"
        If TimeSpan.TryParseExact(value, "%d", Nothing, interval) Then
            Console.WriteLine("{0} --> {1}", value, interval.ToString("c"))
        Else
            Console.WriteLine("Unable to parse '{0}'", value)
        End If

        value = "16:32.05"
        If TimeSpan.TryParseExact(value, "mm\:ss\.\ff", Nothing, interval) Then
            Console.WriteLine("{0} --> {1}", value, interval.ToString("c"))
        Else
            Console.WriteLine("Unable to parse '{0}'", value)
        End If

        value = "12.035"
        If TimeSpan.TryParseExact(value, "ss\.\fff", Nothing, interval) Then
            Console.WriteLine("{0} --> {1}", value, interval.ToString("c"))
        Else
            Console.WriteLine("Unable to parse '{0}'", value)
        End If
    End Sub
End Module
' The example displays the following output:
'      6 --> 6.00:00:00
'  16:32.05 --> 00:16:32.050000
'  12.035 --> 00:00:12.035000

```

The following table describes the custom date and time format specifiers.

FORMAT SPECIFIER	DESCRIPTION	EXAMPLE
"d", "%d"	<p>The number of whole days in the time interval.</p> <p>More information: The "d" custom format specifier.</p>	<pre>new TimeSpan(6, 14, 32, 17, 685):</pre> <pre>%d --> "6"</pre> <pre>d\.hh\:mm --> "6.14:32"</pre>
"dd"- "dddddd"	<p>The number of whole days in the time interval, padded with leading zeros as needed.</p> <p>More information: The "dd"- "dddddd" custom format specifiers.</p>	<pre>new TimeSpan(6, 14, 32, 17, 685):</pre> <pre>ddd --> "006"</pre> <pre>dd\.hh\:mm --> "06.14:32"</pre>
"h", "%h"	<p>The number of whole hours in the time interval that aren't counted as part of days. Single-digit hours don't have a leading zero.</p> <p>More information: The "h" custom format specifier.</p>	<pre>new TimeSpan(6, 14, 32, 17, 685):</pre> <pre>%h --> "14"</pre> <pre>hh\:mm --> "14:32"</pre>

FORMAT SPECIFIER	DESCRIPTION	EXAMPLE
"hh"	<p>The number of whole hours in the time interval that aren't counted as part of days. Single-digit hours have a leading zero.</p> <p>More information: The "hh" custom format specifier.</p>	<pre>new TimeSpan(6, 14, 32, 17, 685):</pre> <pre>hh --> "14"</pre>
"m", "%m"	<p>The number of whole minutes in the time interval that aren't included as part of hours or days. Single-digit minutes don't have a leading zero.</p> <p>More information: The "m" custom format specifier.</p>	<pre>new TimeSpan(6, 14, 8, 17, 685):</pre> <pre>%m --> "8"</pre> <pre>h\:m --> "14:8"</pre>
"mm"	<p>The number of whole minutes in the time interval that aren't included as part of hours or days. Single-digit minutes have a leading zero.</p> <p>More information: The "mm" custom format specifier.</p>	<pre>new TimeSpan(6, 14, 8, 17, 685):</pre> <pre>mm --> "08"</pre> <pre>new TimeSpan(6, 8, 5, 17, 685):</pre> <pre>d\.hh\:mm\:ss --> 6.08:05:17</pre>
"s", "%s"	<p>The number of whole seconds in the time interval that aren't included as part of hours, days, or minutes. Single-digit seconds don't have a leading zero.</p> <p>More information: The "s" custom format specifier.</p>	<pre>TimeSpan.FromSeconds(12.965):</pre> <pre>%s --> 12</pre> <pre>s\.fff --> 12.965</pre>
"ss"	<p>The number of whole seconds in the time interval that aren't included as part of hours, days, or minutes. Single-digit seconds have a leading zero.</p> <p>More information: The "ss" custom format specifier.</p>	<pre>TimeSpan.FromSeconds(6.965):</pre> <pre>ss --> 06</pre> <pre>ss\.fff --> 06.965</pre>
"f", "%f"	<p>The tenths of a second in a time interval.</p> <p>More information: The "f" custom format specifier.</p>	<pre>TimeSpan.FromSeconds(6.895):</pre> <pre>f --> 8</pre> <pre>ss\.f --> 06.8</pre>
"ff"	<p>The hundredths of a second in a time interval.</p> <p>More information: The "ff" custom format specifier.</p>	<pre>TimeSpan.FromSeconds(6.895):</pre> <pre>ff --> 89</pre> <pre>ss\.ff --> 06.89</pre>

Format Specifier	Description	Example
"fff"	<p>The milliseconds in a time interval.</p> <p>More information: The "fff" custom format specifier.</p>	<pre>TimeSpan.FromSeconds(6.895) : fff --> 895 ss\.fff --> 06.895</pre>
"ffff"	<p>The ten-thousandths of a second in a time interval.</p> <p>More information: The "ffff" custom format specifier.</p>	<pre>TimeSpan.Parse("0:0:6.8954321") : ffff --> 8954 ss\.ffff --> 06.8954</pre>
"fffff"	<p>The hundred-thousandths of a second in a time interval.</p> <p>More information: The "fffff" custom format specifier.</p>	<pre>TimeSpan.Parse("0:0:6.8954321") : fffff --> 89543 ss\.fffff --> 06.89543</pre>
"fffffff"	<p>The millionths of a second in a time interval.</p> <p>More information: The "fffffff" custom format specifier.</p>	<pre>TimeSpan.Parse("0:0:6.8954321") : fffffff --> 895432 ss\.fffffff --> 06.895432</pre>
"fffffff"	<p>The ten-millionths of a second (or the fractional ticks) in a time interval.</p> <p>More information: The "fffffff" custom format specifier.</p>	<pre>TimeSpan.Parse("0:0:6.8954321") : fffffff --> 8954321 ss\.fffffff --> 06.8954321</pre>
"F", "%F"	<p>The tenths of a second in a time interval. Nothing is displayed if the digit is zero.</p> <p>More information: The "F" custom format specifier.</p>	<pre>TimeSpan.Parse("00:00:06.32") : %F : 3 TimeSpan.Parse("0:0:3.091") : ss\.F : 03.</pre>
"FF"	<p>The hundredths of a second in a time interval. Any fractional trailing zeros or two zero digits aren't included.</p> <p>More information: The "FF" custom format specifier.</p>	<pre>TimeSpan.Parse("00:00:06.329") : FF : 32 TimeSpan.Parse("0:0:3.101") : ss\.FF : 03.1</pre>
"FFF"	<p>The milliseconds in a time interval. Any fractional trailing zeros aren't included.</p> <p>More information:</p>	<pre>TimeSpan.Parse("00:00:06.3291") : FFF : 329 TimeSpan.Parse("0:0:3.1009") : ss\.FFF : 03.1</pre>

FORMAT SPECIFIER	DESCRIPTION	EXAMPLE
"FFFF"	The ten-thousandths of a second in a time interval. Any fractional trailing zeros aren't included. More information: The "FFFF" custom format specifier .	<code>TimeSpan.Parse("00:00:06.32917") :</code> <code>FFFF : 3291</code> <code>TimeSpan.Parse("0:0:3.10009") :</code> <code>ss\.FFFF : 03.1</code>
"FFFFF"	The hundred-thousandths of a second in a time interval. Any fractional trailing zeros aren't included. More information: The "FFFFF" custom format specifier .	<code>TimeSpan.Parse("00:00:06.329179") :</code> <code>FFFFF : 32917</code> <code>TimeSpan.Parse("0:0:3.100009") :</code> <code>ss\.FFFFF : 03.1</code>
"FFFFFF"	The millionths of a second in a time interval. Any fractional trailing zeros aren't displayed. More information: The "FFFFFF" custom format specifier .	<code>TimeSpan.Parse("00:00:06.3291791") :</code> <code>FFFFFF : 329179</code> <code>TimeSpan.Parse("0:0:3.1000009") :</code> <code>ss\.FFFFFF : 03.1</code>
"FFFFFFF"	The ten-millions of a second in a time interval. Any fractional trailing zeros or seven zeros aren't displayed. More information: The "FFFFFFF" custom format specifier .	<code>TimeSpan.Parse("00:00:06.3291791") :</code> <code>FFFFFFF : 3291791</code> <code>TimeSpan.Parse("0:0:3.1900000") :</code> <code>ss\.FFFFFFF : 03.19</code>
'string'	Literal string delimiter. More information: Other characters .	<code>new TimeSpan(14, 32, 17) :</code> <code>hh':mm':ss --> "14:32:17"</code>
\	The escape character. More information: Other characters .	<code>new TimeSpan(14, 32, 17) :</code> <code>hh\:mm\:ss --> "14:32:17"</code>
Any other character	Any other unescaped character is interpreted as a custom format specifier. More Information: Other characters .	<code>new TimeSpan(14, 32, 17) :</code> <code>hh\:mm\:ss --> "14:32:17"</code>

The "d" custom format specifier

The "d" custom format specifier outputs the value of the `TimeSpan.Days` property, which represents the number of whole days in the time interval. It outputs the full number of days in a `TimeSpan` value, even if the value has more than one digit. If the value of the `TimeSpan.Days` property is zero, the specifier outputs "0".

If the "d" custom format specifier is used alone, specify "%d" so that it isn't misinterpreted as a standard format string. The following example provides an illustration.

```
TimeSpan ts1 = new TimeSpan(16, 4, 3, 17, 250);
Console.WriteLine(ts1.ToString("%d"));
// Displays 16
```

```
Dim ts As New TimeSpan(16, 4, 3, 17, 250)
Console.WriteLine(ts.ToString("%d"))
' Displays 16
```

The following example illustrates the use of the "d" custom format specifier.

```
TimeSpan ts2 = new TimeSpan(4, 3, 17);
Console.WriteLine(ts2.ToString(@"d\.hh\:mm\:ss"));

TimeSpan ts3 = new TimeSpan(3, 4, 3, 17);
Console.WriteLine(ts3.ToString(@"d\.hh\:mm\:ss"));
// The example displays the following output:
//      0.04:03:17
//      3.04:03:17
```

```
Dim ts2 As New TimeSpan(4, 3, 17)
Console.WriteLine(ts2.ToString("d\.hh\:mm\:ss"))

Dim ts3 As New TimeSpan(3, 4, 3, 17)
Console.WriteLine(ts3.ToString("d\.hh\:mm\:ss"))
' The example displays the following output:
'      0.04:03:17
'      3.04:03:17
```

[Back to table](#)

The "dd"- "dddddd" custom format specifiers

The "dd", "ddd", "ddd", "ddddd", "ddddd", "ddddddd", and "ddddddd" custom format specifiers output the value of the [TimeSpan.Days](#) property, which represents the number of whole days in the time interval.

The output string includes a minimum number of digits specified by the number of "d" characters in the format specifier, and it's padded with leading zeros as needed. If the digits in the number of days exceed the number of "d" characters in the format specifier, the full number of days is output in the result string.

The following example uses these format specifiers to display the string representation of two [TimeSpan](#) values. The value of the days component of the first time interval is zero; the value of the days component of the second is 365.

```

TimeSpan ts1 = new TimeSpan(0, 23, 17, 47);
TimeSpan ts2 = new TimeSpan(365, 21, 19, 45);

for (int ctr = 2; ctr <= 8; ctr++)
{
    string fmt = new String('d', ctr) + @"\.\hh\:\mm\:\ss";
    Console.WriteLine("{0} --> {1:" + fmt + "}", fmt, ts1);
    Console.WriteLine("{0} --> {1:" + fmt + "}", fmt, ts2);
    Console.WriteLine();
}
// The example displays the following output:
//      dd\.\hh\:\mm\:\ss --> 00.23:17:47
//      dd\.\hh\:\mm\:\ss --> 365.21:19:45
//
//      ddd\.\hh\:\mm\:\ss --> 000.23:17:47
//      ddd\.\hh\:\mm\:\ss --> 365.21:19:45
//
//      dddd\.\hh\:\mm\:\ss --> 0000.23:17:47
//      dddd\.\hh\:\mm\:\ss --> 365.21:19:45
//
//      ddddd\.\hh\:\mm\:\ss --> 00000.23:17:47
//      dddd\.\hh\:\mm\:\ss --> 365.21:19:45
//
//      dddddd\.\hh\:\mm\:\ss --> 000000.23:17:47
//      ddd\.\hh\:\mm\:\ss --> 365.21:19:45
//
//      ddddddd\.\hh\:\mm\:\ss --> 0000000.23:17:47
//      ddd\.\hh\:\mm\:\ss --> 365.21:19:45

```

```

Dim ts1 As New TimeSpan(0, 23, 17, 47)
Dim ts2 As New TimeSpan(365, 21, 19, 45)

For ctr As Integer = 2 To 8
    Dim fmt As String = New String("d"c, ctr) + "\.\hh\:\mm\:\ss"
    Console.WriteLine("{0} --> {1:" + fmt + "}", fmt, ts1)
    Console.WriteLine("{0} --> {1:" + fmt + "}", fmt, ts2)
    Console.WriteLine()
Next
' The example displays the following output:
'      dd\.\hh\:\mm\:\ss --> 00.23:17:47
'      dd\.\hh\:\mm\:\ss --> 365.21:19:45
'
'      ddd\.\hh\:\mm\:\ss --> 000.23:17:47
'      ddd\.\hh\:\mm\:\ss --> 365.21:19:45
'
'      dddd\.\hh\:\mm\:\ss --> 0000.23:17:47
'      dddd\.\hh\:\mm\:\ss --> 365.21:19:45
'
'      dddddd\.\hh\:\mm\:\ss --> 00000.23:17:47
'      dddd\.\hh\:\mm\:\ss --> 365.21:19:45
'
'      ddddddd\.\hh\:\mm\:\ss --> 000000.23:17:47
'      ddd\.\hh\:\mm\:\ss --> 365.21:19:45
'
'      ddddddd\.\hh\:\mm\:\ss --> 0000000.23:17:47
'      ddd\.\hh\:\mm\:\ss --> 365.21:19:45

```

The "h" custom format specifier

The "h" custom format specifier outputs the value of the `TimeSpan.Hours` property, which represents the number of whole hours in the time interval that isn't counted as part of its day component. It returns a one-digit string value if the value of the `TimeSpan.Hours` property is 0 through 9, and it returns a two-digit string value if the value of the `TimeSpan.Hours` property ranges from 10 to 23.

If the "h" custom format specifier is used alone, specify "%h" so that it isn't misinterpreted as a standard format string. The following example provides an illustration.

```
TimeSpan ts = new TimeSpan(3, 42, 0);
Console.WriteLine("{0:%h} hours {0:%m} minutes", ts);
// The example displays the following output:
//      3 hours 42 minutes
```

```
Dim ts As New TimeSpan(3, 42, 0)
Console.WriteLine("{0:%h} hours {0:%m} minutes", ts)
' The example displays the following output:
'      3 hours 42 minutes
```

Ordinarily, in a parsing operation, an input string that includes only a single number is interpreted as the number of days. You can use the "%h" custom format specifier instead to interpret the numeric string as the number of hours. The following example provides an illustration.

```
string value = "8";
TimeSpan interval;
if (TimeSpan.TryParseExact(value, "%h", null, out interval))
    Console.WriteLine(interval.ToString("c"));
else
    Console.WriteLine("Unable to convert '{0}' to a time interval",
                    value);
// The example displays the following output:
//      08:00:00
```

```
Dim value As String = "8"
Dim interval As TimeSpan
If TimeSpan.TryParseExact(value, "%h", Nothing, interval) Then
    Console.WriteLine(interval.ToString("c"))
Else
    Console.WriteLine("Unable to convert '{0}' to a time interval",
                    value)
End If
' The example displays the following output:
'      08:00:00
```

The following example illustrates the use of the "h" custom format specifier.

```
TimeSpan ts1 = new TimeSpan(14, 3, 17);
Console.WriteLine(ts1.ToString(@"d\.h\:mm\:ss"));

TimeSpan ts2 = new TimeSpan(3, 4, 3, 17);
Console.WriteLine(ts2.ToString(@"d\.h\:mm\:ss"));
// The example displays the following output:
//      0.14:03:17
//      3.4:03:17
```

```

Dim ts1 As New TimeSpan(14, 3, 17)
Console.WriteLine(ts1.ToString("d\.h\:mm\:ss"))

Dim ts2 As New TimeSpan(3, 4, 3, 17)
Console.WriteLine(ts2.ToString("d\.h\:mm\:ss"))
' The example displays the following output:
'      0.14:03:17
'      3.4:03:17

```

[Back to table](#)

The "hh" custom format specifier

The "hh" custom format specifier outputs the value of the [TimeSpan.Hours](#) property, which represents the number of whole hours in the time interval that isn't counted as part of its day component. For values from 0 through 9, the output string includes a leading zero.

Ordinarily, in a parsing operation, an input string that includes only a single number is interpreted as the number of days. You can use the "hh" custom format specifier instead to interpret the numeric string as the number of hours. The following example provides an illustration.

```

string value = "08";
TimeSpan interval;
if (TimeSpan.TryParseExact(value, "hh", null, out interval))
    Console.WriteLine(interval.ToString("c"));
else
    Console.WriteLine("Unable to convert '{0}' to a time interval",
                     value);
// The example displays the following output:
//      08:00:00

```

```

Dim value As String = "08"
Dim interval As TimeSpan
If TimeSpan.TryParseExact(value, "hh", Nothing, interval) Then
    Console.WriteLine(interval.ToString("c"))
Else
    Console.WriteLine("Unable to convert '{0}' to a time interval",
                     value)
End If
' The example displays the following output:
'      08:00:00

```

The following example illustrates the use of the "hh" custom format specifier.

```

TimeSpan ts1 = new TimeSpan(14, 3, 17);
Console.WriteLine(ts1.ToString(@"d\.hh\:mm\:ss"));

TimeSpan ts2 = new TimeSpan(3, 4, 3, 17);
Console.WriteLine(ts2.ToString(@"d\.hh\:mm\:ss"));
// The example displays the following output:
//      0.14:03:17
//      3.04:03:17

```

```

Dim ts1 As New TimeSpan(14, 3, 17)
Console.WriteLine(ts1.ToString("d\:hh\:mm\:ss"))

Dim ts2 As New TimeSpan(3, 4, 3, 17)
Console.WriteLine(ts2.ToString("d\:hh\:mm\:ss"))
' The example displays the following output:
'      0.14:03:17
'      3.04:03:17

```

[Back to table](#)

The "m" custom format specifier

The "m" custom format specifier outputs the value of the [TimeSpan.Minutes](#) property, which represents the number of whole minutes in the time interval that isn't counted as part of its day component. It returns a one-digit string value if the value of the [TimeSpan.Minutes](#) property is 0 through 9, and it returns a two-digit string value if the value of the [TimeSpan.Minutes](#) property ranges from 10 to 59.

If the "m" custom format specifier is used alone, specify "%m" so that it isn't misinterpreted as a standard format string. The following example provides an illustration.

```

TimeSpan ts = new TimeSpan(3, 42, 0);
Console.WriteLine("{0:%h} hours {0:%m} minutes", ts);
// The example displays the following output:
//      3 hours 42 minutes

```

```

Dim ts As New TimeSpan(3, 42, 0)
Console.WriteLine("{0:%h} hours {0:%m} minutes", ts)
' The example displays the following output:
'      3 hours 42 minutes

```

Ordinarily, in a parsing operation, an input string that includes only a single number is interpreted as the number of days. You can use the "%m" custom format specifier instead to interpret the numeric string as the number of minutes. The following example provides an illustration.

```

string value = "3";
TimeSpan interval;
if (TimeSpan.TryParseExact(value, "%m", null, out interval))
    Console.WriteLine(interval.ToString("c"));
else
    Console.WriteLine("Unable to convert '{0}' to a time interval",
                    value);
// The example displays the following output:
//      00:03:00

```

```

Dim value As String = "3"
Dim interval As TimeSpan
If TimeSpan.TryParseExact(value, "%m", Nothing, interval) Then
    Console.WriteLine(interval.ToString("c"))
Else
    Console.WriteLine("Unable to convert '{0}' to a time interval",
                    value)
End If
' The example displays the following output:
'      00:03:00

```

The following example illustrates the use of the "m" custom format specifier.

```
TimeSpan ts1 = new TimeSpan(0, 6, 32);
Console.WriteLine("{0:m\\:ss} minutes", ts1);

TimeSpan ts2 = new TimeSpan(3, 4, 3, 17);
Console.WriteLine("Elapsed time: {0:m\\:ss}", ts2);
// The example displays the following output:
//      6:32 minutes
//      Elapsed time: 18:44
```

```
Dim ts1 As New TimeSpan(0, 6, 32)
Console.WriteLine("{0:m\\:ss} minutes", ts1)

Dim ts2 As New TimeSpan(0, 18, 44)
Console.WriteLine("Elapsed time: {0:m\\:ss}", ts2)
' The example displays the following output:
'      6:32 minutes
'      Elapsed time: 18:44
```

[Back to table](#)

The "mm" custom format specifier

The "mm" custom format specifier outputs the value of the [TimeSpan.Minutes](#) property, which represents the number of whole minutes in the time interval that isn't included as part of its hours or days component. For values from 0 through 9, the output string includes a leading zero.

Ordinarily, in a parsing operation, an input string that includes only a single number is interpreted as the number of days. You can use the "mm" custom format specifier instead to interpret the numeric string as the number of minutes. The following example provides an illustration.

```
string value = "07";
TimeSpan interval;
if (TimeSpan.TryParseExact(value, "mm", null, out interval))
    Console.WriteLine(interval.ToString("c"));
else
    Console.WriteLine("Unable to convert '{0}' to a time interval",
                    value);
// The example displays the following output:
//      00:07:00
```

```
Dim value As String = "05"
Dim interval As TimeSpan
If TimeSpan.TryParseExact(value, "mm", Nothing, interval) Then
    Console.WriteLine(interval.ToString("c"))
Else
    Console.WriteLine("Unable to convert '{0}' to a time interval",
                    value)
End If
' The example displays the following output:
'      00:05:00
```

The following example illustrates the use of the "mm" custom format specifier.

```

TimeSpan departTime = new TimeSpan(11, 12, 00);
TimeSpan arriveTime = new TimeSpan(16, 28, 00);
Console.WriteLine("Travel time: {0:hh\\\:mm}",
                  arriveTime - departTime);
// The example displays the following output:
//      Travel time: 05:16

```

```

Dim departTime As New TimeSpan(11, 12, 00)
Dim arriveTime As New TimeSpan(16, 28, 00)
Console.WriteLine("Travel time: {0:hh\\\:mm}",
                  arriveTime - departTime)
' The example displays the following output:
'      Travel time: 05:16

```

[Back to table](#)

The "s" custom format specifier

The "s" custom format specifier outputs the value of the [TimeSpan.Seconds](#) property, which represents the number of whole seconds in the time interval that isn't included as part of its hours, days, or minutes component. It returns a one-digit string value if the value of the [TimeSpan.Seconds](#) property is 0 through 9, and it returns a two-digit string value if the value of the [TimeSpan.Seconds](#) property ranges from 10 to 59.

If the "s" custom format specifier is used alone, specify "%s" so that it isn't misinterpreted as a standard format string. The following example provides an illustration.

```

TimeSpan ts = TimeSpan.FromSeconds(12.465);
Console.WriteLine(ts.ToString("%s"));
// The example displays the following output:
//      12

```

```

Dim ts As TimeSpan = TimeSpan.FromSeconds(12.465)
Console.WriteLine(ts.ToString("%s"))
' The example displays the following output:
'      12

```

Ordinarily, in a parsing operation, an input string that includes only a single number is interpreted as the number of days. You can use the "%s" custom format specifier instead to interpret the numeric string as the number of seconds. The following example provides an illustration.

```

string value = "9";
TimeSpan interval;
if (TimeSpan.TryParseExact(value, "%s", null, out interval))
    Console.WriteLine(interval.ToString("c"));
else
    Console.WriteLine("Unable to convert '{0}' to a time interval",
                     value);
// The example displays the following output:
//      00:00:09

```

```

Dim value As String = "9"
Dim interval As TimeSpan
If TimeSpan.TryParseExact(value, "%s", Nothing, interval) Then
    Console.WriteLine(interval.ToString("c"))
Else
    Console.WriteLine("Unable to convert '{0}' to a time interval",
                     value)
End If
' The example displays the following output:
'     00:00:09

```

The following example illustrates the use of the "s" custom format specifier.

```

TimeSpan startTime = new TimeSpan(0, 12, 30, 15, 0);
TimeSpan endTime = new TimeSpan(0, 12, 30, 21, 3);
Console.WriteLine(@"Elapsed Time: {0:s\:ffff} seconds",
                  endTime - startTime);
// The example displays the following output:
//     Elapsed Time: 6:003 seconds

```

```

Dim startTime As New TimeSpan(0, 12, 30, 15, 0)
Dim endTime As New TimeSpan(0, 12, 30, 21, 3)
Console.WriteLine("Elapsed Time: {0:s\:ffff} seconds",
                  endTime - startTime)
' The example displays the following output:
'     Elapsed Time: 6:003 seconds

```

[Back to table](#)

The "ss" custom format specifier

The "ss" custom format specifier outputs the value of the [TimeSpan.Seconds](#) property, which represents the number of whole seconds in the time interval that isn't included as part of its hours, days, or minutes component. For values from 0 through 9, the output string includes a leading zero.

Ordinarily, in a parsing operation, an input string that includes only a single number is interpreted as the number of days. You can use the "ss" custom format specifier instead to interpret the numeric string as the number of seconds. The following example provides an illustration.

```

string[] values = { "49", "9", "06" };
TimeSpan interval;
foreach (string value in values)
{
    if (TimeSpan.TryParseExact(value, "ss", null, out interval))
        Console.WriteLine(interval.ToString("c"));
    else
        Console.WriteLine("Unable to convert '{0}' to a time interval",
                         value);
}
// The example displays the following output:
//     00:00:49
//     Unable to convert '9' to a time interval
//     00:00:06

```

```

Dim values() As String = {"49", "9", "06"}
Dim interval As TimeSpan
For Each value As String In values
    If TimeSpan.TryParseExact(value, "ss", Nothing, interval) Then
        Console.WriteLine(interval.ToString("c"))
    Else
        Console.WriteLine("Unable to convert '{0}' to a time interval",
                         value)
    End If
Next
' The example displays the following output:
'      00:00:49
'      Unable to convert '9' to a time interval
'      00:00:06

```

The following example illustrates the use of the "ss" custom format specifier.

```

TimeSpan interval1 = TimeSpan.FromSeconds(12.60);
Console.WriteLine(interval1.ToString("ss\\.fff"));

TimeSpan interval2 = TimeSpan.FromSeconds(6.485);
Console.WriteLine(interval2.ToString("ss\\.fff"));
// The example displays the following output:
//      12.600
//      06.485

```

```

Dim interval1 As TimeSpan = TimeSpan.FromSeconds(12.60)
Console.WriteLine(interval1.ToString("ss\\.fff"))
Dim interval2 As TimeSpan = TimeSpan.FromSeconds(6.485)
Console.WriteLine(interval2.ToString("ss\\.fff"))
' The example displays the following output:
'      12.600
'      06.485

```

[Back to table](#)

The "f" custom format specifier

The "f" custom format specifier outputs the tenths of a second in a time interval. In a formatting operation, any remaining fractional digits are truncated. In a parsing operation that calls the [TimeSpan.ParseExact](#) or [TimeSpan.TryParseExact](#) method, the input string must contain exactly one fractional digit.

If the "f" custom format specifier is used alone, specify "%f" so that it isn't misinterpreted as a standard format string.

The following example uses the "f" custom format specifier to display the tenths of a second in a [TimeSpan](#) value. "f" is used first as the only format specifier, and then combined with the "s" specifier in a custom format string.

```

TimeSpan ts = new TimeSpan(1003498765432);
string fmt;
Console.WriteLine(ts.ToString("c"));
Console.WriteLine();

for (int ctr = 1; ctr <= 7; ctr++) {
    fmt = new String('f', ctr);
    if (fmt.Length == 1) fmt = "%" + fmt;
    Console.WriteLine("{0,10}: {1:" + fmt + "}", fmt, ts);
}
Console.WriteLine();

for (int ctr = 1; ctr <= 7; ctr++) {
    fmt = new String('f', ctr);
    Console.WriteLine("{0,10}: {1:s\\.+" + fmt + "}", "s\\." + fmt, ts);
}

// The example displays the following output:
//          %f: 8
//          ff: 87
//          fff: 876
//          ffff: 8765
//          fffff: 87654
//          ffffff: 876543
//          fffffff: 8765432
//
//          s\\.f: 29.8
//          s\\.ff: 29.87
//          s\\.fff: 29.876
//          s\\.ffff: 29.8765
//          s\\.fffff: 29.87654
//          s\\.fffffff: 29.876543
//          s\\.fffffff: 29.8765432

```

```

Dim ts As New TimeSpan(1003498765432)
Dim fmt As String
Console.WriteLine(ts.ToString("c"))
Console.WriteLine()

For ctr = 1 To 7
    fmt = New String("f"c, ctr)
    If fmt.Length = 1 Then fmt = "%" + fmt
    Console.WriteLine("{0,10}: {1:" + fmt + "}", fmt, ts)
Next
Console.WriteLine()

For ctr = 1 To 7
    fmt = New String("f"c, ctr)
    Console.WriteLine("{0,10}: {1:s\\.+" + fmt + "}", "s\\." + fmt, ts)
Next

' The example displays the following output:
'          %f: 8
'          ff: 87
'          fff: 876
'          ffff: 8765
'          fffff: 87654
'          ffffff: 876543
'          fffffff: 8765432
'
'          s\\.f: 29.8
'          s\\.ff: 29.87
'          s\\.fff: 29.876
'          s\\.ffff: 29.8765
'          s\\.fffff: 29.87654
'          s\\.fffffff: 29.876543
'          s\\.fffffff: 29.8765432

```

[Back to table](#)

The "ff" custom format specifier

The "ff" custom format specifier outputs the hundredths of a second in a time interval. In a formatting operation, any remaining fractional digits are truncated. In a parsing operation that calls the [TimeSpan.ParseExact](#) or [TimeSpan.TryParseExact](#) method, the input string must contain exactly two fractional digits.

The following example uses the "ff" custom format specifier to display the hundredths of a second in a [TimeSpan](#) value. "ff" is used first as the only format specifier, and then combined with the "s" specifier in a custom format string.

```
TimeSpan ts = new TimeSpan(1003498765432);
string fmt;
Console.WriteLine(ts.ToString("c"));
Console.WriteLine();

for (int ctr = 1; ctr <= 7; ctr++) {
    fmt = new String('f', ctr);
    if (fmt.Length == 1) fmt = "%" + fmt;
    Console.WriteLine("{0,10}: {1:" + fmt + "}", fmt, ts);
}
Console.WriteLine();

for (int ctr = 1; ctr <= 7; ctr++) {
    fmt = new String('f', ctr);
    Console.WriteLine("{0,10}: {1:s\\.\" + fmt + "}", "s\\.\" + fmt, ts);
}
// The example displays the following output:
//          %f: 8
//          ff: 87
//         fff: 876
//        ffff: 8765
//       fffff: 87654
//      ffffff: 876543
//     fffffff: 8765432
//
//          s\\.f: 29.8
//          s\\.ff: 29.87
//         s\\.fff: 29.876
//        s\\.ffff: 29.8765
//       s\\.fffff: 29.87654
//      s\\.fffffff: 29.876543
//     s\\.fffffff: 29.8765432
```

```

Dim ts As New TimeSpan(1003498765432)
Dim fmt As String
Console.WriteLine(ts.ToString("c"))
Console.WriteLine()

For ctr = 1 To 7
    fmt = New String("f"c, ctr)
    If fmt.Length = 1 Then fmt = "%" + fmt
    Console.WriteLine("{0,10}: {1:" + fmt + "}", fmt, ts)
Next
Console.WriteLine()

For ctr = 1 To 7
    fmt = New String("f"c, ctr)
    Console.WriteLine("{0,10}: {1:s\." + fmt + "}", "s\." + fmt, ts)
Next
' The example displays the following output:
'      %f: 8
'      ff: 87
'      fff: 876
'      ffff: 8765
'      fffff: 87654
'      ffffff: 876543
'      ffffffff: 8765432
'
'      s\.f: 29.8
'      s\.ff: 29.87
'      s\.fff: 29.876
'      s\.ffff: 29.8765
'      s\.fffff: 29.87654
'      s\.fffffff: 29.876543
'      s\.fffffff: 29.8765432

```

[Back to table](#)

The "fff" custom format specifier

The "fff" custom format specifier (with three "f" characters) outputs the milliseconds in a time interval. In a formatting operation, any remaining fractional digits are truncated. In a parsing operation that calls the [TimeSpan.ParseExact](#) or [TimeSpan.TryParseExact](#) method, the input string must contain exactly three fractional digits.

The following example uses the "fff" custom format specifier to display the milliseconds in a [TimeSpan](#) value. "fff" is used first as the only format specifier, and then combined with the "s" specifier in a custom format string.

```

TimeSpan ts = new TimeSpan(1003498765432);
string fmt;
Console.WriteLine(ts.ToString("c"));
Console.WriteLine();

for (int ctr = 1; ctr <= 7; ctr++) {
    fmt = new String('f', ctr);
    if (fmt.Length == 1) fmt = "%" + fmt;
    Console.WriteLine("{0,10}: {1:" + fmt + "}", fmt, ts);
}
Console.WriteLine();

for (int ctr = 1; ctr <= 7; ctr++) {
    fmt = new String('f', ctr);
    Console.WriteLine("{0,10}: {1:s\\.+" + fmt + "}", "s\\." + fmt, ts);
}

// The example displays the following output:
//          %f: 8
//          ff: 87
//          fff: 876
//          ffff: 8765
//          fffff: 87654
//          ffffff: 876543
//          fffffff: 8765432
//
//          s\\.f: 29.8
//          s\\.ff: 29.87
//          s\\.fff: 29.876
//          s\\.ffff: 29.8765
//          s\\.fffff: 29.87654
//          s\\.fffffff: 29.876543
//          s\\.fffffff: 29.8765432

```

```

Dim ts As New TimeSpan(1003498765432)
Dim fmt As String
Console.WriteLine(ts.ToString("c"))
Console.WriteLine()

For ctr = 1 To 7
    fmt = New String("f"c, ctr)
    If fmt.Length = 1 Then fmt = "%" + fmt
    Console.WriteLine("{0,10}: {1:" + fmt + "}", fmt, ts)
Next
Console.WriteLine()

For ctr = 1 To 7
    fmt = New String("f"c, ctr)
    Console.WriteLine("{0,10}: {1:s\\.+" + fmt + "}", "s\\." + fmt, ts)
Next

' The example displays the following output:
'          %f: 8
'          ff: 87
'          fff: 876
'          ffff: 8765
'          fffff: 87654
'          ffffff: 876543
'          fffffff: 8765432
'
'          s\\.f: 29.8
'          s\\.ff: 29.87
'          s\\.fff: 29.876
'          s\\.ffff: 29.8765
'          s\\.fffff: 29.87654
'          s\\.fffffff: 29.876543
'          s\\.fffffff: 29.8765432

```

[Back to table](#)

The "ffff" custom format specifier

The "ffff" custom format specifier (with four "f" characters) outputs the ten-thousandths of a second in a time interval. In a formatting operation, any remaining fractional digits are truncated. In a parsing operation that calls the [TimeSpan.ParseExact](#) or [TimeSpan.TryParseExact](#) method, the input string must contain exactly four fractional digits.

The following example uses the "ffff" custom format specifier to display the ten-thousandths of a second in a [TimeSpan](#) value. "ffff" is used first as the only format specifier, and then combined with the "s" specifier in a custom format string.

```
TimeSpan ts = new TimeSpan(1003498765432);
string fmt;
Console.WriteLine(ts.ToString("c"));
Console.WriteLine();

for (int ctr = 1; ctr <= 7; ctr++) {
    fmt = new String('f', ctr);
    if (fmt.Length == 1) fmt = "%" + fmt;
    Console.WriteLine("{0,10}: {1:" + fmt + "}", fmt, ts);
}
Console.WriteLine();

for (int ctr = 1; ctr <= 7; ctr++) {
    fmt = new String('f', ctr);
    Console.WriteLine("{0,10}: {1:s\\\" + fmt + "}", "s\\\" + fmt, ts);
}
// The example displays the following output:
//      %f: 8
//      ff: 87
//      fff: 876
//      ffff: 8765
//      fffff: 87654
//      ffffff: 876543
//      ffffffff: 8765432
//
//      s\.f: 29.8
//      s\.ff: 29.87
//      s\.fff: 29.876
//      s\.ffff: 29.8765
//      s\.fffff: 29.87654
//      s\.fffffff: 29.876543
//      s\.fffffff: 29.8765432
```

```

Dim ts As New TimeSpan(1003498765432)
Dim fmt As String
Console.WriteLine(ts.ToString("c"))
Console.WriteLine()

For ctr = 1 To 7
    fmt = New String("f"c, ctr)
    If fmt.Length = 1 Then fmt = "%" + fmt
    Console.WriteLine("{0,10}: {1:" + fmt + "}", fmt, ts)
Next
Console.WriteLine()

For ctr = 1 To 7
    fmt = New String("f"c, ctr)
    Console.WriteLine("{0,10}: {1:s\." + fmt + "}", "s\." + fmt, ts)
Next
' The example displays the following output:
'      %f: 8
'      ff: 87
'      fff: 876
'      ffff: 8765
'      fffff: 87654
'      ffffff: 876543
'      ffffffff: 8765432
'
'      s\.f: 29.8
'      s\.ff: 29.87
'      s\.fff: 29.876
'      s\.ffff: 29.8765
'      s\.fffff: 29.87654
'      s\.fffffff: 29.876543
'      s\.fffffff: 29.8765432

```

[Back to table](#)

The "fffff" custom format specifier

The "fffff" custom format specifier (with five "f" characters) outputs the hundred-thousandths of a second in a time interval. In a formatting operation, any remaining fractional digits are truncated. In a parsing operation that calls the [TimeSpan.ParseExact](#) or [TimeSpan.TryParseExact](#) method, the input string must contain exactly five fractional digits.

The following example uses the "fffff" custom format specifier to display the hundred-thousandths of a second in a [TimeSpan](#) value. "fffff" is used first as the only format specifier, and then combined with the "s" specifier in a custom format string.

```

TimeSpan ts = new TimeSpan(1003498765432);
string fmt;
Console.WriteLine(ts.ToString("c"));
Console.WriteLine();

for (int ctr = 1; ctr <= 7; ctr++) {
    fmt = new String('f', ctr);
    if (fmt.Length == 1) fmt = "%" + fmt;
    Console.WriteLine("{0,10}: {1:" + fmt + "}", fmt, ts);
}
Console.WriteLine();

for (int ctr = 1; ctr <= 7; ctr++) {
    fmt = new String('f', ctr);
    Console.WriteLine("{0,10}: {1:s\\.+" + fmt + "}", "s\\." + fmt, ts);
}

// The example displays the following output:
//          %f: 8
//          ff: 87
//          fff: 876
//          ffff: 8765
//          fffff: 87654
//          ffffff: 876543
//          fffffff: 8765432
//
//          s\\.f: 29.8
//          s\\.ff: 29.87
//          s\\.fff: 29.876
//          s\\.ffff: 29.8765
//          s\\.fffff: 29.87654
//          s\\.fffffff: 29.876543
//          s\\.fffffff: 29.8765432

```

```

Dim ts As New TimeSpan(1003498765432)
Dim fmt As String
Console.WriteLine(ts.ToString("c"))
Console.WriteLine()

For ctr = 1 To 7
    fmt = New String("f"c, ctr)
    If fmt.Length = 1 Then fmt = "%" + fmt
    Console.WriteLine("{0,10}: {1:" + fmt + "}", fmt, ts)
Next
Console.WriteLine()

For ctr = 1 To 7
    fmt = New String("f"c, ctr)
    Console.WriteLine("{0,10}: {1:s\\.+" + fmt + "}", "s\\." + fmt, ts)
Next

' The example displays the following output:
'          %f: 8
'          ff: 87
'          fff: 876
'          ffff: 8765
'          fffff: 87654
'          ffffff: 876543
'          fffffff: 8765432
'
'          s\\.f: 29.8
'          s\\.ff: 29.87
'          s\\.fff: 29.876
'          s\\.ffff: 29.8765
'          s\\.fffff: 29.87654
'          s\\.fffffff: 29.876543
'          s\\.fffffff: 29.8765432

```

[Back to table](#)

The "fffffff" custom format specifier

The "fffffff" custom format specifier (with six "f" characters) outputs the millionths of a second in a time interval. In a formatting operation, any remaining fractional digits are truncated. In a parsing operation that calls the [TimeSpan.ParseExact](#) or [TimeSpan.TryParseExact](#) method, the input string must contain exactly six fractional digits.

The following example uses the "fffffff" custom format specifier to display the millionths of a second in a [TimeSpan](#) value. It is used first as the only format specifier, and then combined with the "s" specifier in a custom format string.

```
TimeSpan ts = new TimeSpan(1003498765432);
string fmt;
Console.WriteLine(ts.ToString("c"));
Console.WriteLine();

for (int ctr = 1; ctr <= 7; ctr++) {
    fmt = new String('f', ctr);
    if (fmt.Length == 1) fmt = "%" + fmt;
    Console.WriteLine("{0,10}: {1:" + fmt + "}", fmt, ts);
}
Console.WriteLine();

for (int ctr = 1; ctr <= 7; ctr++) {
    fmt = new String('f', ctr);
    Console.WriteLine("{0,10}: {1:s\\\" + fmt + "}", "s\\\" + fmt, ts);
}
// The example displays the following output:
//          %f: 8
//          ff: 87
//         fff: 876
//        ffff: 8765
//       fffff: 87654
//      ffffff: 876543
//     fffffff: 8765432
//
//           s\.f: 29.8
//           s\.ff: 29.87
//          s\.fff: 29.876
//         s\.ffff: 29.8765
//        s\.fffff: 29.87654
//       s\.fffffff: 29.876543
//      s\.fffffff: 29.8765432
```

```

Dim ts As New TimeSpan(1003498765432)
Dim fmt As String
Console.WriteLine(ts.ToString("c"))
Console.WriteLine()

For ctr = 1 To 7
    fmt = New String("f"c, ctr)
    If fmt.Length = 1 Then fmt = "%" + fmt
    Console.WriteLine("{0,10}: {1:" + fmt + "}", fmt, ts)
Next
Console.WriteLine()

For ctr = 1 To 7
    fmt = New String("f"c, ctr)
    Console.WriteLine("{0,10}: {1:s\." + fmt + "}", "s\." + fmt, ts)
Next
' The example displays the following output:
'      %f: 8
'      ff: 87
'      fff: 876
'      ffff: 8765
'      fffff: 87654
'      ffffff: 876543
'      ffffffff: 8765432
'
'      s\.f: 29.8
'      s\.ff: 29.87
'      s\.fff: 29.876
'      s\.ffff: 29.8765
'      s\.fffff: 29.87654
'      s\.fffffff: 29.876543
'      s\.fffffff: 29.8765432

```

[Back to table](#)

The "fffffff" custom format specifier

The "fffffff" custom format specifier (with seven "f" characters) outputs the ten-millionths of a second (or the fractional number of ticks) in a time interval. In a parsing operation that calls the [TimeSpan.ParseExact](#) or [TimeSpan.TryParseExact](#) method, the input string must contain exactly seven fractional digits.

The following example uses the "fffffff" custom format specifier to display the fractional number of ticks in a [TimeSpan](#) value. It is used first as the only format specifier, and then combined with the "s" specifier in a custom format string.

```

TimeSpan ts = new TimeSpan(1003498765432);
string fmt;
Console.WriteLine(ts.ToString("c"));
Console.WriteLine();

for (int ctr = 1; ctr <= 7; ctr++) {
    fmt = new String('f', ctr);
    if (fmt.Length == 1) fmt = "%" + fmt;
    Console.WriteLine("{0,10}: {1:" + fmt + "}", fmt, ts);
}
Console.WriteLine();

for (int ctr = 1; ctr <= 7; ctr++) {
    fmt = new String('f', ctr);
    Console.WriteLine("{0,10}: {1:s\\.+" + fmt + "}", "s\\." + fmt, ts);
}

// The example displays the following output:
//          %f: 8
//          ff: 87
//          fff: 876
//          ffff: 8765
//          fffff: 87654
//          ffffff: 876543
//          fffffff: 8765432
//
//          s\\.f: 29.8
//          s\\.ff: 29.87
//          s\\.fff: 29.876
//          s\\.ffff: 29.8765
//          s\\.fffff: 29.87654
//          s\\.fffffff: 29.876543
//          s\\.fffffff: 29.8765432

```

```

Dim ts As New TimeSpan(1003498765432)
Dim fmt As String
Console.WriteLine(ts.ToString("c"))
Console.WriteLine()

For ctr = 1 To 7
    fmt = New String("f"c, ctr)
    If fmt.Length = 1 Then fmt = "%" + fmt
    Console.WriteLine("{0,10}: {1:" + fmt + "}", fmt, ts)
Next
Console.WriteLine()

For ctr = 1 To 7
    fmt = New String("f"c, ctr)
    Console.WriteLine("{0,10}: {1:s\\.+" + fmt + "}", "s\\." + fmt, ts)
Next

' The example displays the following output:
'          %f: 8
'          ff: 87
'          fff: 876
'          ffff: 8765
'          fffff: 87654
'          ffffff: 876543
'          fffffff: 8765432
'
'          s\\.f: 29.8
'          s\\.ff: 29.87
'          s\\.fff: 29.876
'          s\\.ffff: 29.8765
'          s\\.fffff: 29.87654
'          s\\.fffffff: 29.876543
'          s\\.fffffff: 29.8765432

```

[Back to table](#)

The "F" custom format specifier

The "F" custom format specifier outputs the tenths of a second in a time interval. In a formatting operation, any remaining fractional digits are truncated. If the value of the time interval's tenths of a second is zero, it isn't included in the result string. In a parsing operation that calls the [TimeSpan.ParseExact](#) or [TimeSpan.TryParseExact](#) method, the presence of the tenths of a second digit is optional.

If the "F" custom format specifier is used alone, specify "%F" so that it isn't misinterpreted as a standard format string.

The following example uses the "F" custom format specifier to display the tenths of a second in a [TimeSpan](#) value. It also uses this custom format specifier in a parsing operation.

```
Console.WriteLine("Formatting:");
TimeSpan ts1 = TimeSpan.Parse("0:0:3.669");
Console.WriteLine("{0} ('%F') --> {0:%F}", ts1);

TimeSpan ts2 = TimeSpan.Parse("0:0:3.091");
Console.WriteLine("{0} ('ss\\.F') --> {0:ss\\.F}", ts2);
Console.WriteLine();

Console.WriteLine("Parsing:");
string[] inputs = { "0:0:03.", "0:0:03.1", "0:0:03.12" };
string fmt = @"h\:m\:ss\.F";
TimeSpan ts3;

foreach (string input in inputs) {
    if (TimeSpan.TryParseExact(input, fmt, null, out ts3))
        Console.WriteLine("{0} ('{1}') --> {2}", input, fmt, ts3);
    else
        Console.WriteLine("Cannot parse {0} with '{1}'.",
                         input, fmt);
}
// The example displays the following output:
//      Formatting:
//      00:00:03.6690000 ('%F') --> 6
//      00:00:03.0910000 ('ss\\.F') --> 03.
//
//      Parsing:
//      0:0:03. ('h\:m\:ss\.F') --> 00:00:03
//      0:0:03.1 ('h\:m\:ss\.F') --> 00:00:03.1000000
//      Cannot parse 0:0:03.12 with 'h\:m\:ss\.F'.
```

```

Console.WriteLine("Formatting:")
Dim ts1 As TimeSpan = TimeSpan.Parse("0:0:3.669")
Console.WriteLine("{0} ('%F') --> {0:%F}", ts1)

Dim ts2 As TimeSpan = TimeSpan.Parse("0:0:3.091")
Console.WriteLine("{0} ('ss\.F') --> {0:ss\.F}", ts2)
Console.WriteLine()

Console.WriteLine("Parsing:")
Dim inputs() As String = {"0:0:03.", "0:0:03.1", "0:0:03.12"}
Dim fmt As String = "h\:m\:ss\.F"
Dim ts3 As TimeSpan

For Each input As String In inputs
    If TimeSpan.TryParseExact(input, fmt, Nothing, ts3)
        Console.WriteLine("{0} ('{1}') --> {2}", input, fmt, ts3)
    Else
        Console.WriteLine("Cannot parse {0} with '{1}'.",
                          input, fmt)
    End If
Next
' The example displays the following output:
'     Formatting:
'     00:00:03.6690000 ('%F') --> 6
'     00:00:03.0910000 ('ss\.F') --> 03.
'
'     Parsing:
'     0:0:03. ('h\:m\:ss\.F') --> 00:00:03
'     0:0:03.1 ('h\:m\:ss\.F') --> 00:00:03.100000
'     Cannot parse 0:0:03.12 with 'h\:m\:ss\.F'.

```

[Back to table](#)

The "FF" custom format specifier

The "FF" custom format specifier outputs the hundredths of a second in a time interval. In a formatting operation, any remaining fractional digits are truncated. If there are any trailing fractional zeros, they aren't included in the result string. In a parsing operation that calls the [TimeSpan.ParseExact](#) or [TimeSpan.TryParseExact](#) method, the presence of the tenths and hundredths of a second digit is optional.

The following example uses the "FF" custom format specifier to display the hundredths of a second in a [TimeSpan](#) value. It also uses this custom format specifier in a parsing operation.

```

Console.WriteLine("Formatting:");
TimeSpan ts1 = TimeSpan.Parse("0:0:3.697");
Console.WriteLine("{0} ('FF') --> {0:FF}", ts1);

TimeSpan ts2 = TimeSpan.Parse("0:0:3.809");
Console.WriteLine("{0} ('ss\\.FF') --> {0:ss\\.FF}", ts2);
Console.WriteLine();

Console.WriteLine("Parsing:");
string[] inputs = { "0:0:03.", "0:0:03.1", "0:0:03.127" };
string fmt = @"h\:m\:ss\.FF";
TimeSpan ts3;

foreach (string input in inputs) {
    if (TimeSpan.TryParseExact(input, fmt, null, out ts3))
        Console.WriteLine("{0} ('{1}') --> {2}", input, fmt, ts3);
    else
        Console.WriteLine("Cannot parse {0} with '{1}'.",
                         input, fmt);
}
// The example displays the following output:
//      Formatting:
//      00:00:03.6970000 ('FF') --> 69
//      00:00:03.8090000 ('ss\.FF') --> 03.8
//
//      Parsing:
//      0:0:03. ('h\:m\:ss\.FF') --> 00:00:03
//      0:0:03.1 ('h\:m\:ss\.FF') --> 00:00:03.1000000
//      Cannot parse 0:0:03.127 with 'h\:m\:ss\.FF'.

```

```

Console.WriteLine("Formatting:")
Dim ts1 As TimeSpan = TimeSpan.Parse("0:0:3.697")
Console.WriteLine("{0} ('FF') --> {0:FF}", ts1)

Dim ts2 As TimeSpan = TimeSpan.Parse("0:0:3.809")
Console.WriteLine("{0} ('ss\\.FF') --> {0:ss\\.FF}", ts2)
Console.WriteLine()

Console.WriteLine("Parsing:")
Dim inputs() As String = {"0:0:03.", "0:0:03.1", "0:0:03.127"}
Dim fmt As String = "h\:m\:ss\.FF"
Dim ts3 As TimeSpan

For Each input As String In inputs
    If TimeSpan.TryParseExact(input, fmt, Nothing, ts3)
        Console.WriteLine("{0} ('{1}') --> {2}", input, fmt, ts3)
    Else
        Console.WriteLine("Cannot parse {0} with '{1}'.",
                         input, fmt)
    End If
Next
' The example displays the following output:
'      Formatting:
'      00:00:03.6970000 ('FF') --> 69
'      00:00:03.8090000 ('ss\.FF') --> 03.8
'
'      Parsing:
'      0:0:03. ('h\:m\:ss\.FF') --> 00:00:03
'      0:0:03.1 ('h\:m\:ss\.FF') --> 00:00:03.1000000
'      Cannot parse 0:0:03.127 with 'h\:m\:ss\.FF'.

```

[Back to table](#)

The "FFF" custom format specifier

The "FFF" custom format specifier (with three "F" characters) outputs the milliseconds in a time interval. In a formatting operation, any remaining fractional digits are truncated. If there are any trailing fractional zeros, they aren't included in the result string. In a parsing operation that calls the [TimeSpan.ParseExact](#) or [TimeSpan.TryParseExact](#) method, the presence of the tenths, hundredths, and thousandths of a second digit is optional.

The following example uses the "FFF" custom format specifier to display the thousandths of a second in a [TimeSpan](#) value. It also uses this custom format specifier in a parsing operation.

```
Console.WriteLine("Formatting:");
TimeSpan ts1 = TimeSpan.Parse("0:0:3.6974");
Console.WriteLine("{0} ('FFF') --> {0:FFF}", ts1);

TimeSpan ts2 = TimeSpan.Parse("0:0:3.8009");
Console.WriteLine("{0} ('ss\\.FFF') --> {0:ss\\.FFF}", ts2);
Console.WriteLine();

Console.WriteLine("Parsing:");
string[] inputs = { "0:0:03.", "0:0:03.12", "0:0:03.1279" };
string fmt = @"h\:m\:ss\.FFF";
TimeSpan ts3;

foreach (string input in inputs) {
    if (TimeSpan.TryParseExact(input, fmt, null, out ts3))
        Console.WriteLine("{0} ('{1}') --> {2}", input, fmt, ts3);
    else
        Console.WriteLine("Cannot parse {0} with '{1}'.",
                         input, fmt);
}
// The example displays the following output:
//      Formatting:
//      00:00:03.6974000 ('FFF') --> 697
//      00:00:03.8009000 ('ss\\.FFF') --> 03.8
//
//      Parsing:
//      0:0:03. ('h\:m\:ss\\.FFF') --> 00:00:03
//      0:0:03.12 ('h\:m\:ss\\.FFF') --> 00:00:03.1200000
//      Cannot parse 0:0:03.1279 with 'h\:m\:ss\\.FFF'.
```

```

Console.WriteLine("Formatting:")
Dim ts1 As TimeSpan = TimeSpan.Parse("0:0:3.6974")
Console.WriteLine("{0} ('FFF') --> {0:FFF}", ts1)

Dim ts2 As TimeSpan = TimeSpan.Parse("0:0:3.8009")
Console.WriteLine("{0} ('ss\FFF') --> {0:ss\.FFF}", ts2)
Console.WriteLine()

Console.WriteLine("Parsing:")
Dim inputs() As String = {"0:0:03.", "0:0:03.12", "0:0:03.1279"}
Dim fmt As String = "h\:m\:ss\FFF"
Dim ts3 As TimeSpan

For Each input As String In inputs
    If TimeSpan.TryParseExact(input, fmt, Nothing, ts3)
        Console.WriteLine("{0} ('{1}') --> {2}", input, fmt, ts3)
    Else
        Console.WriteLine("Cannot parse {0} with '{1}'.",
                          input, fmt)
    End If
Next
' The example displays the following output:
'     Formatting:
'     00:00:03.6974000 ('FFF') --> 697
'     00:00:03.8009000 ('ss\FFF') --> 03.8
'
'     Parsing:
'     0:0:03. ('h\:m\:ss\FFF') --> 00:00:03
'     0:0:03.12 ('h\:m\:ss\FFF') --> 00:00:03.1200000
'     Cannot parse 0:0:03.1279 with 'h\:m\:ss\FFF'.

```

[Back to table](#)

The "FFFF" custom format specifier

The "FFFF" custom format specifier (with four "F" characters) outputs the ten-thousandths of a second in a time interval. In a formatting operation, any remaining fractional digits are truncated. If there are any trailing fractional zeros, they aren't included in the result string. In a parsing operation that calls the [TimeSpan.ParseExact](#) or [TimeSpan.TryParseExact](#) method, the presence of the tenths, hundredths, thousandths, and ten-thousandths of a second digit is optional.

The following example uses the "FFFF" custom format specifier to display the ten-thousandths of a second in a [TimeSpan](#) value. It also uses the "FFFF" custom format specifier in a parsing operation.

```

Console.WriteLine("Formatting:");
TimeSpan ts1 = TimeSpan.Parse("0:0:3.69749");
Console.WriteLine("{0} ('FFFF') --> {0:FFFF}", ts1);

TimeSpan ts2 = TimeSpan.Parse("0:0:3.80009");
Console.WriteLine("{0} ('ss\\.FFFF') --> {0:ss\\.FFFF}", ts2);
Console.WriteLine();

Console.WriteLine("Parsing:");
string[] inputs = { "0:0:03.", "0:0:03.12", "0:0:03.12795" };
string fmt = @"h\:m\:ss\.FFFF";
TimeSpan ts3;

foreach (string input in inputs) {
    if (TimeSpan.TryParseExact(input, fmt, null, out ts3))
        Console.WriteLine("{0} ('{1}') --> {2}", input, fmt, ts3);
    else
        Console.WriteLine("Cannot parse {0} with '{1}'.",
                         input, fmt);
}
// The example displays the following output:
//      Formatting:
//      00:00:03.6974900 ('FFFF') --> 6974
//      00:00:03.8000900 ('ss\\.FFFF') --> 03.8
//
//      Parsing:
//      0:0:03. ('h\:m\:ss\.FFFF') --> 00:00:03
//      0:0:03.12 ('h\:m\:ss\.FFFF') --> 00:00:03.1200000
//      Cannot parse 0:0:03.12795 with 'h\:m\:ss\.FFFF'.

```

```

Console.WriteLine("Formatting:")
Dim ts1 As TimeSpan = TimeSpan.Parse("0:0:3.69749")
Console.WriteLine("{0} ('FFFF') --> {0:FFFF}", ts1)

Dim ts2 As TimeSpan = TimeSpan.Parse("0:0:3.80009")
Console.WriteLine("{0} ('ss\\.FFFF') --> {0:ss\\.FFFF}", ts2)
Console.WriteLine()

Console.WriteLine("Parsing:")
Dim inputs() As String = {"0:0:03.", "0:0:03.12", "0:0:03.12795"}
Dim fmt As String = "h\:m\:ss\.FFFF"
Dim ts3 As TimeSpan

For Each input As String In inputs
    If TimeSpan.TryParseExact(input, fmt, Nothing, ts3)
        Console.WriteLine("{0} ('{1}') --> {2}", input, fmt, ts3)
    Else
        Console.WriteLine("Cannot parse {0} with '{1}'.",
                         input, fmt)
    End If
Next
' The example displays the following output:
'      Formatting:
'      00:00:03.6974900 ('FFFF') --> 6974
'      00:00:03.8000900 ('ss\\.FFFF') --> 03.8
'
'      Parsing:
'      0:0:03. ('h\:m\:ss\.FFFF') --> 00:00:03
'      0:0:03.12 ('h\:m\:ss\.FFFF') --> 00:00:03.1200000
'      Cannot parse 0:0:03.12795 with 'h\:m\:ss\.FFFF'.

```

[Back to table](#)

The "FFFFF" custom format specifier

The "FFFFF" custom format specifier (with five "F" characters) outputs the hundred-thousandths of a second in a time interval. In a formatting operation, any remaining fractional digits are truncated. If there are any trailing fractional zeros, they aren't included in the result string. In a parsing operation that calls the [TimeSpan.ParseExact](#) or [TimeSpan.TryParseExact](#) method, the presence of the tenths, hundredths, thousandths, ten-thousandths, and hundred-thousandths of a second digit is optional.

The following example uses the "FFFFF" custom format specifier to display the hundred-thousandths of a second in a [TimeSpan](#) value. It also uses the "FFFFF" custom format specifier in a parsing operation.

```
Console.WriteLine("Formatting:");
TimeSpan ts1 = TimeSpan.Parse("0:0:3.697497");
Console.WriteLine("{0} ('FFFFF') --> {0:FFFFF}", ts1);

TimeSpan ts2 = TimeSpan.Parse("0:0:3.800009");
Console.WriteLine("{0} ('ss\\.FFFFF') --> {0:ss\\.FFFFF}", ts2);
Console.WriteLine();

Console.WriteLine("Parsing:");
string[] inputs = { "0:0:03.", "0:0:03.12", "0:0:03.127956" };
string fmt = @"h\:m\:ss\.FFFFF";
TimeSpan ts3;

foreach (string input in inputs) {
    if (TimeSpan.TryParseExact(input, fmt, null, out ts3))
        Console.WriteLine("{0} ('{1}') --> {2}", input, fmt, ts3);
    else
        Console.WriteLine("Cannot parse {0} with '{1}'.",
                         input, fmt);
}
// The example displays the following output:
//      Formatting:
//      00:00:03.6974970 ('FFFFF') --> 69749
//      00:00:03.8000090 ('ss\\.FFFFF') --> 03.8
//
//      Parsing:
//      0:0:03. ('h\:m\:ss\\.FFFFF') --> 00:00:03
//      0:0:03.12 ('h\:m\:ss\\.FFFFF') --> 00:00:03.1200000
//      Cannot parse 0:0:03.127956 with 'h\:m\:ss\\.FFFF'.
```

```

Console.WriteLine("Formatting:")
Dim ts1 As TimeSpan = TimeSpan.Parse("0:0:3.697497")
Console.WriteLine("{0} ('FFFFF') --> {0:FFFFF}", ts1)

Dim ts2 As TimeSpan = TimeSpan.Parse("0:0:3.800009")
Console.WriteLine("{0} ('ss\FFFFF') --> {0:ss\.FFFFF}", ts2)
Console.WriteLine()

Console.WriteLine("Parsing:")
Dim inputs() As String = {"0:0:03.", "0:0:03.12", "0:0:03.127956"}
Dim fmt As String = "h\:m\:ss\.FFFFF"
Dim ts3 As TimeSpan

For Each input As String In inputs
    If TimeSpan.TryParseExact(input, fmt, Nothing, ts3)
        Console.WriteLine("{0} ('{1}') --> {2}", input, fmt, ts3)
    Else
        Console.WriteLine("Cannot parse {0} with '{1}'.",
                           input, fmt)
    End If
Next
' The example displays the following output:
'     Formatting:
'     00:00:03.6974970 ('FFFFF') --> 69749
'     00:00:03.8000090 ('ss\FFFFF') --> 03.8
'
'     Parsing:
'     0:0:03. ('h\:m\:ss\.FFFFF') --> 00:00:03
'     0:0:03.12 ('h\:m\:ss\.FFFFF') --> 00:00:03.1200000
'     Cannot parse 0:0:03.127956 with 'h\:m\:ss\.FFFF'.

```

[Back to table](#)

The "FFFFFF" custom format specifier

The "FFFFFF" custom format specifier (with six "F" characters) outputs the millionths of a second in a time interval. In a formatting operation, any remaining fractional digits are truncated. If there are any trailing fractional zeros, they aren't included in the result string. In a parsing operation that calls the [TimeSpan.ParseExact](#) or [TimeSpan.TryParseExact](#) method, the presence of the tenths, hundredths, thousandths, ten-thousandths, hundred-thousandths, and millionths of a second digit is optional.

The following example uses the "FFFFFF" custom format specifier to display the millionths of a second in a [TimeSpan](#) value. It also uses this custom format specifier in a parsing operation.

```

Console.WriteLine("Formatting:");
TimeSpan ts1 = TimeSpan.Parse("0:0:3.6974974");
Console.WriteLine("{0} ('FFFFFF') --> {0:FFFFFF}", ts1);

TimeSpan ts2 = TimeSpan.Parse("0:0:3.8000009");
Console.WriteLine("{0} ('ss\.\FFFFFF') --> {0:ss\.\FFFFFF}", ts2);
Console.WriteLine();

Console.WriteLine("Parsing:");
string[] inputs = { "0:0:03.", "0:0:03.12", "0:0:03.1279569" };
string fmt = @"h\:m\:ss\.\FFFFFF";
TimeSpan ts3;

foreach (string input in inputs) {
    if (TimeSpan.TryParseExact(input, fmt, null, out ts3))
        Console.WriteLine("{0} ('{1}') --> {2}", input, fmt, ts3);
    else
        Console.WriteLine("Cannot parse {0} with '{1}'.",
                         input, fmt);
}
// The example displays the following output:
//      Formatting:
//      00:00:03.6974974 ('FFFFFF') --> 697497
//      00:00:03.8000009 ('ss\.\FFFFFF') --> 03.8
//
//      Parsing:
//      0:0:03. ('h\:m\:ss\.\FFFFFF') --> 00:00:03
//      0:0:03.12 ('h\:m\:ss\.\FFFFFF') --> 00:00:03.1200000
//      Cannot parse 0:0:03.1279569 with 'h\:m\:ss\.\FFFFFF'.

```

```

Console.WriteLine("Formatting:")
Dim ts1 As TimeSpan = TimeSpan.Parse("0:0:3.6974974")
Console.WriteLine("{0} ('FFFFFF') --> {0:FFFFFF}", ts1)

Dim ts2 As TimeSpan = TimeSpan.Parse("0:0:3.8000009")
Console.WriteLine("{0} ('ss\.\FFFFFF') --> {0:ss\.\FFFFFF}", ts2)
Console.WriteLine()

Console.WriteLine("Parsing:")
Dim inputs() As String = {"0:0:03.", "0:0:03.12", "0:0:03.1279569"}
Dim fmt As String = "h\:m\:ss\.\FFFFFF"
Dim ts3 As TimeSpan

For Each input As String In inputs
    If TimeSpan.TryParseExact(input, fmt, Nothing, ts3)
        Console.WriteLine("{0} ('{1}') --> {2}", input, fmt, ts3)
    Else
        Console.WriteLine("Cannot parse {0} with '{1}'.",
                         input, fmt)
    End If
Next
' The example displays the following output:
'      Formatting:
'      00:00:03.6974974 ('FFFFFF') --> 697497
'      00:00:03.8000009 ('ss\.\FFFFFF') --> 03.8
'
'      Parsing:
'      0:0:03. ('h\:m\:ss\.\FFFFFF') --> 00:00:03
'      0:0:03.12 ('h\:m\:ss\.\FFFFFF') --> 00:00:03.1200000
'      Cannot parse 0:0:03.1279569 with 'h\:m\:ss\.\FFFFFF'.

```

[Back to table](#)

The "FFFFFFF" custom format specifier

The "FFFFFFF" custom format specifier (with seven "F" characters) outputs the ten-millionths of a second (or the fractional number of ticks) in a time interval. If there are any trailing fractional zeros, they aren't included in the result string. In a parsing operation that calls the [TimeSpan.ParseExact](#) or [TimeSpan.TryParseExact](#) method, the presence of the seven fractional digits in the input string is optional.

The following example uses the "FFFFFFF" custom format specifier to display the fractional parts of a second in a [TimeSpan](#) value. It also uses this custom format specifier in a parsing operation.

```
Console.WriteLine("Formatting:");
TimeSpan ts1 = TimeSpan.Parse("0:0:3.6974974");
Console.WriteLine("{0} ('FFFFFFF') --> {0:FFFFFFF}", ts1);

TimeSpan ts2 = TimeSpan.Parse("0:0:3.9500000");
Console.WriteLine("{0} ('ss\\.FFFFFFF') --> {0:ss\\.FFFFFFF}", ts2);
Console.WriteLine();

Console.WriteLine("Parsing:");
string[] inputs = { "0:0:03.", "0:0:03.12", "0:0:03.1279569" };
string fmt = @"h\:m\:ss\.FFFFFFF";
TimeSpan ts3;

foreach (string input in inputs) {
    if (TimeSpan.TryParseExact(input, fmt, null, out ts3))
        Console.WriteLine("{0} ('{1}') --> {2}", input, fmt, ts3);
    else
        Console.WriteLine("Cannot parse {0} with '{1}'.",
                         input, fmt);
}
// The example displays the following output:
//   Formatting:
//   00:00:03.6974974 ('FFFFFFF') --> 6974974
//   00:00:03.9500000 ('ss\.FFFFFFF') --> 03.95
//
//   Parsing:
//   0:0:03. ('h\:m\:ss\.FFFFFFF') --> 00:00:03
//   0:0:03.12 ('h\:m\:ss\.FFFFFFF') --> 00:00:03.1200000
//   0:0:03.1279569 ('h\:m\:ss\.FFFFFFF') --> 00:00:03.1279569
```

```

Console.WriteLine("Formatting:")
Dim ts1 As TimeSpan = TimeSpan.Parse("0:0:3.6974974")
Console.WriteLine("{0} ('FFFFFFF') --> {0:FFFFFFF}", ts1)

Dim ts2 As TimeSpan = TimeSpan.Parse("0:0:3.9500000")
Console.WriteLine("{0} ('ss\FFFFFFF') --> {0:ss\.FFFFFFF}", ts2)
Console.WriteLine()

Console.WriteLine("Parsing:")
Dim inputs() As String = {"0:0:03.", "0:0:03.12", "0:0:03.1279569"}
Dim fmt As String = "h\:m\:ss\.\FFFFFFF"
Dim ts3 As TimeSpan

For Each input As String In inputs
    If TimeSpan.TryParseExact(input, fmt, Nothing, ts3)
        Console.WriteLine("{0} ('{1}') --> {2}", input, fmt, ts3)
    Else
        Console.WriteLine("Cannot parse {0} with '{1}'.",
                          input, fmt)
    End If
Next
' The example displays the following output:
'   Formatting:
'   00:00:03.6974974 ('FFFFFFF') --> 6974974
'   00:00:03.9500000 ('ss\.\FFFFFFF') --> 03.95
'
'   Parsing:
'   0:0:03. ('h\:m\:ss\.\FFFFFFF') --> 00:00:03
'   0:0:03.12 ('h\:m\:ss\.\FFFFFFF') --> 00:00:03.1200000
'   0:0:03.1279569 ('h\:m\:ss\.\FFFFFFF') --> 00:00:03.1279569

```

[Back to table](#)

Other characters

Any other unescaped character in a format string, including a white-space character, is interpreted as a custom format specifier. In most cases, the presence of any other unescaped character results in a [FormatException](#).

There are two ways to include a literal character in a format string:

- Enclose it in single quotation marks (the literal string delimiter).
- Precede it with a backslash ("\"), which is interpreted as an escape character. This means that, in C#, the format string must either be @-quoted, or the literal character must be preceded by an additional backslash.

In some cases, you may have to use conditional logic to include an escaped literal in a format string. The following example uses conditional logic to include a sign symbol for negative time intervals.

```

using System;

public class Example
{
    public static void Main()
    {
        TimeSpan result = new DateTime(2010, 01, 01) - DateTime.Now;
        String fmt = (result < TimeSpan.Zero ? "\\" : "") + "dd\\.hh\\:mm";

        Console.WriteLine(result.ToString(fmt));
        Console.WriteLine("Interval: {0:" + fmt + "}", result);
    }
}

// The example displays output like the following:
//      -1291.10:54
//      Interval: -1291.10:54

```

```

Module Example
Public Sub Main()
    Dim result As TimeSpan = New DateTime(2010, 01, 01) - Date.Now
    Dim fmt As String = If(result < TimeSpan.Zero, "\\", "") + "dd\\.hh\\:mm"

    Console.WriteLine(result.ToString(fmt))
    Console.WriteLine("Interval: {0:" + fmt + "}", result)
End Sub
End Module
' The example displays output like the following:
'      -1291.10:54
'      Interval: -1291.10:54

```

.NET doesn't define a grammar for separators in time intervals. This means that the separators between days and hours, hours and minutes, minutes and seconds, and seconds and fractions of a second must all be treated as character literals in a format string.

The following example uses both the escape character and the single quote to define a custom format string that includes the word "minutes" in the output string.

```

TimeSpan interval = new TimeSpan(0, 32, 45);
// Escape literal characters in a format string.
string fmt = @"mm\ss\ \m\i\n\u\t\e\s";
Console.WriteLine(interval.ToString(fmt));
// Delimit literal characters in a format string with the ' symbol.
fmt = "mm':'ss' minutes'";
Console.WriteLine(interval.ToString(fmt));
// The example displays the following output:
//      32:45 minutes
//      32:45 minutes

```

```

Dim interval As New TimeSpan(0, 32, 45)
' Escape literal characters in a format string.
Dim fmt As String = "mm\ss\ \m\i\n\u\t\e\s"
Console.WriteLine(interval.ToString(fmt))
' Delimit literal characters in a format string with the ' symbol.
fmt = "mm':'ss' minutes'"
Console.WriteLine(interval.ToString(fmt))
' The example displays the following output:
'      32:45 minutes
'      32:45 minutes

```

See also

- [Formatting Types](#)
- [Standard TimeSpan Format Strings](#)

Enumeration format strings

9/20/2022 • 3 minutes to read • [Edit Online](#)

You can use the `Enum.ToString` method to create a new string object that represents the numeric, hexadecimal, or string value of an enumeration member. This method takes one of the enumeration formatting strings to specify the value that you want returned.

The following sections list the enumeration formatting strings and the values they return. These format specifiers are not case-sensitive.

G or g

Displays the enumeration entry as a string value, if possible, and otherwise displays the integer value of the current instance. If the enumeration is defined with the `Flags` attribute set, the string values of each valid entry are concatenated together, separated by commas. If the `Flags` attribute is not set, an invalid value is displayed as a numeric entry. The following example illustrates the G format specifier.

```
Console.WriteLine(ConsoleColor.Red.ToString("G"));           // Displays Red
FileAttributes attributes = FileAttributes.Hidden |
                           FileAttributesArchive;
Console.WriteLine(attributes.ToString("G"));    // Displays Hidden, Archive
```

```
Console.WriteLine(ConsoleColor.Red.ToString("G"))           ' Displays Red
Dim attributes As FileAttributes = FileAttributes.Hidden Or _
                           FileAttributesArchive
Console.WriteLine(attributes.ToString("G"))      ' Displays Hidden, Archive
```

F or f

Displays the enumeration entry as a string value, if possible. If the value can be completely displayed as a summation of the entries in the enumeration (even if the `Flags` attribute is not present), the string values of each valid entry are concatenated together, separated by commas. If the value cannot be completely determined by the enumeration entries, then the value is formatted as the integer value. The following example illustrates the F format specifier.

```
Console.WriteLine(ConsoleColor.Blue.ToString("F"));        // Displays Blue
FileAttributes attributes = FileAttributes.Hidden |
                           FileAttributesArchive;
Console.WriteLine(attributes.ToString("F"));    // Displays Hidden, Archive
```

```
Console.WriteLine(ConsoleColor.Blue.ToString("F"))           ' Displays Blue
Dim attributes As FileAttributes = FileAttributes.Hidden Or _
                           FileAttributesArchive
Console.WriteLine(attributes.ToString("F"))      ' Displays Hidden, Archive
```

D or d

Displays the enumeration entry as an integer value in the shortest representation possible. The following example illustrates the D format specifier.

```
Console.WriteLine(ConsoleColor.Cyan.ToString("D"));           // Displays 11
FileAttributes attributes = FileAttributes.Hidden |
                           FileAttributes.Archive;
Console.WriteLine(attributes.ToString("D"));                  // Displays 34
```

```
Console.WriteLine(ConsoleColor.Cyan.ToString("D"))           ' Displays 11
Dim attributes As FileAttributes = FileAttributes.Hidden Or _
                           FileAttributes.Archive
Console.WriteLine(attributes.ToString("D"))                  ' Displays 34
```

X or x

Displays the enumeration entry as a hexadecimal value. The value is represented with leading zeros as necessary, to ensure that the result string has two characters for each byte in the enumeration type's [underlying numeric type](#). The following example illustrates the X format specifier. In the example, the underlying type of both `ConsoleColor` and `FileAttributes` is `Int32`, or a 32-bit (or 4-byte) integer, which produces an 8-character result string.

```
Console.WriteLine(ConsoleColor.Cyan.ToString("X"));    // Displays 0000000B
FileAttributes attributes = FileAttributes.Hidden |
                           FileAttributes.Archive;
Console.WriteLine(attributes.ToString("X"));            // Displays 00000022
```

```
Console.WriteLine(ConsoleColor.Cyan.ToString("X"))       ' Displays 0000000B
Dim attributes As FileAttributes = FileAttributes.Hidden Or _
                           FileAttributes.Archive
Console.WriteLine(attributes.ToString("X"))             ' Displays 00000022
```

Example

The following example defines an enumeration called `Colors` that consists of three entries: `Red`, `Blue`, and `Green`.

```
public enum Color {Red = 1, Blue = 2, Green = 3}
```

```
Public Enum Color
    Red = 1
    Blue = 2
    Green = 3
End Enum
```

After the enumeration is defined, an instance can be declared in the following manner.

```
Color myColor = Color.Green;
```

```
Dim myColor As Color = Color.Green
```

The `color.ToString(System.String)` method can then be used to display the enumeration value in different ways, depending on the format specifier passed to it.

```
Console.WriteLine("The value of myColor is {0}.",
                  myColor.ToString("G"));
Console.WriteLine("The value of myColor is {0}.",
                  myColor.ToString("F"));
Console.WriteLine("The value of myColor is {0}.",
                  myColor.ToString("D"));
Console.WriteLine("The value of myColor is 0x{0}.",
                  myColor.ToString("X"));

// The example displays the following output to the console:
//      The value of myColor is Green.
//      The value of myColor is Green.
//      The value of myColor is 3.
//      The value of myColor is 0x00000003.
```

```
Console.WriteLine("The value of myColor is {0}.",
                  myColor.ToString("G"))
Console.WriteLine("The value of myColor is {0}.",
                  myColor.ToString("F"))
Console.WriteLine("The value of myColor is {0}.",
                  myColor.ToString("D"))
Console.WriteLine("The value of myColor is 0x{0}.",
                  myColor.ToString("X"))

' The example displays the following output to the console:
'      The value of myColor is Green.
'      The value of myColor is Green.
'      The value of myColor is 3.
'      The value of myColor is 0x00000003.
```

See also

- [Formatting Types](#)

Composite formatting

9/20/2022 • 13 minutes to read • [Edit Online](#)

The .NET composite formatting feature takes a list of objects and a composite format string as input. A composite format string consists of fixed text intermixed with indexed placeholders, called format items. These format items correspond to the objects in the list. The formatting operation yields a result string that consists of the original fixed text intermixed with the string representation of the objects in the list.

IMPORTANT

Instead of using composite format strings, you can use *interpolated strings* if the language and its version that you're using support them. An interpolated string contains *interpolated expressions*. Each interpolated expression is resolved with the expression's value and included in the result string when the string is assigned. For more information, see [String interpolation \(C# Reference\)](#) and [Interpolated strings \(Visual Basic Reference\)](#).

The composite formatting feature is supported by the following methods:

- [String.Format](#), which returns a formatted result string.
- [StringBuilder.AppendFormat](#), which appends a formatted result string to a [StringBuilder](#) object.
- Some overloads of the [Console.WriteLine](#) method, which display a formatted result string to the console.
- Some overloads of the [TextWriter.WriteLine](#) method, which write the formatted result string to a stream or file. The classes derived from [TextWriter](#), such as [StreamWriter](#) and [HtmlTextWriter](#), also share this functionality.
- [Debug.WriteLine\(String, Object\[\]\)](#), which outputs a formatted message to trace listeners.
- The [Trace.TraceError\(String, Object\[\]\)](#), [Trace.TraceInformation\(String, Object\[\]\)](#), and [Trace.TraceWarning\(String, Object\[\]\)](#) methods, which output formatted messages to trace listeners.
- The [TraceSource.TraceInformation\(String, Object\[\]\)](#) method, which writes an informational method to trace listeners.

Composite format string

A composite format string and object list are used as arguments of methods that support the composite formatting feature. A composite format string consists of zero or more runs of fixed text intermixed with one or more format items. The fixed text is any string that you choose, and each format item corresponds to an object or boxed structure in the list. The composite formatting feature returns a new result string where each format item is replaced by the string representation of the corresponding object in the list.

Consider the following [Format](#) code fragment:

```
string name = "Fred";
String.Format("Name = {0}, hours = {1:hh}", name, DateTime.Now);
```

```
Dim name As String = "Fred"
String.Format("Name = {0}, hours = {1:hh}", name, DateTime.Now)
```

The fixed text is `Name =` and `, hours =`. The format items are `{0}`, whose index of 0 corresponds to the object `name`, and `{1:hh}`, whose index of 1 corresponds to the object `DateTime.Now`.

Format item syntax

Each format item takes the following form and consists of the following components:

```
{index[,alignment][:formatString]}
```

The matching braces ({ } and) are required.

Index component

The mandatory *index* component, also called a parameter specifier, is a number starting from 0 that identifies a corresponding item in the list of objects. That is, the format item whose parameter specifier is 0 formats the first object in the list. The format item whose parameter specifier is 1 formats the second object in the list, and so on. The following example includes four parameter specifiers, numbered zero through three, to represent prime numbers less than 10:

```
string primes;
primes = String.Format("Prime numbers less than 10: {0}, {1}, {2}, {3}",
    2, 3, 5, 7);
Console.WriteLine(primes);
// The example displays the following output:
//      Prime numbers less than 10: 2, 3, 5, 7
```

```
Dim primes As String
primes = String.Format("Prime numbers less than 10: {0}, {1}, {2}, {3}",
    2, 3, 5, 7)
Console.WriteLine(primes)
' The example displays the following output:
'      Prime numbers less than 10: 2, 3, 5, 7
```

Multiple format items can refer to the same element in the list of objects by specifying the same parameter specifier. For example, you can format the same numeric value in hexadecimal, scientific, and number format by specifying a composite format string such as "0x{0:X} {0:E} {0:N}", as the following example shows:

```
string multiple = String.Format("0x{0:X} {0:E} {0:N}",
    Int64.MaxValue);
Console.WriteLine(multiple);
// The example displays the following output:
//      0x7FFFFFFFFFFFFF 9.223372E+018 9,223,372,036,854,775,807.00
```

```
Dim multiple As String = String.Format("0x{0:X} {0:E} {0:N}",
    Int64.MaxValue)
Console.WriteLine(multiple)
' The example displays the following output:
'      0x7FFFFFFFFFFFFF 9.223372E+018 9,223,372,036,854,775,807.00
```

Each format item can refer to any object in the list. For example, if there are three objects, you can format the second, first, and third object by specifying a composite format string such as {1} {0} {2}. An object that isn't referenced by a format item is ignored. A [FormatException](#) is thrown at run time if a parameter specifier designates an item outside the bounds of the list of objects.

Alignment component

The optional *alignment* component is a signed integer indicating the preferred formatted field width. If the value of *alignment* is less than the length of the formatted string, *alignment* is ignored, and the length of the formatted string is used as the field width. The formatted data in the field is right-aligned if *alignment* is positive and left-aligned if *alignment* is negative. If padding is necessary, white space is used. The comma is required if

alignment is specified.

The following example defines two arrays, one containing the names of employees and the other containing the hours they worked over two weeks. The composite format string left-aligns the names in a 20-character field and right-aligns their hours in a 5-character field. The "N1" standard format string formats the hours with one fractional digit.

```
using System;

public class Example
{
    public static void Main()
    {
        string[] names = { "Adam", "Bridgette", "Carla", "Daniel",
                           "Ebenezer", "Francine", "George" };
        decimal[] hours = { 40, 6.667m, 40.39m, 82, 40.333m, 80,
                            16.75m };

        Console.WriteLine("{0,-20} {1,5}\n", "Name", "Hours");
        for (int ctr = 0; ctr < names.Length; ctr++)
            Console.WriteLine("{0,-20} {1,5:N1}", names[ctr], hours[ctr]);
    }
}

// The example displays the following output:
//      Name          Hours
//
//      Adam          40.0
//      Bridgette     6.7
//      Carla         40.4
//      Daniel         82.0
//      Ebenezer       40.3
//      Francine       80.0
//      George         16.8
```

```
Module Example
    Public Sub Main()
        Dim names() As String = {"Adam", "Bridgette", "Carla", "Daniel",
                               "Ebenezer", "Francine", "George"}
        Dim hours() As Decimal = {40, 6.667d, 40.39d, 82, 40.333d, 80,
                                 16.75d}

        Console.WriteLine("{0,-20} {1,5}", "Name", "Hours")
        Console.WriteLine()
        For ctr As Integer = 0 To names.Length - 1
            Console.WriteLine("{0,-20} {1,5:N1}", names(ctr), hours(ctr))
        Next
    End Sub
End Module

' The example displays the following output:
'      Name          Hours
'
'      Adam          40.0
'      Bridgette     6.7
'      Carla         40.4
'      Daniel         82.0
'      Ebenezer       40.3
'      Francine       80.0
'      George         16.8
```

Format string component

The optional *formatString* component is a format string that's appropriate for the type of object being formatted. You can specify:

- A standard or custom numeric format string if the corresponding object is a numeric value.
- A standard or custom date and time format string if the corresponding object is a [DateTime](#) object.
- An [enumeration format string](#) if the corresponding object is an enumeration value.

If *formatString* isn't specified, the general ("G") format specifier for a numeric, date and time, or enumeration type is used. The colon is required if *formatString* is specified.

The following table lists types or categories of types in the .NET class library that support a predefined set of format strings, and provides links to the articles that list the supported format strings. String formatting is an extensible mechanism that makes it possible to define new format strings for all existing types and to define a set of format strings supported by an application-defined type.

For more information, see the [IFormattable](#) and [ICustomFormatter](#) interface articles.

TYPE OR TYPE CATEGORY	SEE
Date and time types (DateTime , DateTimeOffset)	Standard Date and Time Format Strings Custom Date and Time Format Strings
Enumeration types (all types derived from System.Enum)	Enumeration Format Strings
Numeric types (BigInteger , Byte , Decimal , Double , Int16 , Int32 , Int64 , SByte , Single , UInt16 , UInt32 , UInt64)	Standard Numeric Format Strings Custom Numeric Format Strings
Guid	Guid.ToString(String)
TimeSpan	Standard TimeSpan Format Strings Custom TimeSpan Format Strings

Escaping braces

Opening and closing braces are interpreted as starting and ending a format item. To display a literal opening brace or closing brace, you must use an escape sequence. Specify two opening braces ({{}) in the fixed text to display one opening brace ({), or two closing braces (}}) to display one closing brace (}). Braces in a format item are interpreted sequentially in the order they're encountered. Interpreting nested braces isn't supported.

The way escaped braces are interpreted can lead to unexpected results. For example, consider the format item {{{{0:D}}}}, which is intended to display an opening brace, a numeric value formatted as a decimal number, and a closing brace. However, the format item is interpreted in the following manner:

1. The first two opening braces ({{}) are escaped and yield one opening brace.
2. The next three characters ({{0:}) are interpreted as the start of a format item.
3. The next character (D) would be interpreted as the Decimal standard numeric format specifier, but the next two escaped braces (}}) yield a single brace. Because the resulting string ({{D}}) isn't a standard numeric format specifier, the resulting string is interpreted as a custom format string that means display the literal string {{D}}.
4. The last brace (}}) is interpreted as the end of the format item.
5. The final result that's displayed is the literal string, {{D}}. The numeric value that was to be formatted isn't displayed.

One way to write your code to avoid misinterpreting escaped braces and format items is to format the braces and format items separately. That is, in the first format operation, display a literal opening brace. In the next operation, display the result of the format item, and in the final operation, display a literal closing brace. The

following example illustrates this approach:

```
int value = 6324;
string output = string.Format("{0}{1:D}{2}",
                               "{", value, "}");
Console.WriteLine(output);
// The example displays the following output:
//      {6324}
```

```
Dim value As Integer = 6324
Dim output As String = String.Format("{0}{1:D}{2}",
                                      "{", value, "}")
Console.WriteLine(output)
' The example displays the following output:
'      {6324}
```

Processing order

If the call to the composite formatting method includes an [IFormatProvider](#) argument whose value isn't `null`, the runtime calls its [IFormatProvider.GetFormat](#) method to request an [ICustomFormatter](#) implementation. If the method can return an [ICustomFormatter](#) implementation, it's cached for the duration of the call of the composite formatting method.

Each value in the parameter list that corresponds to a format item is converted to a string as follows:

1. If the value to be formatted is `null`, an empty string [String.Empty](#) is returned.
2. If an [ICustomFormatter](#) implementation is available, the runtime calls its [Format](#) method. The runtime passes the format item's `formatString` value (or `null` if it's not present) to the method. The runtime also passes the [IFormatProvider](#) implementation to the method. If the call to the [ICustomFormatter.Format](#) method returns `null`, execution proceeds to the next step. Otherwise, the result of the [ICustomFormatter.Format](#) call is returned.
3. If the value implements the [IFormattable](#) interface, the interface's [ToString\(String, IFormatProvider\)](#) method is called. If one is present in the format item, the `formatString` value is passed to the method. Otherwise, `null` is passed. The [IFormatProvider](#) argument is determined as follows:
 - For a numeric value, if a composite formatting method with a non-null [IFormatProvider](#) argument is called, the runtime requests a [NumberFormatInfo](#) object from its [IFormatProvider.GetFormat](#) method. If it's unable to supply one, if the value of the argument is `null`, or if the composite formatting method doesn't have an [IFormatProvider](#) parameter, the [NumberFormatInfo](#) object for the current culture is used.
 - For a date and time value, if a composite formatting method with a non-null [IFormatProvider](#) argument is called, the runtime requests a [DateTimeFormatInfo](#) object from its [IFormatProvider.GetFormat](#) method. In the following situations, the [DateTimeFormatInfo](#) object for the current culture is used instead:
 - The [IFormatProvider.GetFormat](#) method is unable to supply a [DateTimeFormatInfo](#) object.
 - The value of the argument is `null`.
 - The composite formatting method doesn't have an [IFormatProvider](#) parameter.
 - For objects of other types, if a composite formatting method is called with an [IFormatProvider](#) argument, its value is passed directly to the [IFormattable.ToString](#) implementation. Otherwise, `null` is passed to the [IFormattable.ToString](#) implementation.
4. The type's parameterless `ToString` method, which either overrides [Object.ToString\(\)](#) or inherits the

behavior of its base class, is called. In this case, the format string specified by the `formatString` component in the format item, if it's present, is ignored.

Alignment is applied after the preceding steps have been performed.

Code examples

The following example shows one string created using composite formatting and another created using an object's `ToString` method. Both types of formatting produce equivalent results.

```
string FormatString1 = String.Format("{0:dddd MMMM}", DateTime.Now);
string FormatString2 = DateTime.Now.ToString("dddd MMMM");
```

```
Dim FormatString1 As String = String.Format("{0:dddd MMMM}", DateTime.Now)
Dim FormatString2 As String = DateTime.Now.ToString("dddd MMMM")
```

Assuming that the current day is a Thursday in May, the value of both strings in the preceding example is `Thursday May` in the U.S. English culture.

`Console.WriteLine` exposes the same functionality as `String.Format`. The only difference between the two methods is that `String.Format` returns its result as a string, while `Console.WriteLine` writes the result to the output stream associated with the `Console` object. The following example uses the `Console.WriteLine` method to format the value of `MyInt` to a currency value:

```
int MyInt = 100;
Console.WriteLine("{0:C}", MyInt);
// The example displays the following output
// if en-US is the current culture:
//      $100.00
```

```
Dim MyInt As Integer = 100
Console.WriteLine("{0:C}", MyInt)
' The example displays the following output
' if en-US is the current culture:
'      $100.00
```

The following example demonstrates formatting multiple objects, including formatting one object in two different ways:

```
string myName = "Fred";
Console.WriteLine(String.Format("Name = {0}, hours = {1:hh}, minutes = {1:mm}", 
    myName, DateTime.Now));
// Depending on the current time, the example displays output like the following:
//      Name = Fred, hours = 11, minutes = 30
```

```
Dim myName As String = "Fred"
Console.WriteLine(String.Format("Name = {0}, hours = {1:hh}, minutes = {1:mm}", _ 
    myName, DateTime.Now))
' Depending on the current time, the example displays output like the following:
'      Name = Fred, hours = 11, minutes = 30
```

The following example demonstrates the use of alignment in formatting. The arguments that are formatted are placed between vertical bar characters (|) to highlight the resulting alignment.

```

string myFName = "Fred";
string myLName = "Opals";
int myInt = 100;
string FormatFName = String.Format("First Name = |{0,10}|", myFName);
string FormatLName = String.Format("Last Name = |{0,10}|", myLName);
string FormatPrice = String.Format("Price = |{0,10:C}|", myInt);
Console.WriteLine(FormatFName);
Console.WriteLine(FormatLName);
Console.WriteLine(FormatPrice);
Console.WriteLine();

FormatFName = String.Format("First Name = |{0,-10}|", myFName);
FormatLName = String.Format("Last Name = |{0,-10}|", myLName);
FormatPrice = String.Format("Price = |{0,-10:C}|", myInt);
Console.WriteLine(FormatFName);
Console.WriteLine(FormatLName);
Console.WriteLine(FormatPrice);
// The example displays the following output on a system whose current
// culture is en-US:
//      First Name = |      Fred|
//      Last Name = |      Opals|
//      Price = |    $100.00|
//
//      First Name = |Fred      |
//      Last Name = |Opals      |
//      Price = |$100.00      |

```

```

Dim myFName As String = "Fred"
Dim myLName As String = "Opals"

Dim myInt As Integer = 100
Dim FormatFName As String = String.Format("First Name = |{0,10}|", myFName)
Dim FormatLName As String = String.Format("Last Name = |{0,10}|", myLName)
Dim FormatPrice As String = String.Format("Price = |{0,10:C}|", myInt)
Console.WriteLine(FormatFName)
Console.WriteLine(FormatLName)
Console.WriteLine(FormatPrice)
Console.WriteLine()

FormatFName = String.Format("First Name = |{0,-10}|", myFName)
FormatLName = String.Format("Last Name = |{0,-10}|", myLName)
FormatPrice = String.Format("Price = |{0,-10:C}|", myInt)
Console.WriteLine(FormatFName)
Console.WriteLine(FormatLName)
Console.WriteLine(FormatPrice)
' The example displays the following output on a system whose current
' culture is en-US:
'      First Name = |      Fred|
'      Last Name = |      Opals|
'      Price = |    $100.00|
'
'      First Name = |Fred      |
'      Last Name = |Opals      |
'      Price = |$100.00      |

```

See also

- [WriteLine](#)
- [String.Format](#)
- [String interpolation \(C#\)](#)
- [String interpolation \(Visual Basic\)](#)
- [Formatting Types](#)

- [Standard Numeric Format Strings](#)
- [Custom Numeric Format Strings](#)
- [Standard Date and Time Format Strings](#)
- [Custom Date and Time Format Strings](#)
- [Standard TimeSpan Format Strings](#)
- [Custom TimeSpan Format Strings](#)
- [Enumeration Format Strings](#)

How to: Pad a Number with Leading Zeros

9/20/2022 • 6 minutes to read • [Edit Online](#)

You can add leading zeros to an integer by using the "D" [standard numeric format string](#) with a precision specifier. You can add leading zeros to both integer and floating-point numbers by using a [custom numeric format string](#). This article shows how to use both methods to pad a number with leading zeros.

To pad an integer with leading zeros to a specific length

1. Determine the minimum number of digits you want the integer value to display. Include any leading digits in this number.
2. Determine whether you want to display the integer as a decimal or hexadecimal value.
 - To display the integer as a decimal value, call its `ToString(String)` method, and pass the string "Dn" as the value of the `format` parameter, where *n* represents the minimum length of the string.
 - To display the integer as a hexadecimal value, call its `ToString(String)` method and pass the string "Xn" as the value of the `format` parameter, where *n* represents the minimum length of the string.

You can also use the format string in an interpolated string in both [C#](#) and [Visual Basic](#). Alternatively, you can call a method such as `String.Format` or `Console.WriteLine` that uses [composite formatting](#).

The following example formats several integer values with leading zeros so that the total length of the formatted number is at least eight characters.

```
byte byteValue = 254;
short shortValue = 10342;
int intValue = 1023983;
long lngValue = 6985321;
ulong ulngValue = UInt64.MaxValue;

// Display integer values by calling the ToString method.
Console.WriteLine("{0,22} {1,22}", byteValue.ToString("D8"), byteValue.ToString("X8"));
Console.WriteLine("{0,22} {1,22}", shortValue.ToString("D8"), shortValue.ToString("X8"));
Console.WriteLine("{0,22} {1,22}", intValue.ToString("D8"), intValue.ToString("X8"));
Console.WriteLine("{0,22} {1,22}", lngValue.ToString("D8"), lngValue.ToString("X8"));
Console.WriteLine("{0,22} {1,22}", ulngValue.ToString("D8"), ulngValue.ToString("X8"));
Console.WriteLine();

// Display the same integer values by using composite formatting.
Console.WriteLine("{0,22:D8} {0,22:X8}", byteValue);
Console.WriteLine("{0,22:D8} {0,22:X8}", shortValue);
Console.WriteLine("{0,22:D8} {0,22:X8}", intValue);
Console.WriteLine("{0,22:D8} {0,22:X8}", lngValue);
Console.WriteLine("{0,22:D8} {0,22:X8}", ulngValue);
// The example displays the following output:
//          00000254      000000FE
//          00010342      00002866
//          01023983      000F9FEF
//          06985321      006A9669
//          18446744073709551615      FFFFFFFFFFFFFF
//
//          00000254      000000FE
//          00010342      00002866
//          01023983      000F9FEF
//          06985321      006A9669
//          18446744073709551615      FFFFFFFFFFFFFF
//          18446744073709551615      FFFFFFFFFFFFFF
```

```

Dim byteValue As Byte = 254
Dim shortValue As Short = 10342
Dim intValue As Integer = 1023983
Dim lngValue As Long = 6985321
Dim ulngValue As ULong = UInt64.MaxValue

' Display integer values by calling the ToString method.
Console.WriteLine("{0,22} {1,22}", byteValue.ToString("D8"), byteValue.ToString("X8"))
Console.WriteLine("{0,22} {1,22}", shortValue.ToString("D8"), shortValue.ToString("X8"))
Console.WriteLine("{0,22} {1,22}", intValue.ToString("D8"), intValue.ToString("X8"))
Console.WriteLine("{0,22} {1,22}", lngValue.ToString("D8"), lngValue.ToString("X8"))
Console.WriteLine("{0,22} {1,22}", ulngValue.ToString("D8"), ulngValue.ToString("X8"))
Console.WriteLine()

' Display the same integer values by using composite formatting.
Console.WriteLine("{0,22:D8} {0,22:X8}", byteValue)
Console.WriteLine("{0,22:D8} {0,22:X8}", shortValue)
Console.WriteLine("{0,22:D8} {0,22:X8}", intValue)
Console.WriteLine("{0,22:D8} {0,22:X8}", lngValue)
Console.WriteLine("{0,22:D8} {0,22:X8}", ulngValue)
' The example displays the following output:
'          00000254      000000FE
'          00010342      00002866
'          01023983      000F9FEE
'          06985321      006A9669
'          18446744073709551615      FFFFFFFFFFFFFF
'
'          00000254      000000FE
'          00010342      00002866
'          01023983      000F9FEE
'          06985321      006A9669
'          18446744073709551615      FFFFFFFFFFFFFF

```

To pad an integer with a specific number of leading zeros

1. Determine how many leading zeros you want the integer value to display.
2. Determine whether you want to display the integer as a decimal or a hexadecimal value.
 - Formatting it as a decimal value requires the "D" standard format specifier.
 - Formatting it as a hexadecimal value requires the "X" standard format specifier.
3. Determine the length of the unpadded numeric string by calling the integer value's `ToString("D").Length` or `ToString("X").Length` method.
4. Add to the length of the unpadded numeric string the number of leading zeros that you want in the formatted string. The result is the total length of the padded string.
5. Call the integer value's `ToString(String)` method, and pass the string "Dn" for decimal strings and "Xn" for hexadecimal strings, where *n* represents the total length of the padded string. You can also use the "Dn" or "Xn" format string in a method that supports composite formatting.

The following example pads an integer value with five leading zeros:

```
int value = 160934;
int decimalLength = value.ToString("D").Length + 5;
int hexLength = value.ToString("X").Length + 5;
Console.WriteLine(value.ToString("D" + decimalLength.ToString()));
Console.WriteLine(value.ToString("X" + hexLength.ToString()));
// The example displays the following output:
//      00000160934
//      00000274A6
```

```
Dim value As Integer = 160934
Dim decimalLength As Integer = value.ToString("D").Length + 5
Dim hexLength As Integer = value.ToString("X").Length + 5
Console.WriteLine(value.ToString("D" + decimalLength.ToString()))
Console.WriteLine(value.ToString("X" + hexLength.ToString()))
' The example displays the following output:
'      00000160934
'      00000274A6
```

To pad a numeric value with leading zeros to a specific length

1. Determine how many digits to the left of the decimal you want the string representation of the number to have. Include any leading zeros in this total number of digits.
2. Define a custom numeric format string that uses the zero placeholder ("0") to represent the minimum number of zeros.
3. Call the number's `ToString(String)` method and pass it the custom format string. You can also use the custom format string with string interpolation or a method that supports composite formatting.

The following example formats several numeric values with leading zeros. As a result, the total length of the formatted number is at least eight digits to the left of the decimal.

```

string fmt = "00000000.##";
int intValue = 1053240;
decimal decValue = 103932.52m;
float sngValue = 1549230.10873992f;
double dblValue = 9034521202.93217412;

// Display the numbers using the ToString method.
Console.WriteLine(intValue.ToString(fmt));
Console.WriteLine(decValue.ToString(fmt));
Console.WriteLine(sngValue.ToString(fmt));
Console.WriteLine(dblValue.ToString(fmt));
Console.WriteLine();

// Display the numbers using composite formatting.
string formatString = " {0,15:" + fmt + "}";
Console.WriteLine(formatString, intValue);
Console.WriteLine(formatString, decValue);
Console.WriteLine(formatString, sngValue);
Console.WriteLine(formatString, dblValue);
// The example displays the following output:
//      01053240
//      00103932.52
//      01549230
//      9034521202.93
//
//      01053240
//      00103932.52
//      01549230
//      9034521202.93

```

```

Dim fmt As String = "00000000.##"
Dim intValue As Integer = 1053240
Dim decValue As Decimal = 103932.52d
Dim sngValue As Single = 1549230.10873992
Dim dblValue As Double = 9034521202.93217412

' Display the numbers using the ToString method.
Console.WriteLine(intValue.ToString(fmt))
Console.WriteLine(decValue.ToString(fmt))
Console.WriteLine(sngValue.ToString(fmt))
Console.WriteLine(dblValue.ToString(fmt))
Console.WriteLine()

' Display the numbers using composite formatting.
Dim formatString As String = " {0,15:" + fmt + "}"
Console.WriteLine(formatString, intValue)
Console.WriteLine(formatString, decValue)
Console.WriteLine(formatString, sngValue)
Console.WriteLine(formatString, dblValue)
' The example displays the following output:
'      01053240
'      00103932.52
'      01549230
'      9034521202.93
'
'      01053240
'      00103932.52
'      01549230
'      9034521202.93

```

To pad a numeric value with a specific number of leading zeros

1. Determine how many leading zeros you want the numeric value to have.

2. Determine the number of digits to the left of the decimal in the unpadded numeric string:
 - a. Determine whether the string representation of a number includes a decimal point symbol.
 - b. If it does include a decimal point symbol, determine the number of characters to the left of the decimal point. If it doesn't include a decimal point symbol, determine the string's length.
3. Create a custom format string that uses:
 - The zero placeholder ("0") for each of the leading zeros to appear in the string.
 - Either the zero placeholder or the digit placeholder "#" to represent each digit in the default string.
4. Supply the custom format string as a parameter either to the number's `Tostring(String)` method or to a method that supports composite formatting.

The following example pads two `Double` values with five leading zeros:

```
double[] dblValues = { 9034521202.93217412, 9034521202 };
foreach (double dblValue in dblValues)
{
    string decSeparator = System.Globalization.NumberFormatInfo.CurrentInfo.NumberDecimalSeparator;
    string fmt, formatString;

    if (dblValue.ToString().Contains(decSeparator))
    {
        int digits = dblValue.ToString().IndexOf(decSeparator);
        fmt = new String('0', 5) + new String('#', digits) + ".##";
    }
    else
    {
        fmt = new String('0', dblValue.ToString().Length);
    }
    formatString = "{0,20:" + fmt + "}";

    Console.WriteLine(dblValue.ToString(fmt));
    Console.WriteLine(formatString, dblValue);
}
// The example displays the following output:
//      000009034521202.93
//      000009034521202.93
//      9034521202
//      9034521202
```

```
Dim dblValues() As Double = {9034521202.93217412, 9034521202}
For Each dblValue As Double In dblValues
    Dim decSeparator As String = System.Globalization.NumberFormatInfo.CurrentInfo.NumberDecimalSeparator
    Dim fmt, formatString As String

    If dblValue.ToString.Contains(decSeparator) Then
        Dim digits As Integer = dblValue.ToString().IndexOf(decSeparator)
        fmt = New String("0"c, 5) + New String("#"c, digits) + ".##"
    Else
        fmt = New String("0"c, dblValue.ToString.Length)
    End If
    formatString = "{0,20:" + fmt + "}"

    Console.WriteLine(dblValue.ToString(fmt))
    Console.WriteLine(formatString, dblValue)
Next
' The example displays the following output:
'      000009034521202.93
'      000009034521202.93
'      9034521202
'      9034521202
```

See also

- [Custom Numeric Format Strings](#)
- [Standard Numeric Format Strings](#)
- [Composite Formatting](#)

How to: Extract the Day of the Week from a Specific Date

9/20/2022 • 8 minutes to read • [Edit Online](#)

.NET makes it easy to determine the ordinal day of the week for a particular date, and to display the localized weekday name for a particular date. An enumerated value that indicates the day of the week corresponding to a particular date is available from the [DayOfWeek](#) or [DayOfWeek](#) property. In contrast, retrieving the weekday name is a formatting operation that can be performed by calling a formatting method, such as a date and time value's [ToString](#) method or the [String.Format](#) method. This topic shows how to perform these formatting operations.

Extract a number indicating the day of the week

1. If you are working with the string representation of a date, convert it to a [DateTime](#) or a [DateTimeOffset](#) value by using the static [DateTime.Parse](#) or [DateTimeOffset.Parse](#) method.
2. Use the [DateTime.DayOfWeek](#) or [DateTimeOffset.DayOfWeek](#) property to retrieve a [DayOfWeek](#) value that indicates the day of the week.
3. If necessary, cast (in C#) or convert (in Visual Basic) the [DayOfWeek](#) value to an integer.

The following example displays an integer that represents the day of the week of a specific date.

```
using System;

public class Example
{
    public static void Main()
    {
        DateTime dateValue = new DateTime(2008, 6, 11);
        Console.WriteLine((int) dateValue.DayOfWeek);
    }
}
// The example displays the following output:
//      3
```

```
Module Example
    Public Sub Main()
        Dim dateValue As Date = #6/11/2008#
        Console.WriteLine(dateValue.DayOfWeek)
    End Sub
End Module
' The example displays the following output:
'      3
```

Extract the abbreviated weekday name

1. If you are working with the string representation of a date, convert it to a [DateTime](#) or a [DateTimeOffset](#) value by using the static [DateTime.Parse](#) or [DateTimeOffset.Parse](#) method.
2. You can extract the abbreviated weekday name of the current culture or of a specific culture:
 - a. To extract the abbreviated weekday name for the current culture, call the date and time value's

`DateTime.ToString(String)` or `DateTimeOffset.ToString(String)` instance method, and pass the string "ddd" as the `format` parameter. The following example illustrates the call to the `ToString(String)` method.

```
using System;

public class Example
{
    public static void Main()
    {
        DateTime dateValue = new DateTime(2008, 6, 11);
        Console.WriteLine(dateValue.ToString("ddd"));
    }
}
// The example displays the following output:
//      Wed
```

```
Module Example
    Public Sub Main()
        Dim dateValue As Date = #6/11/2008#
        Console.WriteLine(dateValue.ToString("ddd"))
    End Sub
End Module
' The example displays the following output:
'      Wed
```

- b. To extract the abbreviated weekday name for a specific culture, call the date and time value's `DateTime.ToString(String, IFormatProvider)` or `DateTimeOffset.ToString(String, IFormatProvider)` instance method. Pass the string "ddd" as the `format` parameter. Pass either a `CultureInfo` or a `DatetimeFormatInfo` object that represents the culture whose weekday name you want to retrieve as the `provider` parameter. The following code illustrates a call to the `ToString(String, IFormatProvider)` method using a `CultureInfo` object that represents the fr-FR culture.

```
using System;
using System.Globalization;

public class Example
{
    public static void Main()
    {
        DateTime dateValue = new DateTime(2008, 6, 11);
        Console.WriteLine(dateValue.ToString("ddd",
            new CultureInfo("fr-FR")));
    }
}
// The example displays the following output:
//      mer.
```

```
Imports System.Globalization

Module Example
    Public Sub Main()
        Dim dateValue As Date = #6/11/2008#
        Console.WriteLine(dateValue.ToString("ddd",
            New CultureInfo("fr-FR")))
    End Sub
End Module
' The example displays the following output:
'      mer.
```

Extract the full weekday name

1. If you are working with the string representation of a date, convert it to a [DateTime](#) or a [DateTimeOffset](#) value by using the static [DateTime.Parse](#) or [DateTimeOffset.Parse](#) method.
2. You can extract the full weekday name of the current culture or of a specific culture:
 - a. To extract the weekday name for the current culture, call the date and time value's [DateTime.ToString\(String\)](#) or [DateTimeOffset.ToString\(String\)](#) instance method, and pass the string "dddd" as the `format` parameter. The following example illustrates the call to the [ToString\(String\)](#) method.

```
using System;

public class Example
{
    public static void Main()
    {
        DateTime dateValue = new DateTime(2008, 6, 11);
        Console.WriteLine(dateValue.ToString("dddd"));
    }
}
// The example displays the following output:
//      Wednesday
```

```
Module Example
Public Sub Main()
    Dim dateValue As Date = #6/11/2008#
    Console.WriteLine(dateValue.ToString("dddd"))
End Sub
End Module
' The example displays the following output:
'      Wednesday
```

- a. To extract the weekday name for a specific culture, call the date and time value's [DateTime.ToString\(String, IFormatProvider\)](#) or [DateTimeOffset.ToString\(String, IFormatProvider\)](#) instance method. Pass the string "dddd" as the `format` parameter. Pass either a [CultureInfo](#) or a [DateTimeFormatInfo](#) object that represents the culture whose weekday name you want to retrieve as the `provider` parameter. The following code illustrates a call to the [ToString\(String, IFormatProvider\)](#) method using a [CultureInfo](#) object that represents the es-ES culture.

```
using System;
using System.Globalization;

public class Example
{
    public static void Main()
    {
        DateTime dateValue = new DateTime(2008, 6, 11);
        Console.WriteLine(dateValue.ToString("dddd",
            new CultureInfo("es-ES")));
    }
}
// The example displays the following output:
//      miércoles.
```

```

Imports System.Globalization

Module Example
    Public Sub Main()
        Dim dateValue As Date = #6/11/2008#
        Console.WriteLine(dateValue.ToString("ddd", _
            New CultureInfo("es-ES")))
    End Sub
End Module
' The example displays the following output:
' miércoles.

```

Example

The example illustrates calls to the [DateTime.DayOfWeek](#) and [DateTimeOffset.DayOfWeek](#) properties and the [DateTime.ToString](#) and [DateTimeOffset.ToString](#) methods to retrieve the number that represents the day of the week, the abbreviated weekday name, and the full weekday name for a particular date.

```

using System;
using System.Globalization;

public class Example
{
    public static void Main()
    {
        string dateString = "6/11/2007";
        DateTime dateValue;
        DateTimeOffset dateOffsetValue;

        try
        {
            DateTimeFormatInfo dateTimeFormats;
            // Convert date representation to a date value
            dateValue = DateTime.Parse(dateString, CultureInfo.InvariantCulture);
            dateOffsetValue = new DateTimeOffset(dateValue,
                TimeZoneInfo.Local.GetUtcOffset(dateValue));

            // Convert date representation to a number indicating the day of week
            Console.WriteLine((int) dateValue.DayOfWeek);
            Console.WriteLine((int) dateOffsetValue.DayOfWeek);

            // Display abbreviated weekday name using current culture
            Console.WriteLine(dateValue.ToString("ddd"));
            Console.WriteLine(dateOffsetValue.ToString("ddd"));

            // Display full weekday name using current culture
            Console.WriteLine(dateValue.ToString("ddd"));
            Console.WriteLine(dateOffsetValue.ToString("ddd"));

            // Display abbreviated weekday name for de-DE culture
            Console.WriteLine(dateValue.ToString("ddd", new CultureInfo("de-DE")));
            Console.WriteLine(dateOffsetValue.ToString("ddd",
                new CultureInfo("de-DE")));

            // Display abbreviated weekday name with de-DE DateTimeFormatInfo object
            dateTimeFormats = new CultureInfo("de-DE").DateTimeFormat;
            Console.WriteLine(dateValue.ToString("ddd", dateTimeFormats));
            Console.WriteLine(dateOffsetValue.ToString("ddd", dateTimeFormats));

            // Display full weekday name for fr-FR culture
            Console.WriteLine(dateValue.ToString("ddd", new CultureInfo("fr-FR")));
            Console.WriteLine(dateOffsetValue.ToString("ddd",
                new CultureInfo("fr-FR")));
        }
    }
}

```

```
// Display abbreviated weekday name with fr-FR DateTimeFormatInfo object
dateTimeFormats = new CultureInfo("fr-FR").DateTimeFormat;
Console.WriteLine(dateValue.ToString("dddd", dateTimeFormats));
Console.WriteLine(dateOffsetValue.ToString("dddd", dateTimeFormats));
}
catch (FormatException)
{
    Console.WriteLine("Unable to convert {0} to a date.", dateString);
}
}

// The example displays the following output:
//      1
//      1
//      Mon
//      Mon
//      Monday
//      Monday
//      Mo
//      Mo
//      Mo
//      Mo
//      Mo
//      lun.
//      lun.
//      lundi
//      lundi
```

```

Imports System.Globalization

Module Example
    Public Sub Main()
        Dim dateString As String = "6/11/2007"
        Dim dateValue As Date
        Dim dateOffsetValue As DateTimeOffset

        Try
            Dim dateTimeFormats As DateTimeFormatInfo
            ' Convert date representation to a date value
            dateValue = Date.Parse(dateString, CultureInfo.InvariantCulture)
            dateOffsetValue = New DateTimeOffset(dateValue, _
                TimeZoneInfo.Local.GetUtcOffset(dateValue))
            ' Convert date representation to a number indicating the day of week
            Console.WriteLine(dateValue.DayOfWeek)
            Console.WriteLine(dateOffsetValue.DayOfWeek)

            ' Display abbreviated weekday name using current culture
            Console.WriteLine(dateValue.ToString("ddd"))
            Console.WriteLine(dateOffsetValue.ToString("ddd"))

            ' Display full weekday name using current culture
            Console.WriteLine(dateValue.ToString("dddd"))
            Console.WriteLine(dateOffsetValue.ToString("dddd"))

            ' Display abbreviated weekday name for de-DE culture
            Console.WriteLine(dateValue.ToString("ddd", New CultureInfo("de-DE")))
            Console.WriteLine(dateOffsetValue.ToString("ddd", _
                New CultureInfo("de-DE")))

            ' Display abbreviated weekday name with de-DE DateTimeFormatInfo object
            dateTimeFormats = New CultureInfo("de-DE").DateTimeFormat
            Console.WriteLine(dateValue.ToString("ddd", dateTimeFormats))
            Console.WriteLine(dateOffsetValue.ToString("ddd", dateTimeFormats))

            ' Display full weekday name for fr-FR culture
            Console.WriteLine(dateValue.ToString("ddd", New CultureInfo("fr-FR")))
            Console.WriteLine(dateOffsetValue.ToString("ddd", _
                New CultureInfo("fr-FR")))

            ' Display abbreviated weekday name with fr-FR DateTimeFormatInfo object
            dateTimeFormats = New CultureInfo("fr-FR").DateTimeFormat
            Console.WriteLine(dateValue.ToString("dddd", dateTimeFormats))
            Console.WriteLine(dateOffsetValue.ToString("dddd", dateTimeFormats))
        Catch e As FormatException
            Console.WriteLine("Unable to convert {0} to a date.", dateString)
        End Try
    End Sub
End Module
' The example displays the following output to the console:
' 1
' 1
' Mon
' Mon
' Monday
' Monday
' Mo
' Mo
' Mo
' Mo
' lun.
' lun.
' lundi
' lundi

```

Individual languages may provide functionality that duplicates or supplements the functionality provided by .NET. For example, Visual Basic includes two such functions:

- `Weekday`, which returns a number that indicates the day of the week of a particular date. It considers the ordinal value of the first day of the week to be one, whereas the `DateTime.DayOfWeek` property considers it to be zero.
- `WeekdayName`, which returns the name of the week in the current culture that corresponds to a particular weekday number.

The following example illustrates the use of the Visual Basic `Weekday` and `WeekdayName` functions.

```
Imports System.Globalization
Imports System.Threading

Module Example
    Public Sub Main()
        Dim dateValue As Date = #6/11/2008#
        
        ' Get weekday number using Visual Basic Weekday function
        Console.WriteLine(Weekday(dateValue))           ' Displays 4
        ' Compare with .NET DateTime.DayOfWeek property
        Console.WriteLine(dateValue.DayOfWeek)          ' Displays 3
        
        ' Get weekday name using Weekday and WeekdayName functions
        Console.WriteLine(WeekdayName(Weekday(dateValue)))   ' Displays Wednesday
        
        ' Change culture to de-DE
        Dim originalCulture As CultureInfo = Thread.CurrentThread.CurrentCulture
        Thread.CurrentThread.CurrentCulture = New CultureInfo("de-DE")
        ' Get weekday name using Weekday and WeekdayName functions
        Console.WriteLine(WeekdayName(Weekday(dateValue)))   ' Displays Donnerstag
        
        ' Restore original culture
        Thread.CurrentThread.CurrentCulture = originalCulture
    End Sub
End Module
```

You can also use the value returned by the `DateTime.DayOfWeek` property to retrieve the weekday name of a particular date. This requires only a call to the `ToString` method on the `DayOfWeek` value returned by the property. However, this technique does not produce a localized weekday name for the current culture, as the following example illustrates.

```

using System;
using System.Globalization;
using System.Threading;

public class Example
{
    public static void Main()
    {
        // Change current culture to fr-FR
        CultureInfo originalCulture = Thread.CurrentThread.CurrentCulture;
        Thread.CurrentThread.CurrentCulture = new CultureInfo("fr-FR");

        DateTime dateValue = new DateTime(2008, 6, 11);
        // Display the DayOfWeek string representation
        Console.WriteLine(dateValue.DayOfWeek.ToString());
        // Restore original current culture
        Thread.CurrentThread.CurrentCulture = originalCulture;
    }
}

// The example displays the following output:
//      Wednesday

```

```

Imports System.Globalization
Imports System.Threading

Module Example
    Public Sub Main()
        ' Change current culture to fr-FR
        Dim originalCulture As CultureInfo = Thread.CurrentThread.CurrentCulture
        Thread.CurrentThread.CurrentCulture = New CultureInfo("fr-FR")

        Dim dateValue As Date = #6/11/2008#
        ' Display the DayOfWeek string representation
        Console.WriteLine(dateValue.DayOfWeek.ToString())
        ' Restore original current culture
        Thread.CurrentThread.CurrentCulture = originalCulture
    End Sub
End Module

' The example displays the following output:
'      Wednesday

```

See also

- [Standard Date and Time Format Strings](#)
- [Custom Date and Time Format Strings](#)

How to: Define and Use Custom Numeric Format Providers

9/20/2022 • 7 minutes to read • [Edit Online](#)

.NET gives you extensive control over the string representation of numeric values. It supports the following features for customizing the format of numeric values:

- Standard numeric format strings, which provide a predefined set of formats for converting numbers to their string representation. You can use them with any numeric formatting method, such as `Decimal.ToString(String)`, that has a `format` parameter. For details, see [Standard Numeric Format Strings](#).
- Custom numeric format strings, which provide a set of symbols that can be combined to define custom numeric format specifiers. They can also be used with any numeric formatting method, such as `Decimal.ToString(String)`, that has a `format` parameter. For details, see [Custom Numeric Format Strings](#).
- Custom `CultureInfo` or `NumberFormatInfo` objects, which define the symbols and format patterns used in displaying the string representations of numeric values. You can use them with any numeric formatting method, such as `ToString`, that has a `provider` parameter. Typically, the `provider` parameter is used to specify culture-specific formatting.

In some cases (such as when an application must display a formatted account number, an identification number, or a postal code) these three techniques are inappropriate. .NET also enables you to define a formatting object that is neither a `CultureInfo` nor a `NumberFormatInfo` object to determine how a numeric value is formatted. This topic provides the step-by-step instructions for implementing such an object, and provides an example that formats telephone numbers.

Define a custom format provider

1. Define a class that implements the `IFormatProvider` and `ICustomFormatter` interfaces.
2. Implement the `IFormatProvider.GetFormat` method. `GetFormat` is a callback method that the formatting method (such as the `String.Format(IFormatProvider, String, Object[])` method) invokes to retrieve the object that is actually responsible for performing custom formatting. A typical implementation of `GetFormat` does the following:
 - a. Determines whether the `Type` object passed as a method parameter represents an `ICustomFormatter` interface.
 - b. If the parameter does represent the `ICustomFormatter` interface, `GetFormat` returns an object that implements the `ICustomFormatter` interface that is responsible for providing custom formatting. Typically, the custom formatting object returns itself.
 - c. If the parameter does not represent the `ICustomFormatter` interface, `GetFormat` returns `null`.
3. Implement the `Format` method. This method is called by the `String.Format(IFormatProvider, String, Object[])` method and is responsible for returning the string representation of a number. Implementing the method typically involves the following:
 - a. Optionally, make sure that the method is legitimately intended to provide formatting services by examining the `provider` parameter. For formatting objects that implement both `IFormatProvider` and `ICustomFormatter`, this involves testing the `provider` parameter for equality with the current formatting object.

- b. Determine whether the formatting object should support custom format specifiers. (For example, an "N" format specifier might indicate that a U.S. telephone number should be output in NANP format, and an "I" might indicate output in ITU-T Recommendation E.123 format.) If format specifiers are used, the method should handle the specific format specifier. It is passed to the method in the `format` parameter. If no specifier is present, the value of the `format` parameter is `String.Empty`.
- c. Retrieve the numeric value passed to the method as the `arg` parameter. Perform whatever manipulations are required to convert it to its string representation.
- d. Return the string representation of the `arg` parameter.

Use a custom numeric formatting object

1. Create a new instance of the custom formatting class.
2. Call the `String.Format(IFormatProvider, String, Object[])` formatting method, passing it the custom formatting object, the formatting specifier (or `String.Empty`, if one is not used), and the numeric value to be formatted.

Example

The following example defines a custom numeric format provider named `TelephoneFormatter` that converts a number that represents a U.S. telephone number to its NANP or E.123 format. The method handles two format specifiers, "N" (which outputs the NANP format) and "I" (which outputs the international E.123 format).

```
using System;
using System.Globalization;

public class TelephoneFormatter : IFormatProvider, ICustomFormatter
{
    public object GetFormat(Type formatType)
    {
        if (formatType == typeof(ICustomFormatter))
            return this;
        else
            return null;
    }

    public string Format(string format, object arg, IFormatProvider formatProvider)
    {
        // Check whether this is an appropriate callback
        if (!this.Equals(formatProvider))
            return null;

        // Set default format specifier
        if (string.IsNullOrEmpty(format))
            format = "N";

        string numericString = arg.ToString();

        if (format == "N")
        {
            if (numericString.Length <= 4)
                return numericString;
            else if (numericString.Length == 7)
                return numericString.Substring(0, 3) + "-" + numericString.Substring(3, 4);
            else if (numericString.Length == 10)
                return "(" + numericString.Substring(0, 3) + " ) " +
                    numericString.Substring(3, 3) + "-" + numericString.Substring(6);
        }
        else
            throw new FormatException();
    }
}
```

```
        string.Format("'{0}' cannot be used to format {1}.",
                      format, arg.ToString())));
    }
    else if (format == "I")
    {
        if (numericString.Length < 10)
            throw new FormatException(string.Format("{0} does not have 10 digits.", arg.ToString()));
        else
            numericString = "+1 " + numericString.Substring(0, 3) + " " + numericString.Substring(3, 3) + "
" + numericString.Substring(6);
    }
    else
    {
        throw new FormatException(string.Format("The {0} format specifier is invalid.", format));
    }
    return numericString;
}
}

public class TestTelephoneFormatter
{
    public static void Main()
    {
        Console.WriteLine(String.Format(new TelephoneFormatter(), "{0}", 0));
        Console.WriteLine(String.Format(new TelephoneFormatter(), "{0}", 911));
        Console.WriteLine(String.Format(new TelephoneFormatter(), "{0}", 8490216));
        Console.WriteLine(String.Format(new TelephoneFormatter(), "{0}", 4257884748));

        Console.WriteLine(String.Format(new TelephoneFormatter(), "{0:N}", 0));
        Console.WriteLine(String.Format(new TelephoneFormatter(), "{0:N}", 911));
        Console.WriteLine(String.Format(new TelephoneFormatter(), "{0:N}", 8490216));
        Console.WriteLine(String.Format(new TelephoneFormatter(), "{0:N}", 4257884748));

        Console.WriteLine(String.Format(new TelephoneFormatter(), "{0:I}", 4257884748));
    }
}
```

```

Public Class TelephoneFormatter : Implements IFormatProvider, ICustomFormatter
    Public Function GetFormat(formatType As Type) As Object _
        Implements IFormatProvider.GetFormat
        If formatType Is GetType(ICustomFormatter) Then
            Return Me
        Else
            Return Nothing
        End If
    End Function

    Public Function Format(fmt As String, arg As Object, _
        formatProvider As IFormatProvider) As String _
        Implements ICustomFormatter.Format
        ' Check whether this is an appropriate callback
        If Not Me.Equals(formatProvider) Then Return Nothing

        ' Set default format specifier
        If String.IsNullOrEmpty(fmt) Then fmt = "N"

        Dim numericString As String = arg.ToString

        If fmt = "N" Then
            Select Case numericString.Length
                Case <= 4
                    Return numericString
                Case 7
                    Return Left(numericString, 3) & "-" & Mid(numericString, 4)
                Case 10
                    Return "(" & Left(numericString, 3) & ")" & _
                        Mid(numericString, 4, 3) & "-" & Mid(numericString, 7)
                Case Else
                    Throw New FormatException( _
                        String.Format("'{0}' cannot be used to format {1}.", _
                        fmt, arg.ToString()))
            End Select
        ElseIf fmt = "I" Then
            If numericString.Length < 10 Then
                Throw New FormatException(String.Format("{0} does not have 10 digits.", arg.ToString()))
            Else
                numericString = "+1 " & Left(numericString, 3) & " " & Mid(numericString, 4, 3) & " " &
                Mid(numericString, 7)
            End If
        Else
            Throw New FormatException(String.Format("The {0} format specifier is invalid.", fmt))
        End If
        Return numericString
    End Function
End Class

Public Module TestTelephoneFormatter
    Public Sub Main
        Console.WriteLine(String.Format(New TelephoneFormatter, "{0}", 0))
        Console.WriteLine(String.Format(New TelephoneFormatter, "{0}", 911))
        Console.WriteLine(String.Format(New TelephoneFormatter, "{0}", 8490216))
        Console.WriteLine(String.Format(New TelephoneFormatter, "{0}", 4257884748))

        Console.WriteLine(String.Format(New TelephoneFormatter, "{0:N}", 0))
        Console.WriteLine(String.Format(New TelephoneFormatter, "{0:N}", 911))
        Console.WriteLine(String.Format(New TelephoneFormatter, "{0:N}", 8490216))
        Console.WriteLine(String.Format(New TelephoneFormatter, "{0:N}", 4257884748))

        Console.WriteLine(String.Format(New TelephoneFormatter, "{0:I}", 4257884748))
    End Sub
End Module

```

The custom numeric format provider can be used only with the [String.Format\(IFormatProvider, String, Object\[\]\)](#)

method. The other overloads of numeric formatting methods (such as `ToString`) that have a parameter of type `IFormatProvider` all pass the `IFormatProvider.GetFormat` implementation a `Type` object that represents the `NumberFormatInfo` type. In return, they expect the method to return a `NumberFormatInfo` object. If it does not, the custom numeric format provider is ignored, and the `NumberFormatInfo` object for the current culture is used in its place. In the example, the `TelephoneFormatter.GetFormat` method handles the possibility that it may be inappropriately passed to a numeric formatting method by examining the method parameter and returning `null` if it represents a type other than `ICustomFormatter`.

If a custom numeric format provider supports a set of format specifiers, make sure you provide a default behavior if no format specifier is supplied in the format item used in the `String.Format(IFormatProvider, String, Object[])` method call. In the example, "N" is the default format specifier. This allows for a number to be converted to a formatted telephone number by providing an explicit format specifier. The following example illustrates such a method call.

```
Console.WriteLine(String.Format(new TelephoneFormatter(), "{0:N}", 4257884748));
```

```
Console.WriteLine(String.Format(New TelephoneFormatter, "{0:N}", 4257884748))
```

But it also allows the conversion to occur if no format specifier is present. The following example illustrates such a method call.

```
Console.WriteLine(String.Format(new TelephoneFormatter(), "{0}", 4257884748));
```

```
Console.WriteLine(String.Format(New TelephoneFormatter, "{0}", 4257884748))
```

If no default format specifier is defined, your implementation of the `ICustomFormatter.Format` method should include code such as the following so that .NET can provide formatting that your code does not support.

```
if (arg is IFormattable)
    s = ((IFormattable)arg).ToString(format, formatProvider);
else if (arg != null)
    s = arg.ToString();
```

```
If TypeOf (arg) Is IFormattable Then
    s = DirectCast(arg, IFormattable).ToString(fmt, formatProvider)
ElseIf arg IsNot Nothing Then
    s = arg.ToString()
End If
```

In the case of this example, the method that implements `ICustomFormatter.Format` is intended to serve as a callback method for the `String.Format(IFormatProvider, String, Object[])` method. Therefore, it examines the `formatProvider` parameter to determine whether it contains a reference to the current `TelephoneFormatter` object. However, the method can also be called directly from code. In that case, you can use the `formatProvider` parameter to provide a `CultureInfo` or `NumberFormatInfo` object that supplies culture-specific formatting information.

How to: Round-trip Date and Time Values

9/20/2022 • 8 minutes to read • [Edit Online](#)

In many applications, a date and time value is intended to unambiguously identify a single point in time. This article shows how to save and restore a `DateTime` value, a `DateTimeOffset` value, and a date and time value with time zone information so that the restored value identifies the same time as the saved value.

Round-trip a `DateTime` value

1. Convert the `DateTime` value to its string representation by calling the `DateTime.ToString(String)` method with the "o" format specifier.
2. Save the string representation of the `DateTime` value to a file, or pass it across a process, application domain, or machine boundary.
3. Retrieve the string that represents the `DateTime` value.
4. Call the `DateTime.Parse(String, IFormatProvider, DateTimeStyles)` method, and pass `DateTimeStyles.RoundtripKind` as the value of the `styles` parameter.

The following example illustrates how to round-trip a `DateTime` value.

```
const string fileName = @"\DateFile.txt";

StreamWriter outFile = new StreamWriter(fileName);

// Save DateTime value.
DateTime dateToSave = DateTime.SpecifyKind(new DateTime(2008, 6, 12, 18, 45, 15),
                                             DateTimeKind.Local);
string dateString = dateToSave.ToString("o");
Console.WriteLine("Converted {0} ({1}) to {2}.",
                  dateToSave.ToString(),
                  dateToSave.Kind.ToString(),
                  dateString);
outFile.WriteLine(dateString);
Console.WriteLine("Wrote {0} to {1}.", dateString, fileName);
outFile.Close();

// Restore DateTime value.
DateTime restoredDate;

StreamReader inFile = new StreamReader(fileName);
dateString = inFile.ReadLine();
inFile.Close();
restoredDate = DateTime.Parse(dateString, null, DateTimeStyles.RoundtripKind);
Console.WriteLine("Read {0} ({2}) from {1}.", restoredDate.ToString(),
                  fileName,
                  restoredDate.Kind.ToString());

// The example displays the following output:
//     Converted 6/12/2008 6:45:15 PM (Local) to 2008-06-12T18:45:15.0000000-05:00.
//     Wrote 2008-06-12T18:45:15.0000000-05:00 to .\DateFile.txt.
//     Read 6/12/2008 6:45:15 PM (Local) from .\DateFile.txt.
```

```

Const fileName As String = ".\DateFile.txt"

Dim outFile As New StreamWriter(fileName)

' Save DateTime value.
Dim dateToSave As Date = DateTime.SpecifyKind(#06/12/2008 6:45:15 PM#, _
                                              DateTimeKind.Local)
Dim dateString As String = dateToSave.ToString("o")
Console.WriteLine("Converted {0} ({1}) to {2}.", dateToSave.ToString(), _
                  dateToSave.Kind.ToString(), dateString)
outFile.WriteLine(dateString)
Console.WriteLine("Wrote {0} to {1}.", dateString, fileName)
outFile.Close()

' Restore DateTime value.
Dim restoredDate As Date

Dim inFile As New StreamReader(fileName)
dateString = inFile.ReadLine()
inFile.Close()
restoredDate = DateTime.Parse(dateString, Nothing, DateTimeStyles.RoundTripKind)
Console.WriteLine("Read {0} ({2}) from {1}.", restoredDate.ToString(), _
                  fileName, restoredDate.Kind.ToString())
' The example displays the following output:
'   Converted 6/12/2008 6:45:15 PM (Local) to 2008-06-12T18:45:15.0000000-05:00.
'   Wrote 2008-06-12T18:45:15.0000000-05:00 to .\DateFile.txt.
'   Read 6/12/2008 6:45:15 PM (Local) from .\DateFile.txt.

```

When round-tripping a [DateTime](#) value, this technique successfully preserves the time for all local and universal times. For example, if a local [DateTime](#) value is saved on a system in the U.S. Pacific Standard Time zone and is restored on a system in the U.S. Central Standard Time zone, the restored date and time will be two hours later than the original time, which reflects the time difference between the two time zones. However, this technique is not necessarily accurate for unspecified times. All [DateTime](#) values whose [Kind](#) property is [Unspecified](#) are treated as if they are local times. If it's not a local time, the [DateTime](#) doesn't successfully identify the correct point in time. The workaround for this limitation is to tightly couple a date and time value with its time zone for the save and restore operation.

Round-trip a [DateTimeOffset](#) value

- Convert the [DateTimeOffset](#) value to its string representation by calling the [DateTimeOffset.ToString\(String\)](#) method with the "o" format specifier.
- Save the string representation of the [DateTimeOffset](#) value to a file, or pass it across a process, application domain, or machine boundary.
- Retrieve the string that represents the [DateTimeOffset](#) value.
- Call the [DateTimeOffset.Parse\(String, IFormatProvider, DateTimeStyles\)](#) method, and pass [DateTimeStyles.RoundtripKind](#) as the value of the `styles` parameter.

The following example illustrates how to round-trip a [DateTimeOffset](#) value.

```

const string fileName = @"\DateOff.txt";

StreamWriter outFile = new StreamWriter(fileName);

// Save DateTime value.
DateTimeOffset dateToSave = new DateTimeOffset(2008, 6, 12, 18, 45, 15,
                                                new TimeSpan(7, 0, 0));
string dateString = dateToSave.ToString("o");
Console.WriteLine("Converted {0} to {1}.", dateToSave.ToString(),
                  dateString);
outFile.WriteLine(dateString);
Console.WriteLine("Wrote {0} to {1}.", dateString, fileName);
outFile.Close();

// Restore DateTime value.
DateTimeOffset restoredDateOff;

StreamReader inFile = new StreamReader(fileName);
dateString = inFile.ReadLine();
inFile.Close();
restoredDateOff = DateTimeOffset.Parse(dateString, null,
                                         DateTimeStyles.RoundtripKind);
Console.WriteLine("Read {0} from {1}.", restoredDateOff.ToString(),
                  fileName);
// The example displays the following output:
//     Converted 6/12/2008 6:45:15 PM +07:00 to 2008-06-12T18:45:15.0000000+07:00.
//     Wrote 2008-06-12T18:45:15.0000000+07:00 to .\DateOff.txt.
//     Read 6/12/2008 6:45:15 PM +07:00 from .\DateOff.txt.

```

```

Const fileName As String = ".\DateOff.txt"

Dim outFile As New StreamWriter(fileName)

' Save DateTime value.
Dim dateToSave As New DateTimeOffset(2008, 6, 12, 18, 45, 15, _
                                       New TimeSpan(7, 0, 0))
Dim dateString As String = dateToSave.ToString("o")
Console.WriteLine("Converted {0} to {1}.", dateToSave.ToString(), dateString)
outFile.WriteLine(dateString)
Console.WriteLine("Wrote {0} to {1}.", dateString, fileName)
outFile.Close()

' Restore DateTime value.
Dim restoredDateOff As DateTimeOffset

Dim inFile As New StreamReader(fileName)
dateString = inFile.ReadLine()
inFile.Close()
restoredDateOff = DateTimeOffset.Parse(dateString, Nothing, DateTimeStyles.RoundTripKind)
Console.WriteLine("Read {0} from {1}.", restoredDateOff.ToString(), fileName)
' The example displays the following output:
'     Converted 6/12/2008 6:45:15 PM +07:00 to 2008-06-12T18:45:15.0000000+07:00.
'     Wrote 2008-06-12T18:45:15.0000000+07:00 to .\DateOff.txt.
'     Read 6/12/2008 6:45:15 PM +07:00 from .\DateOff.txt.

```

This technique always unambiguously identifies a [DateTimeOffset](#) value as a single point in time. The value can then be converted to Coordinated Universal Time (UTC) by calling the [DateTimeOffset.ToUniversalTime](#) method, or it can be converted to the time in a particular time zone by calling the [DateTimeOffset.ToOffset](#) or [TimeZoneInfo.ConvertTime\(DateTimeOffset, TimeZoneInfo\)](#) method. The major limitation of this technique is that date and time arithmetic, when performed on a [DateTimeOffset](#) value that represents the time in a particular time zone, may not produce accurate results for that time zone. This is because when a [DateTimeOffset](#) value is instantiated, it is disassociated from its time zone. Therefore, that time zone's

adjustment rules can no longer be applied when you perform date and time calculations. You can work around this problem by defining a custom type that includes both a date and time value and its accompanying time zone.

Round-trip a date and time value with its time zone

1. Define a class or a structure with two fields. The first field is either a [DateTime](#) or a [DateTimeOffset](#) object, and the second is a [TimeZoneInfo](#) object. The following example is a simple version of such a type.

```
[Serializable] public class DateInTimeZone
{
    private TimeZoneInfo tz;
    private DateTimeOffset thisDate;

    public DateInTimeZone() {}

    public DateInTimeZone(DateTimeOffset date, TimeZoneInfo timeZone)
    {
        if (timeZone == null)
            throw new ArgumentNullException("The time zone cannot be null.");

        this.thisDate = date;
        this.tz = timeZone;
    }

    public DateTimeOffset DateAndTime
    {
        get {
            return this.thisDate;
        }
        set {
            if (value.Offset != this.tz.GetUtcOffset(value))
                this.thisDate = TimeZoneInfo.ConvertTime(value, tz);
            else
                this.thisDate = value;
        }
    }

    public TimeZoneInfo TimeZone
    {
        get {
            return this.tz;
        }
    }
}
```

```

<Serializable> Public Class DateInTimeZone
    Private tz As TimeZoneInfo
    Private thisDate As DateTimeOffset

    Public Sub New()
        End Sub

    Public Sub New(date1 As DateTimeOffset, timeZone As TimeZoneInfo)
        If timeZone Is Nothing Then
            Throw New ArgumentNullException("The time zone cannot be null.")
        End If
        Me.thisDate = date1
        Me.tz = timeZone
    End Sub

    Public Property DateAndTime As DateTimeOffset
        Get
            Return Me.thisDate
        End Get
        Set
            If Value.Offset <> Me.tz.GetUtcOffset(Value) Then
                Me.thisDate = TimeZoneInfo.ConvertTime(Value, tz)
            Else
                Me.thisDate = Value
            End If
        End Set
    End Property

    Public ReadOnly Property TimeZone As TimeZoneInfo
        Get
            Return tz
        End Get
    End Property
End Class

```

2. Mark the class with the [SerializableAttribute](#) attribute.
3. Serialize the object using the [BinaryFormatter.Serialize](#) method.
4. Restore the object using the [Deserialize](#) method.
5. Cast (in C#) or convert (in Visual Basic) the deserialized object to an object of the appropriate type.

The following example illustrates how to round-trip an object that stores both time zone and date and time information.

```

const string fileName = @"\DateWithTz.dat";

DateTime tempDate = new DateTime(2008, 9, 3, 19, 0, 0);
TimeZoneInfo tempTz = TimeZoneInfo.FindSystemTimeZoneById("Central Standard Time");
DateInTimeZone dateWithTz = new DateInTimeZone(new DateTimeOffset(tempDate,
    tempTz.GetUtcOffset(tempDate)),
    tempTz);

// Store DateInTimeZone value to a file
FileStream outFile = new FileStream(fileName, FileMode.Create);
try
{
    BinaryFormatter formatter = new BinaryFormatter();
    formatter.Serialize(outFile, dateWithTz);
    Console.WriteLine("Saving {0} {1} to {2}", dateWithTz.DateAndTime,
        dateWithTz.TimeZone.IsDaylightSavingTime(dateWithTz.DateAndTime) ?
        dateWithTz.TimeZone.DaylightName : dateWithTz.TimeZone.DaylightName,
        fileName);
}
catch (SerializationException)
{
    Console.WriteLine("Unable to serialize time data to {0}.", fileName);
}
finally
{
    outFile.Close();
}

// Retrieve DateInTimeZone value
if (File.Exists(fileName))
{
    FileStream inFile = new FileStream(fileName, FileMode.Open);
    DateInTimeZone dateWithTz2 = new DateInTimeZone();
    try
    {
        BinaryFormatter formatter = new BinaryFormatter();
        dateWithTz2 = formatter.Deserialize(inFile) as DateInTimeZone;
        Console.WriteLine("Restored {0} {1} from {2}", dateWithTz2.DateAndTime,
            dateWithTz2.TimeZone.IsDaylightSavingTime(dateWithTz2.DateAndTime) ?
            dateWithTz2.TimeZone.DaylightName : dateWithTz2.TimeZone.DaylightName,
            fileName);
    }
    catch (SerializationException)
    {
        Console.WriteLine("Unable to retrieve date and time information from {0}",
            fileName);
    }
    finally
    {
        inFile.Close();
    }
}
// This example displays the following output to the console:
//   Saving 9/3/2008 7:00:00 PM -05:00 Central Daylight Time to .\DateWithTz.dat
//   Restored 9/3/2008 7:00:00 PM -05:00 Central Daylight Time from .\DateWithTz.dat

```

```

Const fileName As String = ".\DateWithTz.dat"

Dim tempDate As Date = #9/3/2008 7:00:00 PM#
Dim tempTz As TimeZoneInfo = TimeZoneInfo.FindSystemTimeZoneById("Central Standard Time")
Dim dateWithTz As New DateInTimeZone(New DateTimeOffset(tempDate, _
                                         tempTz.GetUtcOffset(tempDate)), _
                                         tempTz)

' Store DateInTimeZone value to a file
Dim outFile As New FileStream(fileName, FileMode.Create)
Try
    Dim formatter As New BinaryFormatter()
    formatter.Serialize(outFile, dateWithTz)
    Console.WriteLine("Saving {0} {1} to {2}", dateWithTz.DateAndTime, _
                      IIf(dateWithTz.TimeZone.IsDaylightSavingTime(dateWithTz.DateAndTime), _
                          dateWithTz.TimeZone.DaylightName, dateWithTz.TimeZone.StandardName), _
                      fileName)
Catch e As SerializationException
    Console.WriteLine("Unable to serialize time data to {0}.", fileName)
Finally
    outFile.Close()
End Try

' Retrieve DateInTimeZone value
If File.Exists(fileName) Then
    Dim inFile As New FileStream(fileName, FileMode.Open)
    Dim dateWithTz2 As New DateInTimeZone()
    Try
        Dim formatter As New BinaryFormatter()
        dateWithTz2 = DirectCast(formatter.Deserialize(inFile), DateInTimeZone)
        Console.WriteLine("Restored {0} {1} from {2}", dateWithTz2.DateAndTime, _
                          IIf(dateWithTz2.TimeZone.IsDaylightSavingTime(dateWithTz2.DateAndTime), _
                              dateWithTz2.TimeZone.DaylightName, dateWithTz2.TimeZone.StandardName), _
                          fileName)
    Catch e As SerializationException
        Console.WriteLine("Unable to retrieve date and time information from {0}", _
                          fileName)
    Finally
        inFile.Close()
    End Try
End If
' This example displays the following output to the console:
'   Saving 9/3/2008 7:00:00 PM -05:00 Central Daylight Time to .\DateWithTz.dat
'   Restored 9/3/2008 7:00:00 PM -05:00 Central Daylight Time from .\DateWithTz.dat

```

This technique should always unambiguously reflect the correct point of time both before and after it is saved and restored, provided that the implementation of the combined date and time and time zone object does not allow the date value to become out of sync with the time zone value.

Compile the code

These examples require that:

- The following namespaces be imported with C# `using` directives or Visual Basic `Imports` statements:
 - [System](#) (C# only)
 - [System.Globalization](#)
 - [System.IO](#)
 - [System.Runtime.Serialization](#)
 - [System.Runtime.Serialization.Formatters.Binary](#)

- Each code example, other than the `DateInTimeZone` class, be included in a class or Visual Basic module, wrapped in methods, and called from the `Main` method.

See also

- [Choosing Between DateTime, DateTimeOffset, TimeSpan, and TimeZoneInfo](#)
- [Standard Date and Time Format Strings](#)

How to: Display Milliseconds in Date and Time Values

9/20/2022 • 5 minutes to read • [Edit Online](#)

The default date and time formatting methods, such as `DateTime.ToString()`, include the hours, minutes, and seconds of a time value but exclude its milliseconds component. This article shows how to include a date and time's millisecond component in formatted date and time strings.

To display the millisecond component of a `DateTime` value

1. If you're working with the string representation of a date, convert it to a `DateTime` or a `DateTimeOffset` value by using the static `DateTime.Parse(String)` or `DateTimeOffset.Parse(String)` method.
2. To extract the string representation of a time's millisecond component, call the date and time value's `DateTime.ToString(String)` or `ToString` method, and pass the `fff` or `FFF` custom format pattern either alone or with other custom format specifiers as the `format` parameter.

TIP

The millisecond separator is specified by the `System.Globalization.NumberFormatInfo.NumberDecimalSeparator` property.

Example

The example displays the millisecond component of a `DateTime` and a `DateTimeOffset` value to the console, both alone and included in a longer date and time string.

```
using System.Globalization;
using System.Text.RegularExpressions;

string dateString = "7/16/2008 8:32:45.126 AM";

try
{
    DateTime dateValue = DateTime.Parse(dateString);
    DateTimeOffset dateOffsetValue = DateTimeOffset.Parse(dateString);

    // Display Millisecond component alone.
    Console.WriteLine("Millisecond component only: {0}",
                      dateValue.ToString("fff"));
    Console.WriteLine("Millisecond component only: {0}",
                      dateOffsetValue.ToString("fff"));

    // Display Millisecond component with full date and time.
    Console.WriteLine("Date and Time with Milliseconds: {0}",
                      dateValue.ToString("MM/dd/yyyy hh:mm:ss.fff tt"));
    Console.WriteLine("Date and Time with Milliseconds: {0}",
                      dateOffsetValue.ToString("MM/dd/yyyy hh:mm:ss.fff tt"));

    string fullPattern = DateTimeFormatInfo.CurrentInfo.FullDateTimePattern;

    // Create a format similar to .fff but based on the current culture.
    string millisecondFormat = $"{NumberFormatInfo.CurrentInfo.NumberDecimalSeparator}fff";

    // Append millisecond pattern to current culture's full date time pattern.
    fullPattern = Regex.Replace(fullPattern, "(:ss|:s)", $"$1{millisecondFormat}");

    // Display Millisecond component with modified full date and time pattern.
    Console.WriteLine("Modified full date time pattern: {0}",
                      dateValue.ToString(fullPattern));
    Console.WriteLine("Modified full date time pattern: {0}",
                      dateOffsetValue.ToString(fullPattern));
}

catch (FormatException)
{
    Console.WriteLine("Unable to convert {0} to a date.", dateString);
}

// The example displays the following output if the current culture is en-US:
//    Millisecond component only: 126
//    Millisecond component only: 126
//    Date and Time with Milliseconds: 07/16/2008 08:32:45.126 AM
//    Date and Time with Milliseconds: 07/16/2008 08:32:45.126 AM
//    Modified full date time pattern: Wednesday, July 16, 2008 8:32:45.126 AM
//    Modified full date time pattern: Wednesday, July 16, 2008 8:32:45.126 AM
```

```

Imports System.Globalization
Imports System.Text.RegularExpressions

Module MillisecondDisplay
    Public Sub Main()

        Dim dateString As String = "7/16/2008 8:32:45.126 AM"

        Try
            Dim dateValue As Date = Date.Parse(dateString)
            Dim dateOffsetValue As DateTimeOffset = DateTimeOffset.Parse(dateString)

            ' Display Millisecond component alone.
            Console.WriteLine("Millisecond component only: {0}", _
                dateValue.ToString("fff"))
            Console.WriteLine("Millisecond component only: {0}", _
                dateOffsetValue.ToString("fff"))

            ' Display Millisecond component with full date and time.
            Console.WriteLine("Date and Time with Milliseconds: {0}", _
                dateValue.ToString("MM/dd/yyyy hh:mm:ss.fff tt"))
            Console.WriteLine("Date and Time with Milliseconds: {0}", _
                dateOffsetValue.ToString("MM/dd/yyyy hh:mm:ss.fff tt"))

            Dim fullPattern As String = DateTimeFormatInfo.CurrentInfo.FullDateTimePattern

            ' Create a format similar to .fff but based on the current culture.
            Dim millisecondFormat as String = $"{NumberFormatInfo.CurrentInfo.NumberDecimalSeparator}fff"

            ' Append millisecond pattern to current culture's full date time pattern.
            fullPattern = Regex.Replace(fullPattern, "(:ss|:s)", $"${1}{millisecondFormat}")

            ' Display Millisecond component with modified full date and time pattern.
            Console.WriteLine("Modified full date time pattern: {0}", _
                dateValue.ToString(fullPattern))
            Console.WriteLine("Modified full date time pattern: {0}", _
                dateOffsetValue.ToString(fullPattern))

            Catch e As FormatException
                Console.WriteLine("Unable to convert {0} to a date.", dateString)
            End Try
        End Sub
    End Module
    ' The example displays the following output if the current culture is en-US:
    '     Millisecond component only: 126
    '     Millisecond component only: 126
    '     Date and Time with Milliseconds: 07/16/2008 08:32:45.126 AM
    '     Date and Time with Milliseconds: 07/16/2008 08:32:45.126 AM
    '     Modified full date time pattern: Wednesday, July 16, 2008 8:32:45.126 AM
    '     Modified full date time pattern: Wednesday, July 16, 2008 8:32:45.126 AM

```

The `fff` format pattern includes any trailing zeros in the millisecond value. The `FFF` format pattern suppresses them. The difference is illustrated in the following example.

```

DateTime dateValue = new DateTime(2008, 7, 16, 8, 32, 45, 180);
Console.WriteLine(dateValue.ToString("fff"));
Console.WriteLine(dateValue.ToString("FFF"));
// The example displays the following output to the console:
//      180
//      18

```

```

Dim dateValue As New Date(2008, 7, 16, 8, 32, 45, 180)
Console.WriteLine(dateValue.ToString("fff"))
Console.WriteLine(dateValue.ToString("FFF"))
' The example displays the following output to the console:
'   180
'   18

```

A problem with defining a complete custom format specifier that includes the millisecond component of a date and time is that it defines a hard-coded format that may not correspond to the arrangement of time elements in the application's current culture. A better alternative is to retrieve one of the date and time display patterns defined by the current culture's [DateTimeFormatInfo](#) object and modify it to include milliseconds. The example also illustrates this approach. It retrieves the current culture's full date and time pattern from the [DateTimeFormatInfo.FullDateTimePattern](#) property, and then inserts the custom pattern `fff` along with the current culture's millisecond separator. The example uses a regular expression to do this operation in a single method call.

You can also use a custom format specifier to display a fractional part of seconds other than milliseconds. For example, the `f` or `F` custom format specifier displays tenths of a second, the `ff` or `FF` custom format specifier displays hundredths of a second, and the `ffff` or `FFFF` custom format specifier displays ten thousandths of a second. Fractional parts of a millisecond are truncated instead of rounded in the returned string. These format specifiers are used in the following example.

```

DateTime dateValue = new DateTime(2008, 7, 16, 8, 32, 45, 180);
Console.WriteLine("{0} seconds", dateValue.ToString("s.f"));
Console.WriteLine("{0} seconds", dateValue.ToString("s.ff"));
Console.WriteLine("{0} seconds", dateValue.ToString("s.ffff"));
// The example displays the following output to the console:
//   45.1 seconds
//   45.18 seconds
//   45.1800 seconds

```

```

Dim dateValue As New DateTime(2008, 7, 16, 8, 32, 45, 180)
Console.WriteLine("{0} seconds", dateValue.ToString("s.f"))
Console.WriteLine("{0} seconds", dateValue.ToString("s.ff"))
Console.WriteLine("{0} seconds", dateValue.ToString("s.ffff"))
' The example displays the following output to the console:
'   45.1 seconds
'   45.18 seconds
'   45.1800 seconds

```

NOTE

It is possible to display very small fractional units of a second, such as ten thousandths of a second or hundred-thousandths of a second. However, these values may not be meaningful. The precision of a date and time value depends on the resolution of the operating system clock. For more information, see the API your operating system uses:

- Windows 7: [GetSystemTimeAsFileTime](#)
- Windows 8 and above: [GetSystemTimePreciseAsFileTime](#)
- Linux and macOS: [clock_gettime](#)

See also

- [DateTimeFormatInfo](#)
- [Custom Date and Time Format Strings](#)

How to: Display Dates in Non-Gregorian Calendars

9/20/2022 • 8 minutes to read • [Edit Online](#)

The [DateTime](#) and [DateTimeOffset](#) types use the Gregorian calendar as their default calendar. This means that calling a date and time value's `ToString` method displays the string representation of that date and time in the Gregorian calendar, even if that date and time was created using another calendar. This is illustrated in the following example, which uses two different ways to create a date and time value with the Persian calendar, but still displays those date and time values in the Gregorian calendar when it calls the `ToString` method. This example reflects two commonly used but incorrect techniques for displaying the date in a particular calendar.

```
PersianCalendar persianCal = new PersianCalendar();

DateTime persianDate = persianCal.ToDateTime(1387, 3, 18, 12, 0, 0, 0);
Console.WriteLine(persianDate.ToString());

persianDate = new DateTime(1387, 3, 18, persianCal);
Console.WriteLine(persianDate.ToString());
// The example displays the following output to the console:
//      6/7/2008 12:00:00 PM
//      6/7/2008 12:00:00 AM
```

```
Dim persianCal As New PersianCalendar()

Dim persianDate As Date = persianCal.ToDateTime(1387, 3, 18, _
                                               12, 0, 0, 0)
Console.WriteLine(persianDate.ToString())

persianDate = New DateTime(1387, 3, 18, persianCal)
Console.WriteLine(persianDate.ToString())
' The example displays the following output to the console:
'      6/7/2008 12:00:00 PM
'      6/7/2008 12:00:00 AM
```

Two different techniques can be used to display the date in a particular calendar. The first requires that the calendar be the default calendar for a particular culture. The second can be used with any calendar.

To display the date for a culture's default calendar

1. Instantiate a calendar object derived from the [Calendar](#) class that represents the calendar to be used.
2. Instantiate a [CultureInfo](#) object representing the culture whose formatting will be used to display the date.
3. Call the [Array.Exists](#) method to determine whether the calendar object is a member of the array returned by the [CultureInfo.OptionalCalendars](#) property. This indicates that the calendar can serve as the default calendar for the [CultureInfo](#) object. If it is not a member of the array, follow the instructions in the "To Display the Date in Any Calendar" section.
4. Assign the calendar object to the [Calendar](#) property of the [DateTimeFormatInfo](#) object returned by the [CultureInfo.DateTimeFormat](#) property.

NOTE

The [CultureInfo](#) class also has a [Calendar](#) property. However, it is read-only and constant; it does not change to reflect the new default calendar assigned to the [DateTimeFormatInfo.Calendar](#) property.

5. Call either the [ToString](#) or the [ToString](#) method, and pass it the [CultureInfo](#) object whose default calendar was modified in the previous step.

To display the date in any calendar

1. Instantiate a calendar object derived from the [Calendar](#) class that represents the calendar to be used.
2. Determine which date and time elements should appear in the string representation of the date and time value.
3. For each date and time element that you want to display, call the calendar object's `Get ...` method. The following methods are available:
 - [GetYear](#), to display the year in the appropriate calendar.
 - [GetMonth](#), to display the month in the appropriate calendar.
 - [GetDayOfMonth](#), to display the number of the day of the month in the appropriate calendar.
 - [GetHour](#), to display the hour of the day in the appropriate calendar.
 - [GetMinute](#), to display the minutes in the hour in the appropriate calendar.
 - [GetSecond](#), to display the seconds in the minute in the appropriate calendar.
 - [GetMilliseconds](#), to display the milliseconds in the second in the appropriate calendar.

Example

The example displays a date using two different calendars. It displays the date after defining the Hijri calendar as the default calendar for the ar-JO culture, and displays the date using the Persian calendar, which is not supported as an optional calendar by the fa-IR culture.

```
using System;
using System.Globalization;

public class CalendarDates
{
    public static void Main()
    {
        HijriCalendar hijriCal = new HijriCalendar();
        CalendarUtility hijriUtil = new CalendarUtility(hijriCal);
        DateTime dateValue1 = new DateTime(1429, 6, 29, hijriCal);
        DateTimeOffset dateValue2 = new DateTimeOffset(dateValue1,
            TimeZoneInfo.Local.GetUtcOffset(dateValue1));
        CultureInfo jc = CultureInfo.CreateSpecificCulture("ar-JO");

        // Display the date using the Gregorian calendar.
        Console.WriteLine("Using the system default culture: {0}",
            dateValue1.ToString("d"));
        // Display the date using the ar-JO culture's original default calendar.
        Console.WriteLine("Using the ar-JO culture's original default calendar: {0}",
            dateValue1.ToString("d", jc));
        // Display the date using the Hijri calendar.
        Console.WriteLine("Using the ar-JO culture with Hijri as the default calendar:");
        // Display a Date value.
        Console.WriteLine(hijriUtil.DisplayDate(dateValue1, jc));
        // Display a DateTimeOffset value.
    }
}
```

```

        Console.WriteLine(hijriUtil.DisplayDate(dateValue2, jc));

        Console.WriteLine();

        PersianCalendar persianCal = new PersianCalendar();
        CalendarUtility persianUtil = new CalendarUtility(persianCal);
        CultureInfo ic = CultureInfo.CreateSpecificCulture("fa-IR");

        // Display the date using the ir-FA culture's default calendar.
        Console.WriteLine("Using the ir-FA culture's default calendar: {0}",
            dateValue1.ToString("d", ic));
        // Display a Date value.
        Console.WriteLine(persianUtil.DisplayDate(dateValue1, ic));
        // Display a DateTimeOffset value.
        Console.WriteLine(persianUtil.DisplayDate(dateValue2, ic));
    }
}

public class CalendarUtility
{
    private Calendar thisCalendar;
    private CultureInfo targetCulture;

    public CalendarUtility(Calendar cal)
    {
        this.thisCalendar = cal;
    }

    private bool CalendarExists(CultureInfo culture)
    {
        this.targetCulture = culture;
        return Array.Exists(this.targetCulture.OptionalCalendars,
            this.HasSameName);
    }

    private bool HasSameName(Calendar cal)
    {
        if (cal.ToString() == thisCalendar.ToString())
            return true;
        else
            return false;
    }

    public string DisplayDate(DateTime dateToDisplay, CultureInfo culture)
    {
        DateTimeOffset displayOffsetDate = dateToDisplay;
        return DisplayDate(displayOffsetDate, culture);
    }

    public string DisplayDate(DateTimeOffset dateToDisplay,
        CultureInfo culture)
    {
        string specifier = "yyyy/MM/dd";

        if (this.CalendarExists(culture))
        {
            Console.WriteLine("Displaying date in supported {0} calendar...",
                this.thisCalendar.GetType().Name);
            culture.DateTimeFormat.Calendar = this.thisCalendar;
            return dateToDisplay.ToString(specifier, culture);
        }
        else
        {
            Console.WriteLine("Displaying date in unsupported {0} calendar...",
                thisCalendar.GetType().Name);

            string separator = targetCulture.DateTimeFormat.DateSeparator;
            return thisCalendar.GetYear(dateToDisplay.DateTime).ToString("0000") +

```

```

        separator +
        thisCalendar.GetMonth(dateToDisplay.DateTime).ToString("00") +
        separator +
        thisCalendar.GetDayOfMonth(dateToDisplay.DateTime).ToString("00");
    }
}
}

// The example displays the following output to the console:
// Using the system default culture: 7/3/2008
// Using the ar-JO culture's original default calendar: 03/07/2008
// Using the ar-JO culture with Hijri as the default calendar:
// Displaying date in supported HijriCalendar calendar...
// 1429/06/29
// Displaying date in supported HijriCalendar calendar...
// 1429/06/29
//
// Using the ir-FA culture's default calendar: 7/3/2008
// Displaying date in unsupported PersianCalendar calendar...
// 1387/04/13
// Displaying date in unsupported PersianCalendar calendar...
// 1387/04/13

```

```

Imports System.Globalization

Public Class CalendarDates
    Public Shared Sub Main()
        Dim hijriCal As New HijriCalendar()
        Dim hijriUtil As New CalendarUtility(hijriCal)
        Dim dateValue1 As Date = New Date(1429, 6, 29, hijriCal)
        Dim dateValue2 As DateTimeOffset = New DateTimeOffset(dateValue1, _
            TimeZoneInfo.Local.GetUtcOffset(dateValue1))
        Dim jc As CultureInfo = CultureInfo.CreateSpecificCulture("ar-JO")

        ' Display the date using the Gregorian calendar.
        Console.WriteLine("Using the system default culture: {0}", _
            dateValue1.ToString("d"))
        ' Display the date using the ar-JO culture's original default calendar.
        Console.WriteLine("Using the ar-JO culture's original default calendar: {0}", _
            dateValue1.ToString("d", jc))
        ' Display the date using the Hijri calendar.
        Console.WriteLine("Using the ar-JO culture with Hijri as the default calendar:")
        ' Display a Date value.
        Console.WriteLine(hijriUtil.DisplayDate(dateValue1, jc))
        ' Display a DateTimeOffset value.
        Console.WriteLine(hijriUtil.DisplayDate(dateValue2, jc))

        Console.WriteLine()

        Dim persianCal As New PersianCalendar()
        Dim persianUtil As New CalendarUtility(persianCal)
        Dim ic As CultureInfo = CultureInfo.CreateSpecificCulture("fa-IR")

        ' Display the date using the ir-FA culture's default calendar.
        Console.WriteLine("Using the ir-FA culture's default calendar: {0}", _
            dateValue1.ToString("d", ic))
        ' Display a Date value.
        Console.WriteLine(persianUtil.DisplayDate(dateValue1, ic))
        ' Display a DateTimeOffset value.
        Console.WriteLine(persianUtil.DisplayDate(dateValue2, ic))
    End Sub
End Class

Public Class CalendarUtility
    Private thisCalendar As Calendar
    Private targetCulture As CultureInfo

    Public Sub New(cal As Calendar)

```

```

        Me.thisCalendar = cal
    End Sub

    Private Function CalendarExists(culture As CultureInfo) As Boolean
        Me.targetCulture = culture
        Return Array.Exists(Me.targetCulture.OptionalCalendars, _
            AddressOf Me.HasSameName)
    End Function

    Private Function HasSameName(cal As Calendar) As Boolean
        If cal.ToString() = thisCalendar.ToString() Then
            Return True
        Else
            Return False
        End If
    End Function

    Public Function DisplayDate(dateToDisplay As Date, _
                                culture As CultureInfo) As String
        Dim displayOffsetDate As DateTimeOffset = dateToDisplay
        Return DisplayDate(displayOffsetDate, culture)
    End Function

    Public Function DisplayDate(dateToDisplay As DateTimeOffset, _
                                culture As CultureInfo) As String
        Dim specifier As String = "yyyy/MM/dd"

        If Me.CalendarExists(culture) Then
            Console.WriteLine("Displaying date in supported {0} calendar...", _
                thisCalendar.GetType().Name)
            culture.DateTimeFormat.Calendar = Me.thisCalendar
            Return dateToDisplay.ToString(specifier, culture)
        Else
            Console.WriteLine("Displaying date in unsupported {0} calendar...", _
                thisCalendar.GetType().Name)

            Dim separator As String = targetCulture.DateTimeFormat.DateSeparator

            Return thisCalendar.GetYear(dateToDisplay.DateTime).ToString("0000") & separator & _
                thisCalendar.GetMonth(dateToDisplay.DateTime).ToString("00") & separator & _
                thisCalendar.GetDayOfMonth(dateToDisplay.DateTime).ToString("00")
        End If
    End Function
End Class

' The example displays the following output to the console:
'   Using the system default culture: 7/3/2008
'   Using the ar-JO culture's original default calendar: 03/07/2008
'   Using the ar-JO culture with Hijri as the default calendar:
'   Displaying date in supported HijriCalendar calendar...
'   1429/06/29
'   Displaying date in supported PersianCalendar calendar...
'   1429/06/29
'
'   Using the ir-FA culture's default calendar: 7/3/2008
'   Displaying date in unsupported PersianCalendar calendar...
'   1387/04/13
'   Displaying date in unsupported PersianCalendar calendar...
'   1387/04/13

```

Each [CultureInfo](#) object can support one or more calendars, which are indicated by the [OptionalCalendars](#) property. One of these is designated as the culture's default calendar and is returned by the read-only [CultureInfo.Calendar](#) property. Another of the optional calendars can be designated as the default by assigning a [Calendar](#) object that represents that calendar to the [DateTimeFormatInfo.Calendar](#) property returned by the [CultureInfo.DateTimeFormat](#) property. However, some calendars, such as the Persian calendar represented by the [PersianCalendar](#) class, do not serve as optional calendars for any culture.

The example defines a reusable calendar utility class, `CalendarUtility`, to handle many of the details of generating the string representation of a date using a particular calendar. The `CalendarUtility` class has the following members:

- A parameterized constructor whose single parameter is a `Calendar` object in which a date is to be represented. This is assigned to a private field of the class.
- `CalendarExists`, a private method that returns a Boolean value indicating whether the calendar represented by the `CalendarUtility` object is supported by the `CultureInfo` object that is passed to the method as a parameter. The method wraps a call to the `Array.Exists` method, to which it passes the `CultureInfo.OptionalCalendars` array.
- `HasSameName`, a private method assigned to the `Predicate<T>` delegate that is passed as a parameter to the `Array.Exists` method. Each member of the array is passed to the method until the method returns `true`. The method determines whether the name of an optional calendar is the same as the calendar represented by the `CalendarUtility` object.
- `DisplayDate`, an overloaded public method that is passed two parameters: either a `DateTime` or `DateTimeOffset` value to express in the calendar represented by the `CalendarUtility` object; and the culture whose formatting rules are to be used. Its behavior in returning the string representation of a date depends on whether the target calendar is supported by the culture whose formatting rules are to be used.

Regardless of the calendar used to create a `DateTime` or `DateTimeOffset` value in this example, that value is typically expressed as a Gregorian date. This is because the `DateTime` and `DateTimeOffset` types do not preserve any calendar information. Internally, they are represented as the number of ticks that have elapsed since midnight of January 1, 0001. The interpretation of that number depends on the calendar. For most cultures, the default calendar is the Gregorian calendar.

Character encoding in .NET

9/20/2022 • 17 minutes to read • [Edit Online](#)

This article provides an introduction to character encoding systems that are used by .NET. The article explains how the `String`, `Char`, `Rune`, and `StringInfo` types work with Unicode, UTF-16, and UTF-8.

The term *character* is used here in the general sense of *what a reader perceives as a single display element*. Common examples are the letter "a", the symbol "@", and the emoji "𩿻". Sometimes what looks like one character is actually composed of multiple independent display elements, as the section on [grapheme clusters](#) explains.

The string and char types

An instance of the `string` class represents some text. A `string` is logically a sequence of 16-bit values, each of which is an instance of the `char` struct. The `string.Length` property returns the number of `char` instances in the `string` instance.

The following sample function prints out the values in hexadecimal notation of all the `char` instances in a `string`:

```
void PrintChars(string s)
{
    Console.WriteLine($"\"{s}\".Length = {s.Length}");
    for (int i = 0; i < s.Length; i++)
    {
        Console.WriteLine($"s[{i}] = '{s[i]}' ('\\u{(int)s[i]:x4}')");
    }
    Console.WriteLine();
}
```

Pass the string "Hello" to this function, and you get the following output:

```
PrintChars("Hello");
```

```
"Hello".Length = 5
s[0] = 'H' ('\u0048')
s[1] = 'e' ('\u0065')
s[2] = 'l' ('\u006c')
s[3] = 'l' ('\u006c')
s[4] = 'o' ('\u006f')
```

Each character is represented by a single `char` value. That pattern holds true for most of the world's languages. For example, here's the output for two Chinese characters that sound like *nǐ hǎo* and mean *Hello*.

```
PrintChars("你好");
```

```
"你好".Length = 2
s[0] = '你' ('\u4f60')
s[1] = '好' ('\u597d')
```

However, for some languages and for some symbols and emoji, it takes two `char` instances to represent a single character. For example, compare the characters and `char` instances in the word that means *Osage* in the Osage language:

```
PrintChars("ဗော်ဗော် ဗုံ");
```

```
"ဗော်ဗော် ဗုံ".Length = 17
s[0] = 'ဗ' ('\ud801')
s[1] = 'ဗ' ('\udccf')
s[2] = 'ဗ' ('\ud801')
s[3] = 'ဗ' ('\udcd8')
s[4] = 'ဗ' ('\ud801')
s[5] = 'ဗ' ('\udcfb')
s[6] = 'ဗ' ('\ud801')
s[7] = 'ဗ' ('\udcd8')
s[8] = 'ဗ' ('\ud801')
s[9] = 'ဗ' ('\udcfb')
s[10] = 'ဗ' ('\ud801')
s[11] = 'ဗ' ('\udcdf')
s[12] = ' ' ('\u0020')
s[13] = 'ဗ' ('\ud801')
s[14] = 'ဗ' ('\udcbb')
s[15] = 'ဗ' ('\ud801')
s[16] = 'ဗ' ('\udcdf')
```

In the preceding example, each character except the space is represented by two `char` instances.

A single Unicode emoji is also represented by two `char`s, as seen in the following example showing an ox emoji:

```
"ဗ".Length = 2
s[0] = 'ဗ' ('\ud83d')
s[1] = 'ဗ' ('\udc02')
```

These examples show that the value of `string.Length`, which indicates the number of `char` instances, doesn't necessarily indicate the number of displayed characters. A single `char` instance by itself doesn't necessarily represent a character.

The `char` pairs that map to a single character are called *surrogate pairs*. To understand how they work, you need to understand Unicode and UTF-16 encoding.

Unicode code points

Unicode is an international encoding standard for use on various platforms and with various languages and scripts.

The Unicode Standard defines over 1.1 million [code points](#). A code point is an integer value that can range from 0 to [U+10FFFF](#) (decimal 1,114,111). Some code points are assigned to letters, symbols, or emoji. Others are assigned to actions that control how text or characters are displayed, such as advance to a new line. Many code points are not yet assigned.

Here are some examples of code point assignments, with links to Unicode charts in which they appear:

DECIMAL	HEX	EXAMPLE	DESCRIPTION
10	U+000A	N/A	LINE FEED

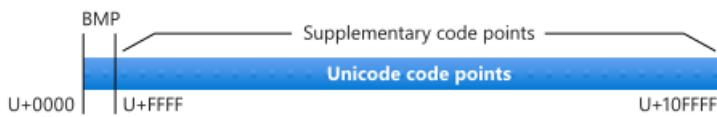
DECIMAL	HEX	EXAMPLE	DESCRIPTION
97	U+0061	a	LATIN SMALL LETTER A
562	U+0232	Ȳ	LATIN CAPITAL LETTER Y WITH MACRON
68,675	U+10C43	□	OLD TURKIC LETTER ORKHON AT
127,801	U+1F339	🌹	ROSE emoji

Code points are customarily referred to by using the syntax `U+xxxx`, where `xxxx` is the hex-encoded integer value.

Within the full range of code points there are two subranges:

- The **Basic Multilingual Plane (BMP)** in the range `U+0000..U+FFFF`. This 16-bit range provides 65,536 code points, enough to cover the majority of the world's writing systems.
- **Supplementary code points** in the range `U+10000..U+10FFFF`. This 21-bit range provides more than a million additional code points that can be used for less well-known languages and other purposes such as emojis.

The following diagram illustrates the relationship between the BMP and the supplementary code points.



UTF-16 code units

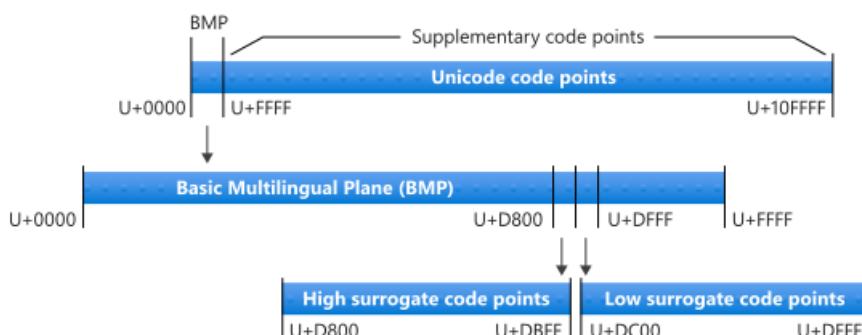
16-bit Unicode Transformation Format ([UTF-16](#)) is a character encoding system that uses 16-bit *code units* to represent Unicode code points. .NET uses UTF-16 to encode the text in a `string`. A `char` instance represents a 16-bit code unit.

A single 16-bit code unit can represent any code point in the 16-bit range of the Basic Multilingual Plane. But for a code point in the supplementary range, two `char` instances are needed.

Surrogate pairs

The translation of two 16-bit values to a single 21-bit value is facilitated by a special range called the *surrogate code points*, from `U+D800` to `U+DFFF` (decimal 55,296 to 57,343), inclusive.

The following diagram illustrates the relationship between the BMP and the surrogate code points.



When a *high surrogate* code point (`U+D800..U+DBFF`) is immediately followed by a *low surrogate* code point (

`U+DC00..U+DFFF`), the pair is interpreted as a supplementary code point by using the following formula:

```
code point = 0x10000 +
((high surrogate code point - 0xD800) * 0x0400) +
(low surrogate code point - 0xDC00)
```

Here's the same formula using decimal notation:

```
code point = 65,536 +
((high surrogate code point - 55,296) * 1,024) +
(low surrogate code point - 56,320)
```

A *high* surrogate code point doesn't have a higher number value than a *low* surrogate code point. The high surrogate code point is called "high" because it's used to calculate the higher-order 10 bits of a 20-bit code point range. The low surrogate code point is used to calculate the lower-order 10 bits.

For example, the actual code point that corresponds to the surrogate pair `0xD83C` and `0xDF39` is computed as follows:

```
actual = 0x10000 + ((0xD83C - 0xD800) * 0x0400) + (0xDF39 - 0xDC00)
= 0x10000 + (          0x003C * 0x0400) +          0x0339
= 0x10000 +                  0xF000 +          0x0339
= 0x1F339
```

Here's the same calculation using decimal notation:

```
actual = 65,536 + ((55,356 - 55,296) * 1,024) + (57,145 - 56320)
= 65,536 + (          60 * 1,024) +          825
= 65,536 +          61,440 +          825
= 127,801
```

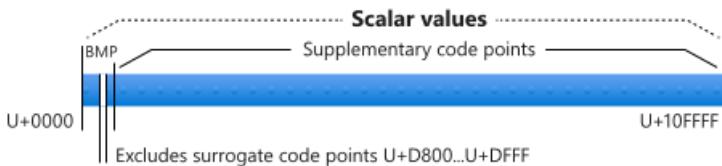
The preceding example demonstrates that `"\ud83c\udf39"` is the UTF-16 encoding of the `U+1F339 ROSE ('ຮ')` code point mentioned earlier.

Unicode scalar values

The term [Unicode scalar value](#) refers to all code points other than the surrogate code points. In other words, a scalar value is any code point that is assigned a character or can be assigned a character in the future.

"Character" here refers to anything that can be assigned to a code point, which includes such things as actions that control how text or characters are displayed.

The following diagram illustrates the scalar value code points.



The Rune type as a scalar value

Beginning with .NET Core 3.0, the `System.Text.Rune` type represents a Unicode scalar value. `Rune` is not available in .NET Core 2.x or .NET Framework 4.x.

The `Rune` constructors validate that the resulting instance is a valid Unicode scalar value, otherwise they throw an exception. The following example shows code that successfully instantiates `Rune` instances because the input

represents valid scalar values:

```
Rune a = new Rune('a');
Rune b = new Rune(0x0061);
Rune c = new Rune('\u0061');
Rune d = new Rune(0x10421);
Rune e = new Rune('\ud801', '\udc21');
```

The following example throws an exception because the code point is in the surrogate range and isn't part of a surrogate pair:

```
Rune f = new Rune('\ud801');
```

The following example throws an exception because the code point is beyond the supplementary range:

```
Rune g = new Rune(0x12345678);
```

Rune usage example: changing letter case

An API that takes a `char` and assumes it is working with a code point that is a scalar value doesn't work correctly if the `char` is from a surrogate pair. For example, consider the following method that calls [Char.ToUpperInvariant](#) on each char in a string:

```
// THE FOLLOWING METHOD SHOWS INCORRECT CODE.
// DO NOT DO THIS IN A PRODUCTION APPLICATION.
static string ConvertToUpperBadExample(string input)
{
    StringBuilder builder = new StringBuilder(input.Length);
    for (int i = 0; i < input.Length; i++) /* or 'foreach' */
    {
        builder.Append(char.ToUpperInvariant(input[i]));
    }
    return builder.ToString();
}
```

If the `input` string contains the lowercase Deseret letter `er` (✉), this code won't convert it to uppercase (✉). The code calls `char.ToUpperInvariant` separately on each surrogate code point, `U+D801` and `U+DC49`. But `U+D801` doesn't have enough information by itself to identify it as a lowercase letter, so `char.ToUpperInvariant` leaves it alone. And it handles `U+DC49` the same way. The result is that lowercase '✉' in the `input` string doesn't get converted to uppercase '✉'.

Here are two options for correctly converting a string to uppercase:

- Call [String.ToUpperInvariant](#) on the input string rather than iterating `char`-by-`char`. The `string.ToUpperInvariant` method has access to both parts of each surrogate pair, so it can handle all Unicode code points correctly.
- Iterate through the Unicode scalar values as `Rune` instances instead of `char` instances, as shown in the following example. Since a `Rune` instance is a valid Unicode scalar value, it can be passed to APIs that expect to operate on a scalar value. For example, calling [Rune.ToUpperInvariant](#) as shown in the following example gives correct results:

```

static string ConvertToUpper(string input)
{
    StringBuilder builder = new StringBuilder(input.Length);
    foreach (Rune rune in input.EnumerateRunes())
    {
        builder.Append(Rune.ToUpperInvariant(rune));
    }
    return builder.ToString();
}

```

Other Rune APIs

The `Rune` type exposes analogs of many of the `char` APIs. For example, the following methods mirror static APIs on the `char` type:

- [Rune.IsLetter](#)
- [Rune.IsWhiteSpace](#)
- [Rune.IsLetterOrDigit](#)
- [Rune.GetUnicodeCategory](#)

To get the raw scalar value from a `Rune` instance, use the [Rune.Value](#) property.

To convert a `Rune` instance back to a sequence of `char`s, use [Rune.ToString](#) or the [Rune.EncodeToUtf16](#) method.

Since any Unicode scalar value is representable by a single `char` or by a surrogate pair, any `Rune` instance can be represented by at most 2 `char` instances. Use [Rune.Utf16SequenceLength](#) to see how many `char` instances are required to represent a `Rune` instance.

For more information about the .NET `Rune` type, see the [Rune API reference](#).

Grapheme clusters

What looks like one character might result from a combination of multiple code points, so a more descriptive term that is often used in place of "character" is [grapheme cluster](#). The equivalent term in .NET is [text element](#).

Consider the `string` instances "a", "á", "á", and "𠁥". If your operating system handles them as specified by the Unicode standard, each of these `string` instances appears as a single text element or grapheme cluster. But the last two are represented by more than one scalar value code point.

- The string "a" is represented by one scalar value and contains one `char` instance.
 - `U+0061 LATIN SMALL LETTER A`
- The string "á" is represented by one scalar value and contains one `char` instance.
 - `U+00E1 LATIN SMALL LETTER A WITH ACUTE`
- The string "á" looks the same as "á" but is represented by two scalar values and contains two `char` instances.
 - `U+0061 LATIN SMALL LETTER A`
 - `U+0301 COMBINING ACUTE ACCENT`
- Finally, the string "𠁥" is represented by four scalar values and contains seven `char` instances.
 - `U+1F469 WOMAN` (supplementary range, requires a surrogate pair)
 - `U+1F3FD EMOJI MODIFIER FITZPATRICK TYPE-4` (supplementary range, requires a surrogate pair)
 - `U+200D ZERO WIDTH JOINER`

- U+1F692 FIRE ENGINE (supplementary range, requires a surrogate pair)

In some of the preceding examples - such as the combining accent modifier or the skin tone modifier - the code point does not display as a standalone element on the screen. Rather, it serves to modify the appearance of a text element that came before it. These examples show that it might take multiple scalar values to make up what we think of as a single "character," or "grapheme cluster."

To enumerate the grapheme clusters of a `string`, use the `StringInfo` class as shown in the following example. If you're familiar with Swift, the .NET `StringInfo` type is conceptually similar to Swift's `Character` type.

Example: count char, Rune, and text element instances

In .NET APIs, a grapheme cluster is called a *text element*. The following method demonstrates the differences between `char`, `Rune`, and text element instances in a `string`:

```
static void PrintTextElementCount(string s)
{
    Console.WriteLine(s);
    Console.WriteLine($"Number of chars: {s.Length}");
    Console.WriteLine($"Number of runes: {s.EnumerateRunes().Count()}");

    TextElementEnumerator enumerator = StringInfo.GetTextElementEnumerator(s);

    int textElementCount = 0;
    while (enumerator.MoveNext())
    {
        textElementCount++;
    }

    Console.WriteLine($"Number of text elements: {textElementCount}");
}
```

```
PrintTextElementCount("a");
// Number of chars: 1
// Number of runes: 1
// Number of text elements: 1

PrintTextElementCount("ää");
// Number of chars: 2
// Number of runes: 2
// Number of text elements: 1

PrintTextElementCount("消防");
// Number of chars: 7
// Number of runes: 4
// Number of text elements: 1
```

If you run this code in .NET Framework or .NET Core 3.1 or earlier, the text element count for the emoji shows 4. That is due to a bug in the `StringInfo` class that is fixed in .NET 5.

Example: splitting string instances

When splitting `string` instances, avoid splitting surrogate pairs and grapheme clusters. Consider the following example of incorrect code, which intends to insert line breaks every 10 characters in a string:

```

// THE FOLLOWING METHOD SHOWS INCORRECT CODE.
// DO NOT DO THIS IN A PRODUCTION APPLICATION.
static string InsertNewlinesEveryTencharsBadExample(string input)
{
    StringBuilder builder = new StringBuilder();

    // First, append chunks in multiples of 10 chars
    // followed by a newline.
    int i = 0;
    for (; i < input.Length - 10; i += 10)
    {
        builder.Append(input, i, 10);
        builder.AppendLine(); // newline
    }

    // Then append any leftover data followed by
    // a final newline.
    builder.Append(input, i, input.Length - i);
    builder.AppendLine(); // newline

    return builder.ToString();
}

```

Because this code enumerates `char` instances, a surrogate pair that happens to straddle a 10-`char` boundary will be split and a newline injected between them. This insertion introduces data corruption, because surrogate code points are meaningful only as pairs.

The potential for data corruption isn't eliminated if you enumerate `Rune` instances (scalar values) instead of `char` instances. A set of `Rune` instances might make up a grapheme cluster that straddles a 10-`char` boundary. If the grapheme cluster set is split up, it can't be interpreted correctly.

A better approach is to break the string by counting grapheme clusters, or text elements, as in the following example:

```

static string InsertNewlinesEveryTenTextElements(string input)
{
    StringBuilder builder = new StringBuilder();

    // Append chunks in multiples of 10 chars

    TextElementEnumerator enumerator = StringInfo.GetTextElementEnumerator(input);

    int textElementCount = 1;
    while (enumerator.MoveNext())
    {
        builder.Append(enumerator.Current);
        if (textElementCount % 10 == 0 && textElementCount > 0)
        {
            builder.AppendLine(); // newline
        }
        textElementCount++;
    }

    // Add a final newline.
    builder.AppendLine(); // newline
    return builder.ToString();
}

```

As noted earlier, however, in implementations of .NET other than .NET 5, the `StringInfo` class might handle some grapheme clusters incorrectly.

UTF-8 and UTF-32

The preceding sections focused on UTF-16 because that's what .NET uses to encode `string` instances. There are other encoding systems for Unicode - [UTF-8](#) and [UTF-32](#). These encodings use 8-bit code units and 32-bit code units, respectively.

Like UTF-16, UTF-8 requires multiple code units to represent some Unicode scalar values. UTF-32 can represent any scalar value in a single 32-bit code unit.

Here are some examples showing how the same Unicode code point is represented in each of these three Unicode encoding systems:

```
Scalar: U+0061 LATIN SMALL LETTER A ('a')
UTF-8 : [ 61 ]           (1x 8-bit code unit = 8 bits total)
UTF-16: [ 0061 ]         (1x 16-bit code unit = 16 bits total)
UTF-32: [ 00000061 ]     (1x 32-bit code unit = 32 bits total)

Scalar: U+0429 CYRILLIC CAPITAL LETTER SHCHA ('₪')
UTF-8 : [ D0 A9 ]        (2x 8-bit code units = 16 bits total)
UTF-16: [ 0429 ]          (1x 16-bit code unit = 16 bits total)
UTF-32: [ 00000429 ]      (1x 32-bit code unit = 32 bits total)

Scalar: U+A992 JAVANESE LETTER GA ('᠁')
UTF-8 : [ EA A6 92 ]      (3x 8-bit code units = 24 bits total)
UTF-16: [ A992 ]          (1x 16-bit code unit = 16 bits total)
UTF-32: [ 0000A992 ]       (1x 32-bit code unit = 32 bits total)

Scalar: U+104CC OSAGE CAPITAL LETTER TSHA ('᠁')
UTF-8 : [ F0 90 93 8C ]    (4x 8-bit code units = 32 bits total)
UTF-16: [ D801 DCCC ]      (2x 16-bit code units = 32 bits total)
UTF-32: [ 000104CC ]       (1x 32-bit code unit = 32 bits total)
```

As noted earlier, a single UTF-16 code unit from a [surrogate pair](#) is meaningless by itself. In the same way, a single UTF-8 code unit is meaningless by itself if it's in a sequence of two, three, or four used to calculate a scalar value.

NOTE

Beginning with C# 11, you can represent UTF-8 string literals using the "u8" suffix on a literal string. For more information on UTF-8 string literals, see the "string literals" section of the article on [built in reference types](#) in the C# Guide.

Endianness

In .NET, the UTF-16 code units of a string are stored in contiguous memory as a sequence of 16-bit integers (`char` instances). The bits of individual code units are laid out according to the [endianness](#) of the current architecture.

On a little-endian architecture, the string consisting of the UTF-16 code points `[D801 DCCC]` would be laid out in memory as the bytes `[0x01, 0xD8, 0xCC, 0xDC]`. On a big-endian architecture that same string would be laid out in memory as the bytes `[0xD8, 0x01, 0xDC, 0xCC]`.

Computer systems that communicate with each other must agree on the representation of data crossing the wire. Most network protocols use UTF-8 as a standard when transmitting text, partly to avoid issues that might result from a big-endian machine communicating with a little-endian machine. The string consisting of the UTF-8 code points `[F0 90 93 8C]` will always be represented as the bytes `[0xF0, 0x90, 0x93, 0x8C]` regardless of endianness.

To use UTF-8 for transmitting text, .NET applications often use code like the following example:

```
string stringToWrite = GetString();
byte[] stringAsUtf8Bytes = Encoding.UTF8.GetBytes(stringToWrite);
await outputStream.WriteAsync(stringAsUtf8Bytes, 0, stringAsUtf8Bytes.Length);
```

In the preceding example, the method `Encoding.UTF8.GetBytes` decodes the UTF-16 `string` back into a series of Unicode scalar values, then it re-encodes those scalar values into UTF-8 and places the resulting sequence into a `byte` array. The method `Encoding.UTF8.GetString` performs the opposite transformation, converting a UTF-8 `byte` array to a UTF-16 `string`.

WARNING

Since UTF-8 is commonplace on the internet, it may be tempting to read raw bytes from the wire and to treat the data as if it were UTF-8. However, you should validate that it is indeed well-formed. A malicious client might submit ill-formed UTF-8 to your service. If you operate on that data as if it were well-formed, it could cause errors or security holes in your application. To validate UTF-8 data, you can use a method like `Encoding.UTF8.GetString`, which will perform validation while converting the incoming data to a `string`.

Well-formed encoding

A well-formed Unicode encoding is a string of code units that can be decoded unambiguously and without error into a sequence of Unicode scalar values. Well-formed data can be transcoded freely back and forth between UTF-8, UTF-16, and UTF-32.

The question of whether an encoding sequence is well-formed or not is unrelated to the endianness of a machine's architecture. An ill-formed UTF-8 sequence is ill-formed in the same way on both big-endian and little-endian machines.

Here are some examples of ill-formed encodings:

- In UTF-8, the sequence `[6C C2 61]` is ill-formed because `C2` cannot be followed by `61`.
- In UTF-16, the sequence `[DC00 DD00]` (or, in C#, the string `"\udc00\udd00"`) is ill-formed because the low surrogate `DC00` cannot be followed by another low surrogate `DD00`.
- In UTF-32, the sequence `[0011ABCD]` is ill-formed because `0011ABCD` is outside the range of Unicode scalar values.

In .NET, `string` instances almost always contain well-formed UTF-16 data, but that isn't guaranteed. The following examples show valid C# code that creates ill-formed UTF-16 data in `string` instances.

- An ill-formed literal:

```
const string s = "\ud800";
```

- A substring that splits up a surrogate pair:

```
string x = "\ud83e\udd70"; // " "
string y = x.Substring(1, 1); // "\udd70" standalone low surrogate
```

APIs like `Encoding.UTF8.GetString` never return ill-formed `string` instances. `Encoding.GetString` and `Encoding.GetBytes` methods detect ill-formed sequences in the input and perform character substitution when generating the output. For example, if `Encoding.ASCII.GetString(byte[])` sees a non-ASCII byte in the input (outside the range U+0000..U+007F), it inserts a '?' into the returned `string` instance.

`Encoding.UTF8.GetString(byte[])` replaces ill-formed UTF-8 sequences with

U+FFFD REPLACEMENT CHARACTER ('◊') in the returned `string` instance. For more information, see the [Unicode Standard](#), Sections 5.22 and 3.9.

The built-in `Encoding` classes can also be configured to throw an exception rather than perform character substitution when ill-formed sequences are seen. This approach is often used in security-sensitive applications where character substitution might not be acceptable.

```
byte[] utf8Bytes = ReadFromNetwork();
UTF8Encoding encoding = new UTF8Encoding(encoderShouldEmitUTF8Identifier: false, throwOnInvalidBytes: true);
string asString = encoding.GetString(utf8Bytes); // will throw if 'utf8Bytes' is ill-formed
```

For information about how to use the built-in `Encoding` classes, see [How to use character encoding classes in .NET](#).

See also

- [String](#)
- [Char](#)
- [Rune](#)
- [Globalization and localization](#)

How to use character encoding classes in .NET

9/20/2022 • 43 minutes to read • [Edit Online](#)

This article explains how to use the classes that .NET provides for encoding and decoding text by using various encoding schemes. The instructions assume you have read [Introduction to character encoding in .NET](#).

Encoders and decoders

.NET provides encoding classes that encode and decode text by using various encoding systems. For example, the [UTF8Encoding](#) class describes the rules for encoding to, and decoding from, UTF-8. .NET uses UTF-16 encoding (represented by the [UnicodeEncoding](#) class) for `string` instances. Encoders and decoders are available for other encoding schemes.

Encoding and decoding can also include validation. For example, the [UnicodeEncoding](#) class checks all `char` instances in the surrogate range to make sure they're in valid surrogate pairs. A fallback strategy determines how an encoder handles invalid characters or how a decoder handles invalid bytes.

WARNING

.NET encoding classes provide a way to store and convert character data. They should not be used to store binary data in string form. Depending on the encoding used, converting binary data to string format with the encoding classes can introduce unexpected behavior and produce inaccurate or corrupted data. To convert binary data to a string form, use the [Convert.ToString](#) method.

All character encoding classes in .NET inherit from the [System.Text.Encoding](#) class, which is an abstract class that defines the functionality common to all character encodings. To access the individual encoding objects implemented in .NET, do the following:

- Use the static properties of the [Encoding](#) class, which return objects that represent the standard character encodings available in .NET (ASCII, UTF-7, UTF-8, UTF-16, and UTF-32). For example, the [Encoding.Unicode](#) property returns a [UnicodeEncoding](#) object. Each object uses replacement fallback to handle strings that it cannot encode and bytes that it cannot decode. For more information, see [Replacement fallback](#).
- Call the encoding's class constructor. Objects for the ASCII, UTF-7, UTF-8, UTF-16, and UTF-32 encodings can be instantiated in this way. By default, each object uses replacement fallback to handle strings that it cannot encode and bytes that it cannot decode, but you can specify that an exception should be thrown instead. For more information, see [Replacement fallback](#) and [Exception fallback](#).
- Call the [Encoding\(Int32\)](#) constructor and pass it an integer that represents the encoding. Standard encoding objects use replacement fallback, and code page and double-byte character set (DBCS) encoding objects use best-fit fallback to handle strings that they cannot encode and bytes that they cannot decode. For more information, see [Best-fit fallback](#).
- Call the [Encoding.GetEncoding](#) method, which returns any standard, code page, or DBCS encoding available in .NET. Overloads let you specify a fallback object for both the encoder and the decoder.

You can retrieve information about all the encodings available in .NET by calling the [Encoding.GetEncodings](#) method. .NET supports the character encoding schemes listed in the following table.

ENCODING CLASS	DESCRIPTION
ASCII	Encodes a limited range of characters by using the lower seven bits of a byte. Because this encoding only supports character values from <code>U+0000</code> through <code>U+007F</code> , in most cases it is inadequate for internationalized applications.
UTF-7	Represents characters as sequences of 7-bit ASCII characters. Non-ASCII Unicode characters are represented by an escape sequence of ASCII characters. UTF-7 supports protocols such as email and newsgroup. However, UTF-7 is not particularly secure or robust. In some cases, changing one bit can radically alter the interpretation of an entire UTF-7 string. In other cases, different UTF-7 strings can encode the same text. For sequences that include non-ASCII characters, UTF-7 requires more space than UTF-8, and encoding/decoding is slower. Consequently, you should use UTF-8 instead of UTF-7 if possible.
UTF-8	Represents each Unicode code point as a sequence of one to four bytes. UTF-8 supports 8-bit data sizes and works well with many existing operating systems. For the ASCII range of characters, UTF-8 is identical to ASCII encoding and allows a broader set of characters. However, for Chinese-Japanese-Korean (CJK) scripts, UTF-8 can require three bytes for each character, and can cause larger data sizes than UTF-16. Sometimes the amount of ASCII data, such as HTML tags, justifies the increased size for the CJK range.
UTF-16	Represents each Unicode code point as a sequence of one or two 16-bit integers. Most common Unicode characters require only one UTF-16 code point, although Unicode supplementary characters (U+10000 and greater) require two UTF-16 surrogate code points. Both little-endian and big-endian byte orders are supported. UTF-16 encoding is used by the common language runtime to represent Char and String values, and it is used by the Windows operating system to represent <code>WCHAR</code> values.
UTF-32	Represents each Unicode code point as a 32-bit integer. Both little-endian and big-endian byte orders are supported. UTF-32 encoding is used when applications want to avoid the surrogate code point behavior of UTF-16 encoding on operating systems for which encoded space is too important. Single glyphs rendered on a display can still be encoded with more than one UTF-32 character.

ENCODING CLASS	DESCRIPTION
ANSI/ISO encoding	Provides support for a variety of code pages. On Windows operating systems, code pages are used to support a specific language or group of languages. For a table that lists the code pages supported by .NET, see the Encoding class. You can retrieve an encoding object for a particular code page by calling the Encoding.GetEncoding(Int32) method. A code page contains 256 code points and is zero-based. In most code pages, code points 0 through 127 represent the ASCII character set, and code points 128 through 255 differ significantly between code pages. For example, code page 1252 provides the characters for Latin writing systems, including English, German, and French. The last 128 code points in code page 1252 contain the accent characters. Code page 1253 provides character codes that are required in the Greek writing system. The last 128 code points in code page 1253 contain the Greek characters. As a result, an application that relies on ANSI code pages cannot store Greek and German in the same text stream unless it includes an identifier that indicates the referenced code page.
Double-byte character set (DBCS) encodings	Supports languages, such as Chinese, Japanese, and Korean, that contain more than 256 characters. In a DBCS, a pair of code points (a double byte) represents each character. The Encoding.IsSingleByte property returns <code>false</code> for DBCS encodings. You can retrieve an encoding object for a particular DBCS by calling the Encoding.GetEncoding(Int32) method. When an application handles DBCS data, the first byte of a DBCS character (the lead byte) is processed in combination with the trail byte that immediately follows it. Because a single pair of double-byte code points can represent different characters depending on the code page, this scheme still does not allow for the combination of two languages, such as Japanese and Chinese, in the same data stream.

These encodings enable you to work with Unicode characters as well as with encodings that are most commonly used in legacy applications. In addition, you can create a custom encoding by defining a class that derives from [Encoding](#) and overriding its members.

.NET Core encoding support

By default, .NET Core does not make available any code page encodings other than code page 28591 and the Unicode encodings, such as UTF-8 and UTF-16. However, you can add the code page encodings found in standard Windows apps that target .NET to your app. For more information, see the [CodePagesEncodingProvider](#) topic.

Selecting an Encoding Class

If you have the opportunity to choose the encoding to be used by your application, you should use a Unicode encoding, preferably either [UTF8Encoding](#) or [UnicodeEncoding](#). (.NET also supports a third Unicode encoding, [UTF32Encoding](#).)

If you are planning to use an ASCII encoding ([ASCIIEncoding](#)), choose [UTF8Encoding](#) instead. The two encodings are identical for the ASCII character set, but [UTF8Encoding](#) has the following advantages:

- It can represent every Unicode character, whereas [ASCIIEncoding](#) supports only the Unicode character values between U+0000 and U+007F.

- It provides error detection and better security.
- It has been tuned to be as fast as possible and should be faster than any other encoding. Even for content that is entirely ASCII, operations performed with [UTF8Encoding](#) are faster than operations performed with [ASCIIEncoding](#).

You should consider using [ASCIIEncoding](#) only for legacy applications. However, even for legacy applications, [UTF8Encoding](#) might be a better choice for the following reasons (assuming default settings):

- If your application has content that is not strictly ASCII and encodes it with [ASCIIEncoding](#), each non-ASCII character encodes as a question mark (?). If the application then decodes this data, the information is lost.
- If your application has content that is not strictly ASCII and encodes it with [UTF8Encoding](#), the result seems unintelligible if interpreted as ASCII. However, if the application then uses a UTF-8 decoder to decode this data, the data performs a round trip successfully.

In a web application, characters sent to the client in response to a web request should reflect the encoding used on the client. In most cases, you should set the [HttpResponse.ContentEncoding](#) property to the value returned by the [HttpRequest.ContentEncoding](#) property to display text in the encoding that the user expects.

Using an Encoding Object

An encoder converts a string of characters (most commonly, Unicode characters) to its numeric (byte) equivalent. For example, you might use an ASCII encoder to convert Unicode characters to ASCII so that they can be displayed at the console. To perform the conversion, you call the [Encoding.GetBytes](#) method. If you want to determine how many bytes are needed to store the encoded characters before performing the encoding, you can call the [GetByteCount](#) method.

The following example uses a single byte array to encode strings in two separate operations. It maintains an index that indicates the starting position in the byte array for the next set of ASCII-encoded bytes. It calls the [ASCIIEncoding.GetByteCount\(String\)](#) method to ensure that the byte array is large enough to accommodate the encoded string. It then calls the [ASCIIEncoding.GetBytes\(String, Int32, Int32, Byte\[\], Int32\)](#) method to encode the characters in the string.

```

using System;
using System.Text;

public class Example
{
    public static void Main()
    {
        string[] strings= { "This is the first sentence. ",
                           "This is the second sentence. " };
        Encoding asciiEncoding = Encoding.ASCII;

        // Create array of adequate size.
        byte[] bytes = new byte[49];
        // Create index for current position of array.
        int index = 0;

        Console.WriteLine("Strings to encode:");
        foreach (var stringValue in strings) {
            Console.WriteLine("  {0}", stringValue);

            int count = asciiEncoding.GetByteCount(stringValue);
            if (count + index >= bytes.Length)
                Array.Resize(ref bytes, bytes.Length + 50);

            int written = asciiEncoding.GetBytes(stringValue, 0,
                                                 stringValue.Length,
                                                 bytes, index);

            index = index + written;
        }
        Console.WriteLine("\nEncoded bytes:");
        Console.WriteLine("{0}", ShowByteValues(bytes, index));
        Console.WriteLine();

        // Decode Unicode byte array to a string.
        string newString = asciiEncoding.GetString(bytes, 0, index);
        Console.WriteLine("Decoded: {0}", newString);
    }

    private static string ShowByteValues(byte[] bytes, int last )
    {
        string returnString = "  ";
        for (int ctr = 0; ctr <= last - 1; ctr++) {
            if (ctr % 20 == 0)
                returnString += "\n  ";
            returnString += String.Format("{0:X2} ", bytes[ctr]);
        }
        return returnString;
    }
}

// The example displays the following output:
//      Strings to encode:
//          This is the first sentence.
//          This is the second sentence.
//
//      Encoded bytes:
//
//          54 68 69 73 20 69 73 20 74 68 65 20 66 69 72 73 74 20 73 65
//          6E 74 65 6E 63 65 2E 20 54 68 69 73 20 69 73 20 74 68 65 20
//          73 65 63 6F 6E 64 20 73 65 6E 74 65 6E 63 65 2E 20
//
//      Decoded: This is the first sentence. This is the second sentence.

```

```

Imports System.Text

Module Example
    Public Sub Main()
        Dim strings() As String = {"This is the first sentence. ",
                                  "This is the second sentence. "}
        Dim asciiEncoding As Encoding = Encoding.ASCII

        ' Create array of adequate size.
        Dim bytes(50) As Byte
        ' Create index for current position of array.
        Dim index As Integer = 0

        Console.WriteLine("Strings to encode:")
        For Each stringValue In strings
            Console.WriteLine("    {0}", stringValue)

            Dim count As Integer = asciiEncoding.GetByteCount(stringValue)
            If count + index >= bytes.Length Then
                Array.Resize(bytes, bytes.Length + 50)
            End If
            Dim written As Integer = asciiEncoding.GetBytes(stringValue, 0,
                                                          stringValue.Length,
                                                          bytes, index)

            index = index + written
        Next
        Console.WriteLine()
        Console.WriteLine("Encoded bytes:")
        Console.WriteLine("{0}", ShowByteValues(bytes, index))
        Console.WriteLine()

        ' Decode Unicode byte array to a string.
        Dim newString As String = asciiEncoding.GetString(bytes, 0, index)
        Console.WriteLine("Decoded: {0}", newString)
    End Sub

    Private Function ShowByteValues(bytes As Byte(), last As Integer) As String
        Dim returnString As String = ""
        For ctr As Integer = 0 To last - 1
            If ctr Mod 20 = 0 Then returnString += vbCrLf + " "
            returnString += String.Format("{0:X2} ", bytes(ctr))
        Next
        Return returnString
    End Function
End Module
' The example displays the following output:
'     Strings to encode:
'         This is the first sentence.
'         This is the second sentence.
'
'     Encoded bytes:
'
'         54 68 69 73 20 69 73 20 74 68 65 20 66 69 72 73 74 20 73 65
'         6E 74 65 6E 63 65 2E 20 54 68 69 73 20 69 73 20 74 68 65 20
'         73 65 63 6F 6E 64 20 73 65 6E 74 65 6E 63 65 2E 20
'
'     Decoded: This is the first sentence. This is the second sentence.

```

A decoder converts a byte array that reflects a particular character encoding into a set of characters, either in a character array or in a string. To decode a byte array into a character array, you call the [Encoding.GetChars](#) method. To decode a byte array into a string, you call the [GetString](#) method. If you want to determine how many characters are needed to store the decoded bytes before performing the decoding, you can call the [GetCharCount](#) method.

The following example encodes three strings and then decodes them into a single array of characters. It maintains an index that indicates the starting position in the character array for the next set of decoded characters. It calls the [GetCharCount](#) method to ensure that the character array is large enough to accommodate all the decoded characters. It then calls the [ASCIIEncoding.GetChars\(Byte\[\], Int32, Int32, Char\[\], Int32\)](#) method to decode the byte array.

```

using System;
using System.Text;

public class Example
{
    public static void Main()
    {
        string[] strings = { "This is the first sentence. ",
                            "This is the second sentence. ",
                            "This is the third sentence. " };
        Encoding asciiEncoding = Encoding.ASCII;
        // Array to hold encoded bytes.
        byte[] bytes;
        // Array to hold decoded characters.
        char[] chars = new char[50];
        // Create index for current position of character array.
        int index = 0;

        foreach (var stringValue in strings) {
            Console.WriteLine("String to Encode: {0}", stringValue);
            // Encode the string to a byte array.
            bytes = asciiEncoding.GetBytes(stringValue);
            // Display the encoded bytes.
            Console.Write("Encoded bytes: ");
            for (int ctr = 0; ctr < bytes.Length; ctr++)
                Console.Write("{0}{1:X2}",
                             ctr % 20 == 0 ? Environment.NewLine : "",
                             bytes[ctr]);
            Console.WriteLine();

            // Decode the bytes to a single character array.
            int count = asciiEncoding.GetCharCount(bytes);
            if (count + index >= chars.Length)
                Array.Resize(ref chars, chars.Length + 50);

            int written = asciiEncoding.GetChars(bytes, 0,
                                                 bytes.Length,
                                                 chars, index);
            index = index + written;
            Console.WriteLine();
        }

        // Instantiate a single string containing the characters.
        string decodedString = new string(chars, 0, index - 1);
        Console.WriteLine("Decoded string: ");
        Console.WriteLine(decodedString);
    }
}

// The example displays the following output:
// String to Encode: This is the first sentence.
// Encoded bytes:
// 54 68 69 73 20 69 73 20 74 68 65 20 66 69 72 73 74 20 73 65
// 6E 74 65 6E 63 65 2E 20
//
// String to Encode: This is the second sentence.
// Encoded bytes:
// 54 68 69 73 20 69 73 20 74 68 65 20 73 65 63 6F 6E 64 20 73
// 65 6E 74 65 6E 63 65 2E 20
//
// String to Encode: This is the third sentence.
// Encoded bytes:
// 54 68 69 73 20 69 73 20 74 68 65 20 74 68 69 72 64 20 73 65
// 6E 74 65 6E 63 65 2E 20
//
// Decoded string:
// This is the first sentence. This is the second sentence. This is the third sentence.

```

```

Imports System.Text

Module Example
    Public Sub Main()
        Dim strings() As String = {"This is the first sentence. ",
                                  "This is the second sentence. ",
                                  "This is the third sentence. "}
        Dim asciiEncoding As Encoding = Encoding.ASCII
        ' Array to hold encoded bytes.
        Dim bytes() As Byte
        ' Array to hold decoded characters.
        Dim chars(50) As Char
        ' Create index for current position of character array.
        Dim index As Integer

        For Each stringValue In strings
            Console.WriteLine("String to Encode: {0}", stringValue)
            ' Encode the string to a byte array.
            bytes = asciiEncoding.GetBytes(stringValue)
            ' Display the encoded bytes.
            Console.Write("Encoded bytes: ")
            For ctr As Integer = 0 To bytes.Length - 1
                Console.Write(" {0}{1:X2}", If(ctr Mod 20 = 0, vbCrLf, ""), bytes(ctr))
            Next
            Console.WriteLine()

            ' Decode the bytes to a single character array.
            Dim count As Integer = asciiEncoding.GetCharCount(bytes)
            If count + index >= chars.Length Then
                Array.Resize(chars, chars.Length + 50)
            End If
            Dim written As Integer = asciiEncoding.GetChars(bytes, 0,
                                                          bytes.Length,
                                                          chars, index)
            index = index + written
            Console.WriteLine()
        Next

        ' Instantiate a single string containing the characters.
        Dim decodedString As New String(chars, 0, index - 1)
        Console.WriteLine("Decoded string: ")
        Console.WriteLine(decodedString)
    End Sub
End Module
' The example displays the following output:
'   String to Encode: This is the first sentence.
'   Encoded bytes:
'   54 68 69 73 20 69 73 20 74 68 65 20 66 69 72 73 74 20 73 65
'   6E 74 65 6E 63 65 2E 20
'
'   String to Encode: This is the second sentence.
'   Encoded bytes:
'   54 68 69 73 20 69 73 20 74 68 65 20 73 65 63 6F 6E 64 20 73
'   65 6E 74 65 6E 63 65 2E 20
'
'   String to Encode: This is the third sentence.
'   Encoded bytes:
'   54 68 69 73 20 69 73 20 74 68 65 20 74 68 69 72 64 20 73 65
'   6E 74 65 6E 63 65 2E 20
'
'   Decoded string:
'   This is the first sentence. This is the second sentence. This is the third sentence.

```

The encoding and decoding methods of a class derived from [Encoding](#) are designed to work on a complete set of data; that is, all the data to be encoded or decoded is supplied in a single method call. However, in some cases,

data is available in a stream, and the data to be encoded or decoded may be available only from separate read operations. This requires the encoding or decoding operation to remember any saved state from its previous invocation. Methods of classes derived from [Encoder](#) and [Decoder](#) are able to handle encoding and decoding operations that span multiple method calls.

An [Encoder](#) object for a particular encoding is available from that encoding's [Encoding.GetEncoder](#) property. A [Decoder](#) object for a particular encoding is available from that encoding's [Encoding.GetDecoder](#) property. For decoding operations, note that classes derived from [Decoder](#) include a [Decoder.GetChars](#) method, but they do not have a method that corresponds to [Encoding.GetString](#).

The following example illustrates the difference between using the [Encoding.GetString](#) and [Decoder.GetChars](#) methods for decoding a Unicode byte array. The example encodes a string that contains some Unicode characters to a file, and then uses the two decoding methods to decode them ten bytes at a time. Because a surrogate pair occurs in the tenth and eleventh bytes, it is decoded in separate method calls. As the output shows, the [Encoding.GetString](#) method is not able to correctly decode the bytes and instead replaces them with U+FFFD (REPLACEMENT CHARACTER). On the other hand, the [Decoder.GetChars](#) method is able to successfully decode the byte array to get the original string.

```
using System;
using System.IO;
using System.Text;

public class Example
{
    public static void Main()
    {
        // Use default replacement fallback for invalid encoding.
        UnicodeEncoding enc = new UnicodeEncoding(true, false, false);

        // Define a string with various Unicode characters.
        string str1 = "AB YZ 19 \uD800\udc05 \u00e4";
        str1 += "Unicode characters. \u00a9 \u010C s \u0062\u0308";
        Console.WriteLine("Created original string...\n");

        // Convert string to byte array.
        byte[] bytes = enc.GetBytes(str1);

        FileStream fs = File.Create(@".\characters.bin");
        BinaryWriter bw = new BinaryWriter(fs);
        bw.Write(bytes);
        bw.Close();

        // Read bytes from file.
        FileStream fsIn = File.OpenRead(@".\characters.bin");
        BinaryReader br = new BinaryReader(fsIn);

        const int count = 10;           // Number of bytes to read at a time.
        byte[] bytesRead = new byte[10]; // Buffer (byte array).
        int read;                     // Number of bytes actually read.
        string str2 = String.Empty;    // Decoded string.

        // Try using Encoding object for all operations.
        do {
            read = br.Read(bytesRead, 0, count);
            str2 += enc.GetString(bytesRead, 0, read);
        } while (read == count);
        br.Close();
        Console.WriteLine("Decoded string using UnicodeEncoding.GetString()...");
        CompareForEquality(str1, str2);
        Console.WriteLine();

        // Use Decoder for all operations.
        fsIn = File.OpenRead(@".\characters.bin");
        br = new BinaryReader(fsIn);
```

```

Decoder decoder = enc.GetDecoder();
char[] chars = new char[50];
int index = 0;                                // Next character to write in array.
int written = 0;                               // Number of chars written to array.
do {
    read = br.Read(bytesRead, 0, count);
    if (index + decoder.GetCharCount(bytesRead, 0, read) - 1 >= chars.Length)
        Array.Resize(ref chars, chars.Length + 50);

    written = decoder.GetChars(bytesRead, 0, read, chars, index);
    index += written;
} while (read == count);
br.Close();
// Instantiate a string with the decoded characters.
string str3 = new String(chars, 0, index);
Console.WriteLine("Decoded string using UnicodeEncoding.Decoder.GetString()...");
CompareForEquality(str1, str3);
}

private static void CompareForEquality(string original, string decoded)
{
    bool result = original.Equals(decoded);
    Console.WriteLine("original = decoded: {0}",
                      original.Equals(decoded, StringComparison.OrdinalIgnoreCase));
    if (!result) {
        Console.WriteLine("Code points in original string:");
        foreach (var ch in original)
            Console.Write("{0} ", Convert.ToInt16(ch).ToString("X4"));
        Console.WriteLine();

        Console.WriteLine("Code points in decoded string:");
        foreach (var ch in decoded)
            Console.Write("{0} ", Convert.ToInt16(ch).ToString("X4"));
        Console.WriteLine();
    }
}
}

// The example displays the following output:
// Created original string...
//
// Decoded string using UnicodeEncoding.GetString()...
// original = decoded: False
// Code points in original string:
// 0041 0042 0020 0059 005A 0020 0031 0039 0020 D800 DC05 0020 00E4 0055 006E 0069 0063 006F
// 0064 0065 0020 0063 0068 0061 0072 0061 0063 0074 0065 0072 0073 002E 0020 00A9 0020 010C
// 0020 0073 0020 0062 0308
// Code points in decoded string:
// 0041 0042 0020 0059 005A 0020 0031 0039 0020 FFFD 0020 00E4 0055 006E 0069 0063 006F
// 0064 0065 0020 0063 0068 0061 0072 0061 0063 0074 0065 0072 0073 002E 0020 00A9 0020 010C
// 0020 0073 0020 0062 0308
//
// Decoded string using UnicodeEncoding.Decoder.GetString()...
// original = decoded: True

```

```

Imports System.IO
Imports System.Text

Module Example
    Public Sub Main()
        ' Use default replacement fallback for invalid encoding.
        Dim enc As New UnicodeEncoding(True, False, False)

        ' Define a string with various Unicode characters.
        Dim str1 As String = String.Format("AB YZ 19 {0}{1} {2}",
                                           ChrW(&hD800), ChrW(&hDC05), ChrW(&h00e4))
        str1 += String.Format("Unicode characters. {0} {1} s {2}{3}",
                             ChrW(&h00a9), ChrW(&h10C), ChrW(&h0062), ChrW(&h0308))
    End Sub
End Module

```

```

Console.WriteLine("Created original string...")
Console.WriteLine()

' Convert string to byte array.
Dim bytes() As Byte = enc.GetBytes(str1)

Dim fs As FileStream = File.Create(".\characters.bin")
Dim bw As New BinaryWriter(fs)
bw.Write(bytes)
bw.Close()

' Read bytes from file.
Dim fsIn As FileStream = File.OpenRead(".\characters.bin")
Dim br As New BinaryReader(fsIn)

Const count As Integer = 10      ' Number of bytes to read at a time.
Dim bytesRead(9) As Byte        ' Buffer (byte array).
Dim read As Integer            ' Number of bytes actually read.
Dim str2 As String = ""         ' Decoded string.

' Try using Encoding object for all operations.
Do
    read = br.Read(bytesRead, 0, count)
    str2 += enc.GetString(bytesRead, 0, read)
Loop While read = count
br.Close()
Console.WriteLine("Decoded string using UnicodeEncoding.GetString()...")
CompareForEquality(str1, str2)
Console.WriteLine()

' Use Decoder for all operations.
fsIn = File.OpenRead(".\characters.bin")
br = New BinaryReader(fsIn)
Dim decoder As Decoder = enc.GetDecoder()
Dim chars(50) As Char
Dim index As Integer = 0          ' Next character to write in array.
Dim written As Integer = 0        ' Number of chars written to array.
Do
    read = br.Read(bytesRead, 0, count)
    If index + decoder.GetCharCount(bytesRead, 0, read) - 1 >= chars.Length Then
        Array.Resize(chars, chars.Length + 50)
    End If
    written = decoder.GetChars(bytesRead, 0, read, chars, index)
    index += written
Loop While read = count
br.Close()
' Instantiate a string with the decoded characters.
Dim str3 As New String(chars, 0, index)
Console.WriteLine("Decoded string using UnicodeEncoding.Decoder.GetString()...")
CompareForEquality(str1, str3)
End Sub

Private Sub CompareForEquality(original As String, decoded As String)
    Dim result As Boolean = original.Equals(decoded)
    Console.WriteLine("original = decoded: {0}",
                      original.Equals(decoded, StringComparison.OrdinalIgnoreCase))
    If Not result Then
        Console.WriteLine("Code points in original string:")
        For Each ch In original
            Console.Write("{0} ", Convert.ToInt16(ch).ToString("X4"))
        Next
        Console.WriteLine()
        Console.WriteLine("Code points in decoded string:")
        For Each ch In decoded
            Console.Write("{0} ", Convert.ToInt16(ch).ToString("X4"))
        Next
        Console.WriteLine()
    End If

```

```

    End Sub
End Module
' The example displays the following output:
'   Created original string...
'
'   Decoded string using UnicodeEncoding.GetString()...
'   original = decoded: False
'   Code points in original string:
'   0041 0042 0020 0059 005A 0020 0031 0039 0020 D800 DC05 0020 00E4 0055 006E 0069 0063 006F
'   0064 0065 0020 0063 0068 0061 0072 0061 0063 0074 0065 0072 0073 002E 0020 00A9 0020 010C
'   0020 0073 0020 0062 0308
'   Code points in decoded string:
'   0041 0042 0020 0059 005A 0020 0031 0039 0020 FFFD FFFD 0020 00E4 0055 006E 0069 0063 006F
'   0064 0065 0020 0063 0068 0061 0072 0061 0063 0074 0065 0072 0073 002E 0020 00A9 0020 010C
'   0020 0073 0020 0062 0308
'
'   Decoded string using UnicodeEncoding.Decoder.GetString()...
'   original = decoded: True

```

Choosing a Fallback Strategy

When a method tries to encode or decode a character but no mapping exists, it must implement a fallback strategy that determines how the failed mapping should be handled. There are three types of fallback strategies:

- Best-fit fallback
- Replacement fallback
- Exception fallback

IMPORTANT

The most common problems in encoding operations occur when a Unicode character cannot be mapped to a particular code page encoding. The most common problems in decoding operations occur when invalid byte sequences cannot be translated into valid Unicode characters. For these reasons, you should know which fallback strategy a particular encoding object uses. Whenever possible, you should specify the fallback strategy used by an encoding object when you instantiate the object.

Best-Fit Fallback

When a character does not have an exact match in the target encoding, the encoder can try to map it to a similar character. (Best-fit fallback is mostly an encoding rather than a decoding issue. There are very few code pages that contain characters that cannot be successfully mapped to Unicode.) Best-fit fallback is the default for code page and double-byte character set encodings that are retrieved by the [Encoding.GetEncoding\(Int32\)](#) and [Encoding.GetEncoding\(String\)](#) overloads.

NOTE

In theory, the Unicode encoding classes provided in .NET ([UTF8Encoding](#), [UnicodeEncoding](#), and [UTF32Encoding](#)) support every character in every character set, so they can be used to eliminate best-fit fallback issues.

Best-fit strategies vary for different code pages. For example, for some code pages, full-width Latin characters map to the more common half-width Latin characters. For other code pages, this mapping is not made. Even under an aggressive best-fit strategy, there is no imaginable fit for some characters in some encodings. For example, a Chinese ideograph has no reasonable mapping to code page 1252. In this case, a replacement string is used. By default, this string is just a single QUESTION MARK (U+003F).

NOTE

Best-fit strategies are not documented in detail. However, several code pages are documented at the [Unicode Consortium's](#) website. Please review the `readme.txt` file in that folder for a description of how to interpret the mapping files.

The following example uses code page 1252 (the Windows code page for Western European languages) to illustrate best-fit mapping and its drawbacks. The `Encoding.GetEncoding(Int32)` method is used to retrieve an encoding object for code page 1252. By default, it uses a best-fit mapping for Unicode characters that it does not support. The example instantiates a string that contains three non-ASCII characters - CIRCLED LATIN CAPITAL LETTER S (U+24C8), SUPERSCRIPT FIVE (U+2075), and INFINITY (U+221E) - separated by spaces. As the output from the example shows, when the string is encoded, the three original non-space characters are replaced by QUESTION MARK (U+003F), DIGIT FIVE (U+0035), and DIGIT EIGHT (U+0038). DIGIT EIGHT is a particularly poor replacement for the unsupported INFINITY character, and QUESTION MARK indicates that no mapping was available for the original character.

```
using System;
using System.Text;

public class Example
{
    public static void Main()
    {
        // Get an encoding for code page 1252 (Western Europe character set).
        Encoding cp1252 = Encoding.GetEncoding(1252);

        // Define and display a string.
        string str = "\u24c8 \u2075 \u221e";
        Console.WriteLine("Original string: " + str);
        Console.Write("Code points in string: ");
        foreach (var ch in str)
            Console.Write("{0} ", Convert.ToInt16(ch).ToString("X4"));

        Console.WriteLine("\n");

        // Encode a Unicode string.
        Byte[] bytes = cp1252.GetBytes(str);
        Console.Write("Encoded bytes: ");
        foreach (byte byt in bytes)
            Console.Write("{0:X2} ", byt);
        Console.WriteLine("\n");

        // Decode the string.
        string str2 = cp1252.GetString(bytes);
        Console.WriteLine("String round-tripped: {0}", str.Equals(str2));
        if (!str.Equals(str2)) {
            Console.WriteLine(str2);
            foreach (var ch in str2)
                Console.Write("{0} ", Convert.ToInt16(ch).ToString("X4"));
        }
    }
}

// The example displays the following output:
//      Original string: ® ⁵ ∞
//      Code points in string: 24C8 0020 2075 0020 221E
//
//      Encoded bytes: 3F 20 35 20 38
//
//      String round-tripped: False
//      ? 5 8
//      003F 0020 0035 0020 0038
```

```

Imports System.Text

Module Example
    Public Sub Main()
        ' Get an encoding for code page 1252 (Western Europe character set).
        Dim cp1252 As Encoding = Encoding.GetEncoding(1252)

        ' Define and display a string.
        Dim str As String = String.Format("{0} {1} {2}", ChrW(&h24c8), ChrW(&H2075), ChrW(&h221E))
        Console.WriteLine("Original string: " + str)
        Console.WriteLine("Code points in string: ")
        For Each ch In str
            Console.Write("{0} ", Convert.ToInt16(ch).ToString("X4"))
        Next
        Console.WriteLine()
        Console.WriteLine()

        ' Encode a Unicode string.
        Dim bytes() As Byte = cp1252.GetBytes(str)
        Console.WriteLine("Encoded bytes: ")
        For Each byt In bytes
            Console.Write("{0:X2} ", byt)
        Next
        Console.WriteLine()
        Console.WriteLine()

        ' Decode the string.
        Dim str2 As String = cp1252.GetString(bytes)
        Console.WriteLine("String round-tripped: {0}", str.Equals(str2))
        If Not str.Equals(str2) Then
            Console.WriteLine(str2)
            For Each ch In str2
                Console.Write("{0} ", Convert.ToInt16(ch).ToString("X4"))
            Next
        End If
    End Sub
End Module
' The example displays the following output:
' Original string: ® § °
' Code points in string: 24C8 0020 2075 0020 221E
'
' Encoded bytes: 3F 20 35 20 38
'
' String round-tripped: False
' ? 5 8
' 003F 0020 0035 0020 0038

```

Best-fit mapping is the default behavior for an [Encoding](#) object that encodes Unicode data into code page data, and there are legacy applications that rely on this behavior. However, most new applications should avoid best-fit behavior for security reasons. For example, applications should not put a domain name through a best-fit encoding.

NOTE

You can also implement a custom best-fit fallback mapping for an encoding. For more information, see the [Implementing a Custom Fallback Strategy](#) section.

If best-fit fallback is the default for an encoding object, you can choose another fallback strategy when you retrieve an [Encoding](#) object by calling the [Encoding.GetEncoding\(Int32, EncoderFallback, DecoderFallback\)](#) or [Encoding.GetEncoding\(String, EncoderFallback, DecoderFallback\)](#) overload. The following section includes an example that replaces each character that cannot be mapped to code page 1252 with an asterisk (*).

```
using System;
using System.Text;

public class Example
{
    public static void Main()
    {
        Encoding cp1252r = Encoding.GetEncoding(1252,
            new EncoderReplacementFallback("*"),
            new DecoderReplacementFallback("*"));

        string str1 = "\u24C8 \u2075 \u221E";
        Console.WriteLine(str1);
        foreach (var ch in str1)
            Console.Write("{0} ", Convert.ToInt16(ch).ToString("X4"));

        Console.WriteLine();

        byte[] bytes = cp1252r.GetBytes(str1);
        string str2 = cp1252r.GetString(bytes);
        Console.WriteLine("Round-trip: {0}", str1.Equals(str2));
        if (!str1.Equals(str2)) {
            Console.WriteLine(str2);
            foreach (var ch in str2)
                Console.Write("{0} ", Convert.ToInt16(ch).ToString("X4"));

            Console.WriteLine();
        }
    }
}

// The example displays the following output:
//      ⓧ ∞
//      24C8 0020 2075 0020 221E
//      Round-trip: False
//      * * *
//      002A 0020 002A 0020 002A
```

```

Imports System.Text

Module Example
    Public Sub Main()
        Dim cp1252r As Encoding = Encoding.GetEncoding(1252,
            New EncoderReplacementFallback(""),
            New DecoderReplacementFallback(""))
        
        Dim str1 As String = String.Format("{0} {1} {2}", ChrW(&h24C8), ChrW(&h2075), ChrW(&h221E))
        Console.WriteLine(str1)
        For Each ch In str1
            Console.Write("{0} ", Convert.ToInt16(ch).ToString("X4"))
        Next
        Console.WriteLine()

        Dim bytes() As Byte = cp1252r.GetBytes(str1)
        Dim str2 As String = cp1252r.GetString(bytes)
        Console.WriteLine("Round-trip: {0}", str1.Equals(str2))
        If Not str1.Equals(str2) Then
            Console.WriteLine(str2)
            For Each ch In str2
                Console.Write("{0} ", Convert.ToInt16(ch).ToString("X4"))
            Next
            Console.WriteLine()
        End If
    End Sub
End Module
' The example displays the following output:
'   Ⓜ ȝ ~
'   24C8 0020 2075 0020 221E
'   Round-trip: False
'   * * *
'   002A 0020 002A 0020 002A

```

Replacement Fallback

When a character does not have an exact match in the target scheme, but there is no appropriate character that it can be mapped to, the application can specify a replacement character or string. This is the default behavior for the Unicode decoder, which replaces any two-byte sequence that it cannot decode with REPLACEMENT_CHARACTER (U+FFFD). It is also the default behavior of the [ASCIIEncoding](#) class, which replaces each character that it cannot encode or decode with a question mark. The following example illustrates character replacement for the Unicode string from the previous example. As the output shows, each character that cannot be decoded into an ASCII byte value is replaced by 0x3F, which is the ASCII code for a question mark.

```
using System;
using System.Text;

public class Example
{
    public static void Main()
    {
        Encoding enc = Encoding.ASCII;

        string str1 = "\u24C8 \u2075 \u221E";
        Console.WriteLine(str1);
        foreach (var ch in str1)
            Console.Write("{0} ", Convert.ToInt16(ch).ToString("X4"));

        Console.WriteLine("\n");

        // Encode the original string using the ASCII encoder.
        byte[] bytes = enc.GetBytes(str1);
        Console.Write("Encoded bytes: ");
        foreach (var byt in bytes)
            Console.Write("{0:X2} ", byt);
        Console.WriteLine("\n");

        // Decode the ASCII bytes.
        string str2 = enc.GetString(bytes);
        Console.WriteLine("Round-trip: {0}", str1.Equals(str2));
        if (!str1.Equals(str2)) {
            Console.WriteLine(str2);
            foreach (var ch in str2)
                Console.Write("{0} ", Convert.ToInt16(ch).ToString("X4"));

            Console.WriteLine();
        }
    }
}

// The example displays the following output:
//      ⓧ ጀ ߱
//      24C8 0020 2075 0020 221E
//
//      Encoded bytes: 3F 20 3F 20 3F
//
//      Round-trip: False
//      ? ? ?
//      003F 0020 003F 0020 003F
```

```

Imports System.Text

Module Example
    Public Sub Main()
        Dim enc As Encoding = Encoding.Ascii

        Dim str1 As String = String.Format("{0} {1} {2}", ChrW(&h24C8), ChrW(&h2075), ChrW(&h221E))
        Console.WriteLine(str1)
        For Each ch In str1
            Console.Write("{0} ", Convert.ToInt16(ch).ToString("X4"))
        Next
        Console.WriteLine()
        Console.WriteLine()

        ' Encode the original string using the ASCII encoder.
        Dim bytes() As Byte = enc.GetBytes(str1)
        Console.Write("Encoded bytes: ")
        For Each byt In bytes
            Console.WriteLine("{0:X2} ", byt)
        Next
        Console.WriteLine()
        Console.WriteLine()

        ' Decode the ASCII bytes.
        Dim str2 As String = enc.GetString(bytes)
        Console.WriteLine("Round-trip: {0}", str1.Equals(str2))
        If Not str1.Equals(str2) Then
            Console.WriteLine(str2)
            For Each ch In str2
                Console.Write("{0} ", Convert.ToInt16(ch).ToString("X4"))
            Next
            Console.WriteLine()
        End If
    End Sub
End Module
' The example displays the following output:
'     Ⓜ ₛ Ꮰ
'     24C8 0020 2075 0020 221E
'
'     Encoded bytes: 3F 20 3F 20 3F
'
'     Round-trip: False
'     ? ? ?
'     003F 0020 003F 0020 003F

```

.NET includes the [EncoderReplacementFallback](#) and [DecoderReplacementFallback](#) classes, which substitute a replacement string if a character does not map exactly in an encoding or decoding operation. By default, this replacement string is a question mark, but you can call a class constructor overload to choose a different string. Typically, the replacement string is a single character, although this is not a requirement. The following example changes the behavior of the code page 1252 encoder by instantiating an [EncoderReplacementFallback](#) object that uses an asterisk (*) as a replacement string.

```
using System;
using System.Text;

public class Example
{
    public static void Main()
    {
        Encoding cp1252r = Encoding.GetEncoding(1252,
            new EncoderReplacementFallback("*"),
            new DecoderReplacementFallback("*"));

        string str1 = "\u24C8 \u2075 \u221E";
        Console.WriteLine(str1);
        foreach (var ch in str1)
            Console.Write("{0} ", Convert.ToInt16(ch).ToString("X4"));

        Console.WriteLine();

        byte[] bytes = cp1252r.GetBytes(str1);
        string str2 = cp1252r.GetString(bytes);
        Console.WriteLine("Round-trip: {0}", str1.Equals(str2));
        if (!str1.Equals(str2)) {
            Console.WriteLine(str2);
            foreach (var ch in str2)
                Console.Write("{0} ", Convert.ToInt16(ch).ToString("X4"));

            Console.WriteLine();
        }
    }
}

// The example displays the following output:
//      ⓧ ∞
//      24C8 0020 2075 0020 221E
//      Round-trip: False
//      * * *
//      002A 0020 002A 0020 002A
```

```

Imports System.Text

Module Example
    Public Sub Main()
        Dim cp1252r As Encoding = Encoding.GetEncoding(1252,
            New EncoderReplacementFallback(""),
            New DecoderReplacementFallback(""))
        
        Dim str1 As String = String.Format("{0} {1} {2}", ChrW(&h24C8), ChrW(&h2075), ChrW(&h221E))
        Console.WriteLine(str1)
        For Each ch In str1
            Console.Write("{0} ", Convert.ToInt16(ch).ToString("X4"))
        Next
        Console.WriteLine()

        Dim bytes() As Byte = cp1252r.GetBytes(str1)
        Dim str2 As String = cp1252r.GetString(bytes)
        Console.WriteLine("Round-trip: {0}", str1.Equals(str2))
        If Not str1.Equals(str2) Then
            Console.WriteLine(str2)
            For Each ch In str2
                Console.Write("{0} ", Convert.ToInt16(ch).ToString("X4"))
            Next
            Console.WriteLine()
        End If
    End Sub
End Module
' The example displays the following output:
'   Ⓜ ȝ ø
'   24C8 0020 2075 0020 221E
'   Round-trip: False
'   * * *
'   002A 0020 002A 0020 002A

```

NOTE

You can also implement a replacement class for an encoding. For more information, see the [Implementing a Custom Fallback Strategy](#) section.

In addition to QUESTION MARK (U+003F), the Unicode REPLACEMENT CHARACTER (U+FFFD) is commonly used as a replacement string, particularly when decoding byte sequences that cannot be successfully translated into Unicode characters. However, you are free to choose any replacement string, and it can contain multiple characters.

Exception Fallback

Instead of providing a best-fit fallback or a replacement string, an encoder can throw an [EncoderFallbackException](#) if it is unable to encode a set of characters, and a decoder can throw a [DecoderFallbackException](#) if it is unable to decode a byte array. To throw an exception in encoding and decoding operations, you supply an [EncoderExceptionFallback](#) object and a [DecoderExceptionFallback](#) object, respectively, to the [Encoding.GetEncoding\(String, EncoderFallback, DecoderFallback\)](#) method. The following example illustrates exception fallback with the [ASCIIEncoding](#) class.

```

using System;
using System.Text;

public class Example
{
    public static void Main()
    {
        Encoding enc = Encoding.GetEncoding("us-ascii",

```

```

        new EncoderExceptionFallback(),
        new DecoderExceptionFallback());

string str1 = "\u24C8 \u2075 \u221E";
Console.WriteLine(str1);
foreach (var ch in str1)
    Console.Write("{0} ", Convert.ToInt16(ch).ToString("X4"));

Console.WriteLine("\n");

// Encode the original string using the ASCII encoder.
byte[] bytes = {};
try {
    bytes = enc.GetBytes(str1);
    Console.Write("Encoded bytes: ");
    foreach (var byt in bytes)
        Console.WriteLine("{0:X2} ", byt);

    Console.WriteLine();
}
catch (EncoderFallbackException e) {
    Console.Write("Exception: ");
    if (e.IsUnknownSurrogate())
        Console.WriteLine("Unable to encode surrogate pair 0x{0:X4} 0x{1:X3} at index {2}.",
                        Convert.ToInt16(e.CharUnknownHigh),
                        Convert.ToInt16(e.CharUnknownLow),
                        e.Index);
    else
        Console.WriteLine("Unable to encode 0x{0:X4} at index {1}.",
                        Convert.ToInt16(e.CharUnknown),
                        e.Index);
    return;
}
Console.WriteLine();

// Decode the ASCII bytes.
try {
    string str2 = enc.GetString(bytes);
    Console.WriteLine("Round-trip: {0}", str1.Equals(str2));
    if (!str1.Equals(str2)) {
        Console.WriteLine(str2);
        foreach (var ch in str2)
            Console.Write("{0} ", Convert.ToInt16(ch).ToString("X4"));

        Console.WriteLine();
    }
}
catch (DecoderFallbackException e) {
    Console.Write("Unable to decode byte(s) ");
    foreach (byte unknown in e.BytesUnknown)
        Console.WriteLine("0x{0:X2} ");

    Console.WriteLine("at index {0}", e.Index);
}
}

// The example displays the following output:
//      Ⓜ ጀ ߻
//      24C8 0020 2075 0020 221E
//
//      Exception: Unable to encode 0x24C8 at index 0.

```

```

Imports System.Text

Module Example
    Public Sub Main()
        Dim enc As Encoding = Encoding.GetEncoding("us-ascii",
            New EncoderExceptionFallback(),
            New DecoderExceptionFallback())

        Dim str1 As String = String.Format("{0} {1} {2}", ChrW(&h24C8), ChrW(&h2075), ChrW(&h221E))
        Console.WriteLine(str1)
        For Each ch In str1
            Console.Write("{0} ", Convert.ToInt16(ch).ToString("X4"))
        Next
        Console.WriteLine()
        Console.WriteLine()

        ' Encode the original string using the ASCII encoder.
        Dim bytes() As Byte = {}
        Try
            bytes = enc.GetBytes(str1)
            Console.Write("Encoded bytes: ")
            For Each byt In bytes
                Console.WriteLine("{0:X2} ", byt)
            Next
            Console.WriteLine()
        Catch e As EncoderFallbackException
            Console.WriteLine("Exception: ")
            If e.IsUnknownSurrogate() Then
                Console.WriteLine("Unable to encode surrogate pair 0x{0:X4} 0x{1:X3} at index {2}.",
                    Convert.ToInt16(e.CharUnknownHigh),
                    Convert.ToInt16(e.CharUnknownLow),
                    e.Index)
            Else
                Console.WriteLine("Unable to encode 0x{0:X4} at index {1}.",
                    Convert.ToInt16(e.CharUnknown),
                    e.Index)
            End If
            Exit Sub
        End Try
        Console.WriteLine()

        ' Decode the ASCII bytes.
        Try
            Dim str2 As String = enc.GetString(bytes)
            Console.WriteLine("Round-trip: {0}", str1.Equals(str2))
            If Not str1.Equals(str2) Then
                Console.WriteLine(str2)
                For Each ch In str2
                    Console.Write("{0} ", Convert.ToInt16(ch).ToString("X4"))
                Next
                Console.WriteLine()
            End If
        Catch e As DecoderFallbackException
            Console.WriteLine("Unable to decode byte(s) ")
            For Each unknown As Byte In e.BytesUnknown
                Console.WriteLine("0x{0:X2} ")
            Next
            Console.WriteLine("at index {0}", e.Index)
        End Try
    End Sub
End Module
' The example displays the following output:
'   Ⓜ ߵ ߲
'   24C8 0020 2075 0020 221E
'
'   Exception: Unable to encode 0x24C8 at index 0.

```

NOTE

You can also implement a custom exception handler for an encoding operation. For more information, see the [Implementing a Custom Fallback Strategy](#) section.

The [EncoderFallbackException](#) and [DecoderFallbackException](#) objects provide the following information about the condition that caused the exception:

- The [EncoderFallbackException](#) object includes an [IsUnknownSurrogate](#) method, which indicates whether the character or characters that cannot be encoded represent an unknown surrogate pair (in which case, the method returns `true`) or an unknown single character (in which case, the method returns `false`). The characters in the surrogate pair are available from the [EncoderFallbackException.CharUnknownHigh](#) and [EncoderFallbackException.CharUnknownLow](#) properties. The unknown single character is available from the [EncoderFallbackException.CharUnknown](#) property. The [EncoderFallbackException.Index](#) property indicates the position in the string at which the first character that could not be encoded was found.
- The [DecoderFallbackException](#) object includes a [BytesUnknown](#) property that returns an array of bytes that cannot be decoded. The [DecoderFallbackException.Index](#) property indicates the starting position of the unknown bytes.

Although the [EncoderFallbackException](#) and [DecoderFallbackException](#) objects provide adequate diagnostic information about the exception, they do not provide access to the encoding or decoding buffer. Therefore, they do not allow invalid data to be replaced or corrected within the encoding or decoding method.

Implementing a Custom Fallback Strategy

In addition to the best-fit mapping that is implemented internally by code pages, .NET includes the following classes for implementing a fallback strategy:

- Use [EncoderReplacementFallback](#) and [EncoderReplacementFallbackBuffer](#) to replace characters in encoding operations.
- Use [DecoderReplacementFallback](#) and [DecoderReplacementFallbackBuffer](#) to replace characters in decoding operations.
- Use [EncoderExceptionFallback](#) and [EncoderExceptionFallbackBuffer](#) to throw an [EncoderFallbackException](#) when a character cannot be encoded.
- Use [DecoderExceptionFallback](#) and [DecoderExceptionFallbackBuffer](#) to throw a [DecoderFallbackException](#) when a character cannot be decoded.

In addition, you can implement a custom solution that uses best-fit fallback, replacement fallback, or exception fallback, by following these steps:

1. Derive a class from [EncoderFallback](#) for encoding operations, and from [DecoderFallback](#) for decoding operations.
2. Derive a class from [EncoderFallbackBuffer](#) for encoding operations, and from [DecoderFallbackBuffer](#) for decoding operations.
3. For exception fallback, if the predefined [EncoderFallbackException](#) and [DecoderFallbackException](#) classes do not meet your needs, derive a class from an exception object such as [Exception](#) or [ArgumentException](#).

Deriving from EncoderFallback or DecoderFallback

To implement a custom fallback solution, you must create a class that inherits from [EncoderFallback](#) for encoding operations, and from [DecoderFallback](#) for decoding operations. Instances of these classes are passed

to the `Encoding.GetEncoding(String, EncoderFallback, DecoderFallback)` method and serve as the intermediary between the encoding class and the fallback implementation.

When you create a custom fallback solution for an encoder or decoder, you must implement the following members:

- The `EncoderFallback.MaxCharCount` or `DecoderFallback.MaxCharCount` property, which returns the maximum possible number of characters that the best-fit, replacement, or exception fallback can return to replace a single character. For a custom exception fallback, its value is zero.
- The `EncoderFallback.CreateFallbackBuffer` or `DecoderFallback.CreateFallbackBuffer` method, which returns your custom `EncoderFallbackBuffer` or `DecoderFallbackBuffer` implementation. The method is called by the encoder when it encounters the first character that it is unable to successfully encode, or by the decoder when it encounters the first byte that it is unable to successfully decode.

Deriving from `EncoderFallbackBuffer` or `DecoderFallbackBuffer`

To implement a custom fallback solution, you must also create a class that inherits from `EncoderFallbackBuffer` for encoding operations, and from `DecoderFallbackBuffer` for decoding operations. Instances of these classes are returned by the `CreateFallbackBuffer` method of the `EncoderFallback` and `DecoderFallback` classes. The `EncoderFallback.CreateFallbackBuffer` method is called by the encoder when it encounters the first character that it is not able to encode, and the `DecoderFallback.CreateFallbackBuffer` method is called by the decoder when it encounters one or more bytes that it is not able to decode. The `EncoderFallbackBuffer` and `DecoderFallbackBuffer` classes provide the fallback implementation. Each instance represents a buffer that contains the fallback characters that will replace the character that cannot be encoded or the byte sequence that cannot be decoded.

When you create a custom fallback solution for an encoder or decoder, you must implement the following members:

- The `EncoderFallbackBufferFallback` or `DecoderFallbackBufferFallback` method. `EncoderFallbackBufferFallback` is called by the encoder to provide the fallback buffer with information about the character that it cannot encode. Because the character to be encoded may be a surrogate pair, this method is overloaded. One overload is passed the character to be encoded and its index in the string. The second overload is passed the high and low surrogate along with its index in the string. The `DecoderFallbackBufferFallback` method is called by the decoder to provide the fallback buffer with information about the bytes that it cannot decode. This method is passed an array of bytes that it cannot decode, along with the index of the first byte. The fallback method should return `true` if the fallback buffer can supply a best-fit or replacement character or characters; otherwise, it should return `false`. For an exception fallback, the fallback method should throw an exception.
- The `EncoderFallbackBuffer.GetNextChar` or `DecoderFallbackBuffer.GetNextChar` method, which is called repeatedly by the encoder or decoder to get the next character from the fallback buffer. When all fallback characters have been returned, the method should return `U+0000`.
- The `EncoderFallbackBuffer.Remaining` or `DecoderFallbackBuffer.Remaining` property, which returns the number of characters remaining in the fallback buffer.
- The `EncoderFallbackBuffer.MovePrevious` or `DecoderFallbackBuffer.MovePrevious` method, which moves the current position in the fallback buffer to the previous character.
- The `EncoderFallbackBuffer.Reset` or `DecoderFallbackBuffer.Reset` method, which reinitializes the fallback buffer.

If the fallback implementation is a best-fit fallback or a replacement fallback, the classes derived from `EncoderFallbackBuffer` and `DecoderFallbackBuffer` also maintain two private instance fields: the exact number of characters in the buffer; and the index of the next character in the buffer to return.

An EncoderFallback Example

An earlier example used replacement fallback to replace Unicode characters that did not correspond to ASCII characters with an asterisk (*). The following example uses a custom best-fit fallback implementation instead to provide a better mapping of non-ASCII characters.

The following code defines a class named `CustomMapper` that is derived from `EncoderFallback` to handle the best-fit mapping of non-ASCII characters. Its `CreateFallbackBuffer` method returns a `CustomMapperFallbackBuffer` object, which provides the `EncoderFallbackBuffer` implementation. The `CustomMapper` class uses a `Dictionary< TKey, TValue >` object to store the mappings of unsupported Unicode characters (the key value) and their corresponding 8-bit characters (which are stored in two consecutive bytes in a 64-bit integer). To make this mapping available to the fallback buffer, the `CustomMapper` instance is passed as a parameter to the `CustomMapperFallbackBuffer` class constructor. Because the longest mapping is the string "INF" for the Unicode character U+221E, the `MaxCharCount` property returns 3.

```
public class CustomMapper : EncoderFallback
{
    public string DefaultString;
    internal Dictionary<ushort, ulong> mapping;

    public CustomMapper() : this("*")
    {
    }

    public CustomMapper(string defaultString)
    {
        this.DefaultString = defaultString;

        // Create table of mappings
        mapping = new Dictionary<ushort, ulong>();
        mapping.Add(0x24C8, 0x53);
        mapping.Add(0x2075, 0x35);
        mapping.Add(0x221E, 0x49004E0046);
    }

    public override EncoderFallbackBuffer CreateFallbackBuffer()
    {
        return new CustomMapperFallbackBuffer(this);
    }

    public override int MaxCharCount
    {
        get { return 3; }
    }
}
```

```

Public Class CustomMapper : Inherits EncoderFallback
    Public DefaultString As String
    Friend mapping As Dictionary(Of UShort, UInt)
    
    Public Sub New()
        Me.New("?")
    End Sub

    Public Sub New(ByVal defaultString As String)
        Me.DefaultString = defaultString

        ' Create table of mappings
        mapping = New Dictionary(Of UShort, UInt)
        mapping.Add(&H24C8, &H53)
        mapping.Add(&H2075, &H35)
        mapping.Add(&H221E, &H49004E0046)
    End Sub

    Public Overrides Function CreateFallbackBuffer() As System.Text.EncoderFallbackBuffer
        Return New CustomMapperFallbackBuffer(Me)
    End Function

    Public Overrides ReadOnly Property MaxCharCount As Integer
        Get
            Return 3
        End Get
    End Property
End Class

```

The following code defines the `CustomMapperFallbackBuffer` class, which is derived from `EncoderFallbackBuffer`. The dictionary that contains best-fit mappings and that is defined in the `CustomMapper` instance is available from its class constructor. Its `Fallback` method returns `true` if any of the Unicode characters that the ASCII encoder cannot encode are defined in the mapping dictionary; otherwise, it returns `false`. For each fallback, the private `count` variable indicates the number of characters that remain to be returned, and the private `index` variable indicates the position in the string buffer, `charsToReturn`, of the next character to return.

```

public class CustomMapperFallbackBuffer : EncoderFallbackBuffer
{
    int count = -1;                      // Number of characters to return
    int index = -1;                      // Index of character to return
    CustomMapper fb;
    string charsToReturn;

    public CustomMapperFallbackBuffer(CustomMapper fallback)
    {
        this.fb = fallback;
    }

    public override bool Fallback(char charUnknownHigh, char charUnknownLow, int index)
    {
        // Do not try to map surrogates to ASCII.
        return false;
    }

    public override bool Fallback(char charUnknown, int index)
    {
        // Return false if there are already characters to map.
        if (count >= 1) return false;

        // Determine number of characters to return.
        charsToReturn = String.Empty;

        ushort key = Convert.ToInt16(charUnknown);
        if (fb.mapping.ContainsKey(key)) {

```

```

byte[] bytes = BitConverter.GetBytes(fb.mapping[key]);
int ctr = 0;
foreach (var byt in bytes) {
    if (byt > 0) {
        ctr++;
        charsToReturn += (char) byt;
    }
}
count = ctr;
}
else {
    // Return default.
    charsToReturn = fb.DefaultString;
    count = 1;
}
this.index = charsToReturn.Length - 1;

return true;
}

public override char GetNextChar()
{
    // We'll return a character if possible, so subtract from the count of chars to return.
    count--;
    // If count is less than zero, we've returned all characters.
    if (count < 0)
        return '\u0000';

    this.index--;
    return charsToReturn[this.index + 1];
}

public override bool MovePrevious()
{
    // Original: if count >= -1 and pos >= 0
    if (count >= -1) {
        count++;
        return true;
    }
    else {
        return false;
    }
}

public override int Remaining
{
    get { return count < 0 ? 0 : count; }
}

public override void Reset()
{
    count = -1;
    index = -1;
}
}

```

```

Public Class CustomMapperFallbackBuffer : Inherits EncoderFallbackBuffer

Dim count As Integer = -1          ' Number of characters to return
Dim index As Integer = -1         ' Index of character to return
Dim fb As CustomMapper
Dim charsToReturn As String

Public Sub New(ByVal fallback As CustomMapper)
    MyBase.New()
    Me.fb = fallback
End Sub

```

```

    Public Overloads Overrides Function Fallback(ByVal charUnknownHigh As Char, ByVal charUnknownLow As
Char, ByVal index As Integer) As Boolean
        ' Do not try to map surrogates to ASCII.
        Return False
    End Function

    Public Overloads Overrides Function Fallback(ByVal charUnknown As Char, ByVal index As Integer) As
Boolean
        ' Return false if there are already characters to map.
        If count >= 1 Then Return False

        ' Determine number of characters to return.
        charsToReturn = String.Empty

        Dim key As UInt16 = Convert.ToUInt16(charUnknown)
        If fb.mapping.ContainsKey(key) Then
            Dim bytes() As Byte = BitConverter.GetBytes(fb.mapping.Item(key))
            Dim ctr As Integer
            For Each byt In bytes
                If byt > 0 Then
                    ctr += 1
                    charsToReturn += Chr(byt)
                End If
            Next
            count = ctr
        Else
            ' Return default.
            charsToReturn = fb.DefaultString
            count = 1
        End If
        Me.index = charsToReturn.Length - 1

        Return True
    End Function

    Public Overrides Function GetNextChar() As Char
        ' We'll return a character if possible, so subtract from the count of chars to return.
        count -= 1
        ' If count is less than zero, we've returned all characters.
        If count < 0 Then Return ChrW(0)

        Me.index -= 1
        Return charsToReturn(Me.index + 1)
    End Function

    Public Overrides Function MovePrevious() As Boolean
        ' Original: if count >= -1 and pos >= 0
        If count >= -1 Then
            count += 1
            Return True
        Else
            Return False
        End If
    End Function

    Public Overrides ReadOnly Property Remaining As Integer
        Get
            Return If(count < 0, 0, count)
        End Get
    End Property

    Public Overrides Sub Reset()
        count = -1
        index = -1
    End Sub
End Class

```

The following code then instantiates the `CustomMapper` object and passes an instance of it to the `Encoding.GetEncoding(String, EncoderFallback, DecoderFallback)` method. The output indicates that the best-fit fallback implementation successfully handles the three non-ASCII characters in the original string.

```
using System;
using System.Collections.Generic;
using System.Text;

class Program
{
    static void Main()
    {
        Encoding enc = Encoding.GetEncoding("us-ascii", new CustomMapper(), new DecoderExceptionFallback());

        string str1 = "\u24C8 \u2075 \u221E";
        Console.WriteLine(str1);
        for (int ctr = 0; ctr <= str1.Length - 1; ctr++) {
            Console.Write("{0} ", Convert.ToInt16(str1[ctr]).ToString("X4"));
            if (ctr == str1.Length - 1)
                Console.WriteLine();
        }
        Console.WriteLine();

        // Encode the original string using the ASCII encoder.
        byte[] bytes = enc.GetBytes(str1);
        Console.Write("Encoded bytes: ");
        foreach (var byt in bytes)
            Console.Write("{0:X2} ", byt);

        Console.WriteLine("\n");

        // Decode the ASCII bytes.
        string str2 = enc.GetString(bytes);
        Console.WriteLine("Round-trip: {0}", str1.Equals(str2));
        if (! str1.Equals(str2)) {
            Console.WriteLine(str2);
            foreach (var ch in str2)
                Console.Write("{0} ", Convert.ToInt16(ch).ToString("X4"));

            Console.WriteLine();
        }
    }
}
```

```
Imports System.Text
Imports System.Collections.Generic

Module Module1

    Sub Main()
        Dim enc As Encoding = Encoding.GetEncoding("us-ascii", New CustomMapper(), New DecoderExceptionFallback())

        Dim str1 As String = String.Format("{0} {1} {2}", ChrW(&H24C8), ChrW(&H2075), ChrW(&H221E))
        Console.WriteLine(str1)
        For ctr As Integer = 0 To str1.Length - 1
            Console.Write("{0} ", Convert.ToInt16(str1(ctr)).ToString("X4"))
            If ctr = str1.Length - 1 Then Console.WriteLine()
        Next
        Console.WriteLine()

        ' Encode the original string using the ASCII encoder.
        Dim bytes() As Byte = enc.GetBytes(str1)
        Console.Write("Encoded bytes: ")
        For Each byt In bytes
            Console.Write("{0:X2} ", byt)
        Next
        Console.WriteLine()
        Console.WriteLine()

        ' Decode the ASCII bytes.
        Dim str2 As String = enc.GetString(bytes)
        Console.WriteLine("Round-trip: {0}", str1.Equals(str2))
        If Not str1.Equals(str2) Then
            Console.WriteLine(str2)
            For Each ch In str2
                Console.Write("{0} ", Convert.ToInt16(ch).ToString("X4"))
            Next
            Console.WriteLine()
        End If
    End Sub
End Module
```

See also

- [Introduction to character encoding in .NET](#)
- [Encoder](#)
- [Decoder](#)
- [DecoderFallback](#)
- [Encoding](#)
- [EncoderFallback](#)
- [Globalization and localization](#)

Best practices for comparing strings in .NET

9/20/2022 • 27 minutes to read • [Edit Online](#)

.NET provides extensive support for developing localized and globalized applications, and makes it easy to apply the conventions of either the current culture or a specific culture when performing common operations such as sorting and displaying strings. But sorting or comparing strings is not always a culture-sensitive operation. For example, strings that are used internally by an application typically should be handled identically across all cultures. When culturally independent string data, such as XML tags, HTML tags, user names, file paths, and the names of system objects, are interpreted as if they were culture-sensitive, application code can be subject to subtle bugs, poor performance, and, in some cases, security issues.

This article examines the string sorting, comparison, and casing methods in .NET, presents recommendations for selecting an appropriate string-handling method, and provides additional information about string-handling methods.

Recommendations for string usage

When you develop with .NET, follow these simple recommendations when you compare strings:

- Use overloads that explicitly specify the string comparison rules for string operations. Typically, this involves calling a method overload that has a parameter of type `StringComparison`.
- Use `StringComparison.Ordinal` or `StringComparison.OrdinalIgnoreCase` for comparisons as your safe default for culture-agnostic string matching.
- Use comparisons with `StringComparison.Ordinal` or `StringComparison.OrdinalIgnoreCase` for better performance.
- Use string operations that are based on `StringComparison.CurrentCulture` when you display output to the user.
- Use the non-linguistic `StringComparison.Ordinal` or `StringComparison.OrdinalIgnoreCase` values instead of string operations based on `CultureInfo.InvariantCulture` when the comparison is linguistically irrelevant (symbolic, for example).
- Use the `String.ToUpperInvariant` method instead of the `String.ToLowerInvariant` method when you normalize strings for comparison.
- Use an overload of the `String.Equals` method to test whether two strings are equal.
- Use the `String.Compare` and `String.CompareTo` methods to sort strings, not to check for equality.
- Use culture-sensitive formatting to display non-string data, such as numbers and dates, in a user interface. Use formatting with the `invariant culture` to persist non-string data in string form.

Avoid the following practices when you compare strings:

- Do not use overloads that do not explicitly or implicitly specify the string comparison rules for string operations.
- Do not use string operations based on `StringComparison.InvariantCulture` in most cases. One of the few exceptions is when you are persisting linguistically meaningful but culturally agnostic data.
- Do not use an overload of the `String.Compare` or `CompareTo` method and test for a return value of zero to determine whether two strings are equal.

Specifying string comparisons explicitly

Most of the string manipulation methods in .NET are overloaded. Typically, one or more overloads accept default

settings, whereas others accept no defaults and instead define the precise way in which strings are to be compared or manipulated. Most of the methods that do not rely on defaults include a parameter of type [StringComparison](#), which is an enumeration that explicitly specifies rules for string comparison by culture and case. The following table describes the [StringComparison](#) enumeration members.

StringComparison Member	Description
 CurrentCulture	Performs a case-sensitive comparison using the current culture.
 CurrentCultureIgnoreCase	Performs a case-insensitive comparison using the current culture.
 InvariantCulture	Performs a case-sensitive comparison using the invariant culture.
 InvariantCultureIgnoreCase	Performs a case-insensitive comparison using the invariant culture.
 Ordinal	Performs an ordinal comparison.
 OrdinalIgnoreCase	Performs a case-insensitive ordinal comparison.

For example, the [IndexOf](#) method, which returns the index of a substring in a [String](#) object that matches either a character or a string, has nine overloads:

- [IndexOf\(Char\)](#), [IndexOf\(Char, Int32\)](#), and [IndexOf\(Char, Int32, Int32\)](#), which by default perform an ordinal (case-sensitive and culture-insensitive) search for a character in the string.
- [IndexOf\(String\)](#), [IndexOf\(String, Int32\)](#), and [IndexOf\(String, Int32, Int32\)](#), which by default perform a case-sensitive and culture-sensitive search for a substring in the string.
- [IndexOf\(String, StringComparison\)](#), [IndexOf\(String, Int32, StringComparison\)](#), and [IndexOf\(String, Int32, Int32, StringComparison\)](#), which include a parameter of type [StringComparison](#) that allows the form of the comparison to be specified.

We recommend that you select an overload that does not use default values, for the following reasons:

- Some overloads with default parameters (those that search for a [Char](#) in the string instance) perform an ordinal comparison, whereas others (those that search for a string in the string instance) are culture-sensitive. It is difficult to remember which method uses which default value, and easy to confuse the overloads.
- The intent of the code that relies on default values for method calls is not clear. In the following example, which relies on defaults, it is difficult to know whether the developer actually intended an ordinal or a linguistic comparison of two strings, or whether a case difference between `protocol` and "http" might cause the test for equality to return `false`.

```
string protocol = GetProtocol(url);
if (String.Equals(protocol, "http")) {
    // ...Code to handle HTTP protocol.
}
else {
    throw new InvalidOperationException();
}
```

```
Dim protocol As String = GetProtocol(url)
If String.Equals(protocol, "http") Then
    ' ...Code to handle HTTP protocol.
Else
    Throw New InvalidOperationException()
End If
```

In general, we recommend that you call a method that does not rely on defaults, because it makes the intent of the code unambiguous. This, in turn, makes the code more readable and easier to debug and maintain. The following example addresses the questions raised about the previous example. It makes it clear that ordinal comparison is used and that differences in case are ignored.

```
string protocol = GetProtocol(url);
if (String.Equals(protocol, "http", StringComparison.OrdinalIgnoreCase)) {
    // ...Code to handle HTTP protocol.
}
else {
    throw new InvalidOperationException();
}
```

```
Dim protocol As String = GetProtocol(url)
If String.Equals(protocol, "http", StringComparison.OrdinalIgnoreCase) Then
    ' ...Code to handle HTTP protocol.
Else
    Throw New InvalidOperationException()
End If
```

The details of string comparison

String comparison is the heart of many string-related operations, particularly sorting and testing for equality. Strings sort in a determined order: If "my" appears before "string" in a sorted list of strings, "my" must compare less than or equal to "string". Additionally, comparison implicitly defines equality. The comparison operation returns zero for strings it deems equal. A good interpretation is that neither string is less than the other. Most meaningful operations involving strings include one or both of these procedures: comparing with another string, and executing a well-defined sort operation.

NOTE

You can download the [Sorting Weight Tables](#), a set of text files that contain information on the character weights used in sorting and comparison operations for Windows operating systems, and the [Default Unicode Collation Element Table](#), the latest version of the sort weight table for Linux and macOS. The specific version of the sort weight table on Linux and macOS depends on the version of the [International Components for Unicode](#) libraries installed on the system. For information on ICU versions and the Unicode versions that they implement, see [Downloading ICU](#).

However, evaluating two strings for equality or sort order does not yield a single, correct result; the outcome depends on the criteria used to compare the strings. In particular, string comparisons that are ordinal or that are based on the casing and sorting conventions of the current culture or the [invariant culture](#) (a locale-agnostic culture based on the English language) may produce different results.

In addition, string comparisons using different versions of .NET or using .NET on different operating systems or operating system versions may return different results. For more information, see [Strings and the Unicode Standard](#).

String comparisons that use the current culture

One criterion involves using the conventions of the current culture when comparing strings. Comparisons that are based on the current culture use the thread's current culture or locale. If the culture is not set by the user, it defaults to the setting in the **Regional Options** window in Control Panel. You should always use comparisons that are based on the current culture when data is linguistically relevant, and when it reflects culture-sensitive user interaction.

However, comparison and casing behavior in .NET changes when the culture changes. This happens when an application executes on a computer that has a different culture than the computer on which the application was developed, or when the executing thread changes its culture. This behavior is intentional, but it remains non-obvious to many developers. The following example illustrates differences in sort order between the U.S. English ("en-US") and Swedish ("sv-SE") cultures. Note that the words "ångström", "Windows", and "Visual Studio" appear in different positions in the sorted string arrays.

```
using System;
using System.Globalization;
using System.Threading;

public class Example
{
    public static void Main()
    {
        string[] values= { "able", "ångström", "apple", "Æble",
                          "Windows", "Visual Studio" };
        Array.Sort(values);
        DisplayArray(values);

        // Change culture to Swedish (Sweden).
        string originalCulture = CultureInfo.CurrentCulture.Name;
        Thread.CurrentThread.CurrentCulture = new CultureInfo("sv-SE");
        Array.Sort(values);
        DisplayArray(values);

        // Restore the original culture.
        Thread.CurrentThread.CurrentCulture = new CultureInfo(originalCulture);
    }

    private static void DisplayArray(string[] values)
    {
        Console.WriteLine("Sorting using the {0} culture:",
                         CultureInfo.CurrentCulture.Name);
        foreach (string value in values)
            Console.WriteLine("    {0}", value);

        Console.WriteLine();
    }
}

// The example displays the following output:
//      Sorting using the en-US culture:
//          able
//          Æble
//          ångström
//          apple
//          Visual Studio
//          Windows
//
//      Sorting using the sv-SE culture:
//          able
//          Æble
//          apple
//          Windows
//          Visual Studio
//          ångström
```

```

Imports System.Globalization
Imports System.Threading

Module Example
    Public Sub Main()
        Dim values() As String = {"able", "ångström", "apple", _
                               "Æble", "Windows", "Visual Studio"}
        Array.Sort(values)
        DisplayArray(values)

        ' Change culture to Swedish (Sweden).
        Dim originalCulture As String = CultureInfo.CurrentCulture.Name
        Thread.CurrentThread.CurrentCulture = New CultureInfo("sv-SE")
        Array.Sort(values)
        DisplayArray(values)

        ' Restore the original culture.
        Thread.CurrentThread.CurrentCulture = New CultureInfo(originalCulture)
    End Sub

    Private Sub DisplayArray(values() As String)
        Console.WriteLine("Sorting using the {0} culture:", _
                          CultureInfo.CurrentCulture.Name)
        For Each value As String In values
            Console.WriteLine("    {0}", value)
        Next
        Console.WriteLine()
    End Sub
End Module
' The example displays the following output:
'     Sorting using the en-US culture:
'     able
'     Æble
'     ångström
'     apple
'     Visual Studio
'     Windows
'
'     Sorting using the sv-SE culture:
'     able
'     Æble
'     apple
'     Windows
'     Visual Studio
'     ångström

```

Case-insensitive comparisons that use the current culture are the same as culture-sensitive comparisons, except that they ignore case as dictated by the thread's current culture. This behavior may manifest itself in sort orders as well.

Comparisons that use current culture semantics are the default for the following methods:

- [String.Compare](#) overloads that do not include a [StringComparison](#) parameter.
- [String.CompareTo](#) overloads.
- The default [String.StartsWith\(String\)](#) method, and the [String.StartsWith\(String, Boolean, CultureInfo\)](#) method with a `null` [CultureInfo](#) parameter.
- The default [String.EndsWith\(String\)](#) method, and the [String.EndsWith\(String, Boolean, CultureInfo\)](#) method with a `null` [CultureInfo](#) parameter.
- [String.IndexOf](#) overloads that accept a [String](#) as a search parameter and that do not have a [StringComparison](#) parameter.
- [String.LastIndexOf](#) overloads that accept a [String](#) as a search parameter and that do not have a [StringComparison](#) parameter.

In any case, we recommend that you call an overload that has a [StringComparison](#) parameter to make the intent of the method call clear.

Subtle and not so subtle bugs can emerge when non-linguistic string data is interpreted linguistically, or when string data from a particular culture is interpreted using the conventions of another culture. The canonical example is the Turkish-I problem.

For nearly all Latin alphabets, including U.S. English, the character "i" (\u0069) is the lowercase version of the character "I" (\u0049). This casing rule quickly becomes the default for someone programming in such a culture. However, the Turkish ("tr-TR") alphabet includes an "I with a dot" character "İ" (\u0130), which is the capital version of "i". Turkish also includes a lowercase "i without a dot" character, "ı" (\u0131), which capitalizes to "I". This behavior occurs in the Azerbaijani ("az") culture as well.

Therefore, assumptions made about capitalizing "i" or lowercasing "I" are not valid among all cultures. If you use the default overloads for string comparison routines, they will be subject to variance between cultures. If the data to be compared is non-linguistic, using the default overloads can produce undesirable results, as the following attempt to perform a case-insensitive comparison of the strings "file" and "FILE" illustrates.

```
using System;
using System.Globalization;
using System.Threading;

public class Example
{
    public static void Main()
    {
        string fileUrl = "file";
        Thread.CurrentCulture.CurrentCulture = new CultureInfo("en-US");
        Console.WriteLine("Culture = {0}",
                          Thread.CurrentCulture.CurrentCulture.DisplayName);
        Console.WriteLine("(file == FILE) = {0}",
                          fileUrl.StartsWith("FILE", true, null));
        Console.WriteLine();

        Thread.CurrentCulture.CurrentCulture = new CultureInfo("tr-TR");
        Console.WriteLine("Culture = {0}",
                          Thread.CurrentCulture.CurrentCulture.DisplayName);
        Console.WriteLine("(file == FILE) = {0}",
                          fileUrl.StartsWith("FILE", true, null));
    }
}
// The example displays the following output:
//      Culture = English (United States)
//      (file == FILE) = True
//
//      Culture = Turkish (Turkey)
//      (file == FILE) = False
```

```

Imports System.Globalization
Imports System.Threading

Module Example
    Public Sub Main()
        Dim fileUrl = "file"
        Thread.CurrentThread.CurrentCulture = New CultureInfo("en-US")
        Console.WriteLine("Culture = {0}", _
                          Thread.CurrentThread.CurrentCulture.DisplayName)
        Console.WriteLine("(file == FILE) = {0}", _
                          fileUrl.StartsWith("FILE", True, Nothing))
        Console.WriteLine()

        Thread.CurrentThread.CurrentCulture = New CultureInfo("tr-TR")
        Console.WriteLine("Culture = {0}", _
                          Thread.CurrentThread.CurrentCulture.DisplayName)
        Console.WriteLine("(file == FILE) = {0}", _
                          fileUrl.StartsWith("FILE", True, Nothing))
    End Sub
End Module
' The example displays the following output:
'     Culture = English (United States)
'     (file == FILE) = True
'
'     Culture = Turkish (Turkey)
'     (file == FILE) = False

```

This comparison could cause significant problems if the culture is inadvertently used in security-sensitive settings, as in the following example. A method call such as `IsFileURI("file:")` returns `true` if the current culture is U.S. English, but `false` if the current culture is Turkish. Thus, on Turkish systems, someone could circumvent security measures that block access to case-insensitive URLs that begin with "FILE:".

```

public static bool IsFileURI(String path)
{
    return path.StartsWith("FILE:", true, null);
}

```

```

Public Shared Function IsFileURI(path As String) As Boolean
    Return path.StartsWith("FILE:", True, Nothing)
End Function

```

In this case, because "file:" is meant to be interpreted as a non-linguistic, culture-insensitive identifier, the code should instead be written as shown in the following example:

```

public static bool IsFileURI(string path)
{
    return path.StartsWith("FILE:", StringComparison.OrdinalIgnoreCase);
}

```

```

Public Shared Function IsFileURI(path As String) As Boolean
    Return path.StartsWith("FILE:", StringComparison.OrdinalIgnoreCase)
End Function

```

Ordinal string operations

Specifying the `StringComparison.Ordinal` or `StringComparison.OrdinalIgnoreCase` value in a method call signifies a non-linguistic comparison in which the features of natural languages are ignored. Methods that are invoked with these `StringComparison` values base string operation decisions on simple byte comparisons

instead of casing or equivalence tables that are parameterized by culture. In most cases, this approach best fits the intended interpretation of strings while making code faster and more reliable.

Ordinal comparisons are string comparisons in which each byte of each string is compared without linguistic interpretation; for example, "windows" does not match "Windows". This is essentially a call to the C runtime `strcmp` function. Use this comparison when the context dictates that strings should be matched exactly or demands conservative matching policy. Additionally, ordinal comparison is the fastest comparison operation because it applies no linguistic rules when determining a result.

Strings in .NET can contain embedded null characters. One of the clearest differences between ordinal and culture-sensitive comparison (including comparisons that use the invariant culture) concerns the handling of embedded null characters in a string. These characters are ignored when you use the [String.Compare](#) and [String.Equals](#) methods to perform culture-sensitive comparisons (including comparisons that use the invariant culture). As a result, in culture-sensitive comparisons, strings that contain embedded null characters can be considered equal to strings that do not.

IMPORTANT

Although string comparison methods disregard embedded null characters, string search methods such as [String.Contains](#), [String.EndsWith](#), [String.IndexOf](#), [String.LastIndexOf](#), and [String.StartsWith](#) do not.

The following example performs a culture-sensitive comparison of the string "Aa" with a similar string that contains several embedded null characters between "A" and "a", and shows how the two strings are considered equal:

```

using System;

public class Example
{
    public static void Main()
    {
        string str1 = "Aa";
        string str2 = "A" + new String('\u0000', 3) + "a";
        Console.WriteLine("Comparing '{0}' ({1}) and '{2}' ({3}):",
                          str1, ShowBytes(str1), str2, ShowBytes(str2));
        Console.WriteLine("    With String.Compare:");
        Console.WriteLine("        Current Culture: {0}",
                          String.Compare(str1, str2, StringComparison.CurrentCulture));
        Console.WriteLine("        Invariant Culture: {0}",
                          String.Compare(str1, str2, StringComparison.InvariantCulture));

        Console.WriteLine("    With String.Equals:");
        Console.WriteLine("        Current Culture: {0}",
                          String.Equals(str1, str2, StringComparison.CurrentCulture));
        Console.WriteLine("        Invariant Culture: {0}",
                          String.Equals(str1, str2, StringComparison.InvariantCulture));
    }

    private static string ShowBytes(string str)
    {
        string hexString = String.Empty;
        for (int ctr = 0; ctr < str.Length; ctr++)
        {
            string result = String.Empty;
            result = Convert.ToInt32(str[ctr]).ToString("X4");
            result = " " + result.Substring(0,2) + " " + result.Substring(2, 2);
            hexString += result;
        }
        return hexString.Trim();
    }
}

// The example displays the following output:
// Comparing 'Aa' (00 41 00 61) and 'A a' (00 41 00 00 00 00 00 00 61):
//     With String.Compare:
//         Current Culture: 0
//         Invariant Culture: 0
//     With String.Equals:
//         Current Culture: True
//         Invariant Culture: True

```

```

Module Example
    Public Sub Main()
        Dim str1 As String = "Aa"
        Dim str2 As String = "A" + New String(Convert.ToChar(0), 3) + "a"
        Console.WriteLine("Comparing '{0}' ({1}) and '{2}' ({3}):",
            str1, ShowBytes(str1), str2, ShowBytes(str2))
        Console.WriteLine("    With String.Compare:")
        Console.WriteLine("        Current Culture: {0}",
            String.Compare(str1, str2, StringComparison.CurrentCulture))
        Console.WriteLine("        Invariant Culture: {0}",
            String.Compare(str1, str2, StringComparison.InvariantCulture))

        Console.WriteLine("    With String.Equals:")
        Console.WriteLine("        Current Culture: {0}",
            String.Equals(str1, str2, StringComparison.CurrentCulture))
        Console.WriteLine("        Invariant Culture: {0}",
            String.Equals(str1, str2, StringComparison.InvariantCulture))

    End Sub

    Private Function ShowBytes(str As String) As String
        Dim hexString As String = String.Empty
        For ctr As Integer = 0 To str.Length - 1
            Dim result As String = String.Empty
            result = Convert.ToInt32(str.Chars(ctr)).ToString("X4")
            result = " " + result.Substring(0, 2) + " " + result.Substring(2, 2)
            hexString += result
        Next
        Return hexString.Trim()
    End Function
End Module

```

However, the strings are not considered equal when you use ordinal comparison, as the following example shows:

```

Console.WriteLine("Comparing '{0}' ({1}) and '{2}' ({3}):",
    str1, ShowBytes(str1), str2, ShowBytes(str2));
Console.WriteLine("    With String.Compare:");
Console.WriteLine("        Ordinal: {0}",
    String.Compare(str1, str2, StringComparison.OrdinalIgnoreCase));

Console.WriteLine("    With String.Equals:");
Console.WriteLine("        Ordinal: {0}",
    String.Equals(str1, str2, StringComparison.OrdinalIgnoreCase));
// The example displays the following output:
//      Comparing 'Aa' (00 41 00 61) and 'A  a' (00 41 00 00 00 00 00 61):
//      With String.Compare:
//          Ordinal: 97
//      With String.Equals:
//          Ordinal: False

```

```

Console.WriteLine("Comparing '{0}' ({1}) and '{2}' ({3}):", _
                  str1, ShowBytes(str1), str2, ShowBytes(str2))
Console.WriteLine("  With String.Compare:")
Console.WriteLine("    Ordinal: {0}", _
                  String.Compare(str1, str2, StringComparison.OrdinalIgnoreCase))

Console.WriteLine("  With String.Equals:")
Console.WriteLine("    Ordinal: {0}", _
                  String.Equals(str1, str2, StringComparison.OrdinalIgnoreCase))
' The example displays the following output:
'   Comparing 'Aa' (00 41 00 61) and 'A  a' (00 41 00 00 00 00 00 00 61):
'     With String.Compare:
'       Ordinal: 97
'     With String.Equals:
'       Ordinal: False

```

Case-insensitive ordinal comparisons are the next most conservative approach. These comparisons ignore most casing; for example, "windows" matches "Windows". When dealing with ASCII characters, this policy is equivalent to [StringComparison.OrdinalIgnoreCase](#), except that it ignores the usual ASCII casing. Therefore, any character in [A, Z] (\u0041-\u005A) matches the corresponding character in [a,z] (\u0061-\u007A). Casing outside the ASCII range uses the invariant culture's tables. Therefore, the following comparison:

```
String.Compare(strA, strB, StringComparison.OrdinalIgnoreCase);
```

```
String.Compare(strA, strB, StringComparison.OrdinalIgnoreCase)
```

is equivalent to (but faster than) this comparison:

```
String.Compare(strA.ToUpperInvariant(), strB.ToUpperInvariant(),
               StringComparison.OrdinalIgnoreCase);
```

```
String.Compare(strA.ToUpperInvariant(), strB.ToUpperInvariant(),
               StringComparison.OrdinalIgnoreCase)
```

These comparisons are still very fast.

Both [StringComparison.OrdinalIgnoreCase](#) and [StringComparison.OrdinalIgnoreCase](#) use the binary values directly, and are best suited for matching. When you are not sure about your comparison settings, use one of these two values. However, because they perform a byte-by-byte comparison, they do not sort by a linguistic sort order (like an English dictionary) but by a binary sort order. The results may look odd in most contexts if displayed to users.

Ordinal semantics are the default for [String.Equals](#) overloads that do not include a [StringComparison](#) argument (including the equality operator). In any case, we recommend that you call an overload that has a [StringComparison](#) parameter.

String operations that use the invariant culture

Comparisons with the invariant culture use the [CompareInfo](#) property returned by the static [CultureInfo.InvariantCulture](#) property. This behavior is the same on all systems; it translates any characters outside its range into what it believes are equivalent invariant characters. This policy can be useful for maintaining one set of string behavior across cultures, but it often provides unexpected results.

Case-insensitive comparisons with the invariant culture use the static [CompareInfo](#) property returned by the static [CultureInfo.InvariantCulture](#) property for comparison information as well. Any case differences among these translated characters are ignored.

Comparisons that use [StringComparison.InvariantCulture](#) and [StringComparison.OrdinalIgnoreCase](#) work identically on ASCII strings. However, [StringComparison.InvariantCulture](#) makes linguistic decisions that might not be appropriate for strings that have to be interpreted as a set of bytes. The

`CultureInfo.InvariantCulture.CompareInfo` object makes the [Compare](#) method interpret certain sets of characters as equivalent. For example, the following equivalence is valid under the invariant culture:

InvariantCulture: a + ° = å

The LATIN SMALL LETTER A character "a" (\u0061), when it is next to the COMBINING RING ABOVE character "+" "\u030a", is interpreted as the LATIN SMALL LETTER A WITH RING ABOVE character "å" (\u00e5). As the following example shows, this behavior differs from ordinal comparison.

```
string separated = "\u0061\u030a";
string combined = "\u00e5";

Console.WriteLine("Equal sort weight of {0} and {1} using InvariantCulture: {2}",
    separated, combined,
    String.Compare(separated, combined,
        StringComparison.InvariantCulture) == 0);

Console.WriteLine("Equal sort weight of {0} and {1} using Ordinal: {2}",
    separated, combined,
    String.Compare(separated, combined,
        StringComparison.Ordinal) == 0);
// The example displays the following output:
//   Equal sort weight of a° and å using InvariantCulture: True
//   Equal sort weight of a° and å using Ordinal: False
```

```
Dim separated As String = ChrW(&h61) + ChrW(&h30a)
Dim combined As String = ChrW(&he5)

Console.WriteLine("Equal sort weight of {0} and {1} using InvariantCulture: {2}", _
    separated, combined, _
    String.Compare(separated, combined, _
        StringComparison.InvariantCulture) = 0)

Console.WriteLine("Equal sort weight of {0} and {1} using Ordinal: {2}", _
    separated, combined, _
    String.Compare(separated, combined, _
        StringComparison.Ordinal) = 0)
' The example displays the following output:
'   Equal sort weight of a° and å using InvariantCulture: True
'   Equal sort weight of a° and å using Ordinal: False
```

When interpreting file names, cookies, or anything else where a combination such as "å" can appear, ordinal comparisons still offer the most transparent and fitting behavior.

On balance, the invariant culture has very few properties that make it useful for comparison. It does comparison in a linguistically relevant manner, which prevents it from guaranteeing full symbolic equivalence, but it is not the choice for display in any culture. One of the few reasons to use [StringComparison.InvariantCulture](#) for comparison is to persist ordered data for a cross-culturally identical display. For example, if a large data file that contains a list of sorted identifiers for display accompanies an application, adding to this list would require an insertion with invariant-style sorting.

Choosing a [StringComparison](#) member for your method call

The following table outlines the mapping from semantic string context to a [StringComparison](#) enumeration member:

DATA	BEHAVIOR	CORRESPONDING SYSTEM.STRINGCOMPARISON
		VALUE
Case-sensitive internal identifiers. Case-sensitive identifiers in standards such as XML and HTTP. Case-sensitive security-related settings.	A non-linguistic identifier, where bytes match exactly.	Ordinal
Case-insensitive internal identifiers. Case-insensitive identifiers in standards such as XML and HTTP. File paths. Registry keys and values. Environment variables. Resource identifiers (for example, handle names). Case-insensitive security-related settings.	A non-linguistic identifier, where case is irrelevant; especially data stored in most Windows system services.	OrdinalIgnoreCase
Some persisted, linguistically relevant data. Display of linguistic data that requires a fixed sort order.	Culturally agnostic data that still is linguistically relevant.	InvariantCulture -or- InvariantCultureIgnoreCase
Data displayed to the user. Most user input.	Data that requires local linguistic customs.	CurrentCulture -or- CurrentCultureIgnoreCase

Common string comparison methods in .NET

The following sections describe the methods that are most commonly used for string comparison.

String.Compare

Default interpretation: [StringComparison.CurrentCulture](#).

As the operation most central to string interpretation, all instances of these method calls should be examined to determine whether strings should be interpreted according to the current culture, or dissociated from the culture (symbolically). Typically, it is the latter, and a [StringComparison.Ordinal](#) comparison should be used instead.

The [System.Globalization.CompareInfo](#) class, which is returned by the [CultureInfo.CompareInfo](#) property, also includes a [Compare](#) method that provides a large number of matching options (ordinal, ignoring white space, ignoring kana type, and so on) by means of the [CompareOptions](#) flag enumeration.

String.CompareTo

Default interpretation: [StringComparison.CurrentCulture](#).

This method does not currently offer an overload that specifies a [StringComparison](#) type. It is usually possible to convert this method to the recommended [String.Compare\(String, String, StringComparison\)](#) form.

Types that implement the [IComparable](#) and [IComparable<T>](#) interfaces implement this method. Because it does not offer the option of a [StringComparison](#) parameter, implementing types often let the user specify a [StringComparer](#) in their constructor. The following example defines a [FileName](#) class whose class constructor includes a [StringComparer](#) parameter. This [StringComparer](#) object is then used in the [FileName.CompareTo](#) method.

```
using System;

public class FileName : IComparable
{
    string fname;
    StringComparer comparer;

    public FileName(string name, StringComparer comparer)
    {
        if (String.IsNullOrEmpty(name))
            throw new ArgumentNullException("name");

        this.fname = name;

        if (comparer != null)
            this.comparer = comparer;
        else
            this.comparer = StringComparer.OrdinalIgnoreCase;
    }

    public string Name
    {
        get { return fname; }
    }

    public int CompareTo(object obj)
    {
        if (obj == null) return 1;

        if (! (obj is FileName))
            return comparer.Compare(this.fname, obj.ToString());
        else
            return comparer.Compare(this.fname, ((FileName) obj).Name);
    }
}
```

```

Public Class FileName : Implements IComparable
    Dim fname As String
    Dim comparer As StringComparer

    Public Sub New(name As String, comparer As StringComparer)
        If String.IsNullOrEmpty(name) Then
            Throw New ArgumentNullException("name")
        End If

        Me.fname = name

        If comparer IsNot Nothing Then
            Me.comparer = comparer
        Else
            Me.comparer = StringComparer.OrdinalIgnoreCase
        End If
    End Sub

    Public ReadOnly Property Name As String
        Get
            Return fname
        End Get
    End Property

    Public Function CompareTo(obj As Object) As Integer _
        Implements IComparable.CompareTo
        If obj Is Nothing Then Return 1

        If Not TypeOf obj Is FileName Then
            obj = obj.ToString()
        Else
            obj = CType(obj, FileName).Name
        End If
        Return comparer.Compare(Me.fname, obj)
    End Function
End Class

```

String.Equals

Default interpretation: [StringComparison.Ordinal](#).

The [String](#) class lets you test for equality by calling either the static or instance [Equals](#) method overloads, or by using the static equality operator. The overloads and operator use ordinal comparison by default. However, we still recommend that you call an overload that explicitly specifies the [StringComparison](#) type even if you want to perform an ordinal comparison; this makes it easier to search code for a certain string interpretation.

String.ToUpper and String.ToLower

Default interpretation: [StringComparison.CurrentCulture](#).

Be careful when you use the [String.ToUpper\(\)](#) and [String.ToLower\(\)](#) methods, because forcing a string to a uppercase or lowercase is often used as a small normalization for comparing strings regardless of case. If so, consider using a case-insensitive comparison.

The [String.ToUpperInvariant](#) and [String.ToLowerInvariant](#) methods are also available. [ToUpperInvariant](#) is the standard way to normalize case. Comparisons made using [StringComparison.OrdinalIgnoreCase](#) are behaviorally the composition of two calls: calling [ToUpperInvariant](#) on both string arguments, and doing a comparison using [StringComparison.Ordinal](#).

Overloads are also available for converting to uppercase and lowercase in a specific culture, by passing a [CultureInfo](#) object that represents that culture to the method.

Char.ToUpper and Char.ToLower

Default interpretation: `StringComparison.CurrentCulture`.

The `Char.ToUpper(Char)` and `Char.ToLower(Char)` methods work similarly to the `String.ToUpper()` and `String.ToLower()` methods described in the previous section.

String.StartsWith and String.EndsWith

Default interpretation: `StringComparison.CurrentCulture`.

By default, both of these methods perform a culture-sensitive comparison.

String.IndexOf and String.LastIndexOf

Default interpretation: `StringComparison.CurrentCulture`.

There is a lack of consistency in how the default overloads of these methods perform comparisons. All `String.IndexOf` and `String.LastIndexOf` methods that include a `Char` parameter perform an ordinal comparison, but the default `String.IndexOf` and `String.LastIndexOf` methods that include a `String` parameter perform a culture-sensitive comparison.

If you call the `String.IndexOf(String)` or `String.LastIndexOf(String)` method and pass it a string to locate in the current instance, we recommend that you call an overload that explicitly specifies the `StringComparison` type. The overloads that include a `Char` argument do not allow you to specify a `StringComparison` type.

Methods that perform string comparison indirectly

Some non-string methods that have string comparison as a central operation use the `StringComparer` type. The `StringComparer` class includes six static properties that return `StringComparer` instances whose `StringComparer.Compare` methods perform the following types of string comparisons:

- Culture-sensitive string comparisons using the current culture. This `StringComparer` object is returned by the `StringComparer.CurrentCulture` property.
- Case-insensitive comparisons using the current culture. This `StringComparer` object is returned by the `StringComparer.CurrentCultureIgnoreCase` property.
- Culture-insensitive comparisons using the word comparison rules of the invariant culture. This `StringComparer` object is returned by the `StringComparer.InvariantCulture` property.
- Case-insensitive and culture-insensitive comparisons using the word comparison rules of the invariant culture. This `StringComparer` object is returned by the `StringComparer.InvariantCultureIgnoreCase` property.
- Ordinal comparison. This `StringComparer` object is returned by the `StringComparer.Ordinal` property.
- Case-insensitive ordinal comparison. This `StringComparer` object is returned by the `StringComparer.OrdinalIgnoreCase` property.

Array.Sort and Array.BinarySearch

Default interpretation: `StringComparison.CurrentCulture`.

When you store any data in a collection, or read persisted data from a file or database into a collection, switching the current culture can invalidate the invariants in the collection. The `Array.BinarySearch` method assumes that the elements in the array to be searched are already sorted. To sort any string element in the array, the `Array.Sort` method calls the `String.Compare` method to order individual elements. Using a culture-sensitive comparer can be dangerous if the culture changes between the time that the array is sorted and its contents are searched. For example, in the following code, storage and retrieval operate on the comparer that is provided implicitly by the `Thread.CurrentCulture` property. If the culture can change between the calls to `StoreNames` and `DoesNameExist`, and especially if the array contents are persisted somewhere between the two method calls, the binary search may fail.

```

// Incorrect.
string []storedNames;

public void StoreNames(string [] names)
{
    int index = 0;
    storedNames = new string[names.Length];

    foreach (string name in names)
    {
        this.storedNames[index++] = name;
    }

    Array.Sort(names); // Line A.
}

public bool DoesNameExist(string name)
{
    return (Array.BinarySearch(this.storedNames, name) >= 0); // Line B.
}

```

```

' Incorrect.
Dim storedNames() As String

Public Sub StoreNames(names() As String)
    Dim index As Integer = 0
    ReDim storedNames(names.Length - 1)

    For Each name As String In names
        Me.storedNames(index) = name
        index += 1
    Next

    Array.Sort(names)           ' Line A.
End Sub

Public Function DoesNameExist(name As String) As Boolean
    Return Array.BinarySearch(Me.storedNames, name) >= 0      ' Line B.
End Function

```

A recommended variation appears in the following example, which uses the same ordinal (culture-insensitive) comparison method both to sort and to search the array. The change code is reflected in the lines labeled [Line A](#) and [Line B](#) in the two examples.

```

// Correct.
string []storedNames;

public void StoreNames(string [] names)
{
    int index = 0;
    storedNames = new string[names.Length];

    foreach (string name in names)
    {
        this.storedNames[index++] = name;
    }

    Array.Sort(names, StringComparer.Ordinal); // Line A.
}

public bool DoesNameExist(string name)
{
    return (Array.BinarySearch(this.storedNames, name, StringComparer.Ordinal) >= 0); // Line B.
}

```

```

' Correct.
Dim storedNames() As String

Public Sub StoreNames(names() As String)
    Dim index As Integer = 0
    ReDim storedNames(names.Length - 1)

    For Each name As String In names
        Me.storedNames(index) = name
        index += 1
    Next

    Array.Sort(names, StringComparer.Ordinal)           ' Line A.
End Sub

Public Function DoesNameExist(name As String) As Boolean
    Return Array.BinarySearch(Me.storedNames, name, StringComparer.Ordinal) >= 0           ' Line B.
End Function

```

If this data is persisted and moved across cultures, and sorting is used to present this data to the user, you might consider using [StringComparison.InvariantCulture](#), which operates linguistically for better user output but is unaffected by changes in culture. The following example modifies the two previous examples to use the invariant culture for sorting and searching the array.

```

// Correct.
string []storedNames;

public void StoreNames(string [] names)
{
    int index = 0;
    storedNames = new string[names.Length];

    foreach (string name in names)
    {
        this.storedNames[index++] = name;
    }

    Array.Sort(names, StringComparer.InvariantCulture); // Line A.
}

public bool DoesNameExist(string name)
{
    return (Array.BinarySearch(this.storedNames, name, StringComparer.InvariantCulture) >= 0); // Line B.
}

```

```

' Correct.
Dim storedNames() As String

Public Sub StoreNames(names() As String)
    Dim index As Integer = 0
    ReDim storedNames(names.Length - 1)

    For Each name As String In names
        Me.storedNames(index) = name
        index += 1
    Next

    Array.Sort(names, StringComparer.InvariantCulture)           ' Line A.
End Sub

Public Function DoesNameExist(name As String) As Boolean
    Return Array.BinarySearch(Me.storedNames, name, StringComparer.InvariantCulture) >= 0           ' Line B.
End Function

```

Collections example: Hashtable constructor

Hashing strings provides a second example of an operation that is affected by the way in which strings are compared.

The following example instantiates a [Hashtable](#) object by passing it the [StringComparer](#) object that is returned by the [StringComparer.OrdinalIgnoreCase](#) property. Because a class [StringComparer](#) that is derived from [StringComparer](#) implements the [IEqualityComparer](#) interface, its [GetHashCode](#) method is used to compute the hash code of strings in the hash table.

```

const int initialTableCapacity = 100;
Hashtable h;

public void PopulateFileTable(string directory)
{
    h = new Hashtable(initialTableCapacity,
                      StringComparer.OrdinalIgnoreCase);

    foreach (string file in Directory.GetFiles(directory))
        h.Add(file, File.GetCreationTime(file));
}

public void PrintCreationTime(string targetFile)
{
    Object dt = h[targetFile];
    if (dt != null)
    {
        Console.WriteLine("File {0} was created at time {1}.",
                          targetFile,
                          (DateTime) dt);
    }
    else
    {
        Console.WriteLine("File {0} does not exist.", targetFile);
    }
}

```

```

Const initialTableCapacity As Integer = 100
Dim h As Hashtable

Public Sub PopulateFileTable(dir As String)
    h = New Hashtable(initialTableCapacity, _
                      StringComparer.OrdinalIgnoreCase)

    For Each filename As String In Directory.GetFiles(dir)
        h.Add(filename, File.GetCreationTime(filename))
    Next
End Sub

Public Sub PrintCreationTime(targetFile As String)
    Dim dt As Object = h(targetFile)
    If dt IsNot Nothing Then
        Console.WriteLine("File {0} was created at {1}.",
                          targetFile, _
                          CDate(dt))
    Else
        Console.WriteLine("File {0} does not exist.", targetFile)
    End If
End Sub

```

See also

- [Globalization in .NET apps](#)

Best practices for displaying and persisting formatted data

9/20/2022 • 7 minutes to read • [Edit Online](#)

This article examines how formatted data, such as numeric data and date-and-time data, is handled for display and for storage.

When you develop with .NET, use culture-sensitive formatting to display non-string data, such as numbers and dates, in a user interface. Use formatting with the [invariant culture](#) to persist non-string data in string form. Do not use culture-sensitive formatting to persist numeric or date-and-time data in string form.

Display formatted data

When you display non-string data such as numbers and dates and times to users, format them by using the user's cultural settings. By default, the following all use the current culture in formatting operations:

- Interpolated strings supported by the [C#](#) and [Visual Basic](#) compilers.
- String concatenation operations that use the [C#](#) or [Visual Basic](#) concatenation operators or that call the [String.Concat](#) method directly.
- The [String.Format](#) method.
- The [ToString](#) methods of the numeric types and the date and time types.

To explicitly specify that a string should be formatted by using the conventions of a designated culture or the [invariant culture](#), you can do the following:

- When using the [String.Format](#) and [ToString](#) methods, call an overload that has a [provider](#) parameter, such as [String.Format\(IFormatProvider, String, Object\[\]\)](#) or [DateTime.ToString\(IFormatProvider\)](#), and pass it the [CultureInfo.CurrentCulture](#) property, a [CultureInfo](#) instance that represents the desired culture, or the [CultureInfo.InvariantCulture](#) property.
- For string concatenation, do not allow the compiler to perform any implicit conversions. Instead, perform an explicit conversion by calling a [ToString](#) overload that has a [provider](#) parameter. For example, the compiler implicitly uses the current culture when converting a [Double](#) value to a string in the following code:

```
string concat1 = "The amount is " + 126.03 + ".";
Console.WriteLine(concat1);
```

```
Dim concat1 As String = "The amount is " & 126.03 & "."
Console.WriteLine(concat1)
```

Instead, you can explicitly specify the culture whose formatting conventions are used in the conversion by calling the [Double.ToString\(IFormatProvider\)](#) method, as the following code does:

```
string concat2 = "The amount is " + 126.03.ToString(CultureInfo.InvariantCulture) + ".";
Console.WriteLine(concat2);
```

```
Dim concat2 As String = "The amount is " & 126.03.ToString(CultureInfo.InvariantCulture) & "."
Console.WriteLine(concat2)
```

- For string interpolation, rather than assigning an interpolated string to a `String` instance, assign it to a `FormattableString`. You can then call its `FormattableString.ToString()` method produce a result string that reflects the conventions of the current culture, or you can call the `FormattableString.ToString(IFormatProvider)` method to produce a result string that reflects the conventions of a specified culture. You can also pass the formattable string to the static `FormattableString.Invariant` method to produce a result string that reflects the conventions of the invariant culture. The following example illustrates this approach. (The output from the example reflects a current culture of en-US.)

```
using System;
using System.Globalization;

class Program
{
    static void Main()
    {
        Decimal value = 126.03m;
        FormattableString amount = $"The amount is {value:C}";
        Console.WriteLine(amount.ToString());
        Console.WriteLine(amount.ToString(new CultureInfo("fr-FR")));
        Console.WriteLine(FormattableString.Invariant(amount));
    }
}
// The example displays the following output:
//      The amount is $126.03
//      The amount is 126,03 €
//      The amount is 126.03
```

```
Imports System.Globalization

Module Program
    Sub Main()
        Dim value As Decimal = 126.03
        Dim amount As FormattableString = $"The amount is {value:C}"
        Console.WriteLine(amount.ToString())
        Console.WriteLine(amount.ToString(new CultureInfo("fr-FR")))
        Console.WriteLine(FormattableString.Invariant(amount))
    End Sub
End Module
' The example displays the following output:
'      The amount is $126.03
'      The amount is 126,03 €
'      The amount is 126.03
```

Persist formatted data

You can persist non-string data either as binary data or as formatted data. If you choose to save it as formatted data, you should call a formatting method overload that includes a `provider` parameter and pass it the `CultureInfo.InvariantCulture` property. The invariant culture provides a consistent format for formatted data that is independent of culture and machine. In contrast, persisting data that is formatted by using cultures other than the invariant culture has a number of limitations:

- The data is likely to be unusable if it is retrieved on a system that has a different culture, or if the user of the current system changes the current culture and tries to retrieve the data.

- The properties of a culture on a specific computer can differ from standard values. At any time, a user can customize culture-sensitive display settings. Because of this, formatted data that is saved on a system may not be readable after the user customizes cultural settings. The portability of formatted data across computers is likely to be even more limited.
- International, regional, or national standards that govern the formatting of numbers or dates and times change over time, and these changes are incorporated into Windows operating system updates. When formatting conventions change, data that was formatted by using the previous conventions may become unreadable.

The following example illustrates the limited portability that results from using culture-sensitive formatting to persist data. The example saves an array of date and time values to a file. These are formatted by using the conventions of the English (United States) culture. After the application changes the current culture to French (Switzerland), it tries to read the saved values by using the formatting conventions of the current culture. The attempt to read two of the data items throws a [FormatException](#) exception, and the array of dates now contains two incorrect elements that are equal to [MinValue](#).

```

using System;
using System.Globalization;
using System.IO;
using System.Text;
using System.Threading;

public class Example
{
    private static string filename = @"\dates.dat";

    public static void Main()
    {
        DateTime[] dates = { new DateTime(1758, 5, 6, 21, 26, 0),
                            new DateTime(1818, 5, 5, 7, 19, 0),
                            new DateTime(1870, 4, 22, 23, 54, 0),
                            new DateTime(1890, 9, 8, 6, 47, 0),
                            new DateTime(1905, 2, 18, 15, 12, 0) };

        // Write the data to a file using the current culture.
        WriteData(dates);

        // Change the current culture.
        Thread.CurrentThread.CurrentCulture = CultureInfo.CreateSpecificCulture("fr-CH");
        // Read the data using the current culture.
        DateTime[] newDates = ReadData();
        foreach (var newDate in newDates)
            Console.WriteLine(newDate.ToString("g"));
    }

    private static void WriteData(DateTime[] dates)
    {
        StreamWriter sw = new StreamWriter(filename, false, Encoding.UTF8);
        for (int ctr = 0; ctr < dates.Length; ctr++) {
            sw.Write("{0}", dates[ctr].ToString("g", CultureInfo.CurrentCulture));
            if (ctr < dates.Length - 1) sw.Write("|");
        }
        sw.Close();
    }

    private static DateTime[] ReadData()
    {
        bool exceptionOccurred = false;

        // Read file contents as a single string, then split it.
        StreamReader sr = new StreamReader(filename, Encoding.UTF8);
        string output = sr.ReadToEnd();
        sr.Close();

        string[] values = output.Split( new char[] { '|' } );
        DateTime[] newDates = new DateTime[values.Length];
    }
}

```

```
DateTime[] newDates = new DateTime[values.Length];
for (int ctr = 0; ctr < values.Length; ctr++) {
    try {
        newDates[ctr] = DateTime.Parse(values[ctr], CultureInfo.CurrentCulture);
    }
    catch (FormatException) {
        Console.WriteLine("Failed to parse {0}", values[ctr]);
        exceptionOccurred = true;
    }
}
if (exceptionOccurred) Console.WriteLine();
return newDates;
}

// The example displays the following output:
//      Failed to parse 4/22/1870 11:54 PM
//      Failed to parse 2/18/1905 3:12 PM
//
//      05.06.1758 21:26
//      05.05.1818 07:19
//      01.01.0001 00:00
//      09.08.1890 06:47
//      01.01.0001 00:00
//      01.01.0001 00:00
```

```

Imports System.Globalization
Imports System.IO
Imports System.Text
Imports System.Threading

Module Example
    Private filename As String = ".\dates.dat"

    Public Sub Main()
        Dim dates() As Date = {[#5/6/1758 9:26PM#, #5/5/1818 7:19AM#, _
                               #4/22/1870 11:54PM#, #9/8/1890 6:47AM#, _ 
                               #2/18/1905 3:12PM#]}
        ' Write the data to a file using the current culture.
        WriteData(dates)
        ' Change the current culture.
        Thread.CurrentThread.CurrentCulture = CultureInfo.CreateSpecificCulture("fr-CH")
        ' Read the data using the current culture.
        Dim newDates() As Date = ReadData()
        For Each newDate In newDates
            Console.WriteLine(newDate.ToString("g"))
        Next
    End Sub

    Private Sub WriteData(dates() As Date)
        Dim sw As New StreamWriter(filename, False, Encoding.UTF8)
        For ctr As Integer = 0 To dates.Length - 1
            sw.WriteLine("{0}", dates(ctr).ToString("g", CultureInfo.CurrentCulture))
            If ctr < dates.Length - 1 Then sw.Write("|")
        Next
        sw.Close()
    End Sub

    Private Function ReadData() As Date()
        Dim exceptionOccurred As Boolean = False

        ' Read file contents as a single string, then split it.
        Dim sr As New StreamReader(filename, Encoding.UTF8)
        Dim output As String = sr.ReadToEnd()
        sr.Close()

        Dim values() As String = output.Split("|c")
        Dim newDates(values.Length - 1) As Date
        For ctr As Integer = 0 To values.Length - 1
            Try
                newDates(ctr) = DateTime.Parse(values(ctr), CultureInfo.CurrentCulture)
            Catch e As FormatException
                Console.WriteLine("Failed to parse {0}", values(ctr))
                exceptionOccurred = True
            End Try
        Next
        If exceptionOccurred Then Console.WriteLine()
        Return newDates
    End Function
End Module
' The example displays the following output:
'     Failed to parse 4/22/1870 11:54 PM
'     Failed to parse 2/18/1905 3:12 PM
'
'     05.06.1758 21:26
'     05.05.1818 07:19
'     01.01.0001 00:00
'     09.08.1890 06:47
'     01.01.0001 00:00
'     01.01.0001 00:00
'

```

However, if you replace the `CultureInfo.CurrentCulture` property with `CultureInfo.InvariantCulture` in the calls to `DateTime.ToString(String, IFormatProvider)` and `DateTime.Parse(String, IFormatProvider)`, the persisted date and time data is successfully restored, as the following output shows:

```
06.05.1758 21:26  
05.05.1818 07:19  
22.04.1870 23:54  
08.09.1890 06:47  
18.02.1905 15:12
```

Behavior changes when comparing strings on .NET 5+

9/20/2022 • 12 minutes to read • [Edit Online](#)

.NET 5 introduces a runtime behavioral change where globalization APIs [use ICU by default](#) across all supported platforms. This is a departure from earlier versions of .NET Core and from .NET Framework, which utilize the operating system's national language support (NLS) functionality when running on Windows. For more information on these changes, including compatibility switches that can revert the behavior change, see [.NET globalization and ICU](#).

Reason for change

This change was introduced to unify .NET's globalization behavior across all supported operating systems. It also provides the ability for applications to bundle their own globalization libraries rather than depend on the OS's built-in libraries. For more information, see [the breaking change notification](#).

Behavioral differences

If you use functions like `string.IndexOf(string)` without calling the overload that takes a [StringComparison](#) argument, you might intend to perform an *ordinal* search, but instead you inadvertently take a dependency on culture-specific behavior. Since NLS and ICU implement different logic in their linguistic comparers, the results of methods like `string.IndexOf(string)` can return unexpected values.

This can manifest itself even in places where you aren't always expecting globalization facilities to be active. For example, the following code can produce a different answer depending on the current runtime.

```
const string greeting = "Hel\0lo";
Console.WriteLine($"{greeting.IndexOf("\0")}");

// The snippet prints:
//
// '3' when running on .NET Core 2.x - 3.x (Windows)
// '0' when running on .NET 5 or later (Windows)
// '0' when running on .NET Core 2.x - 3.x or .NET 5 (non-Windows)
// '3' when running on .NET Core 2.x or .NET 5+ (in invariant mode)

string s = "Hello\r\nworld!";
int idx = s.IndexOf("\n");
Console.WriteLine(idx);

// The snippet prints:
//
// '6' when running on .NET Core 3.1
// '-1' when running on .NET 5 or .NET Core 3.1 (non-Windows OS)
// '-1' when running on .NET 5 (Windows 10 May 2019 Update or later)
// '6' when running on .NET 6+ (all Windows and non-Windows OSs)
```

For more information, see [Globalization APIs use ICU libraries on Windows](#).

Guard against unexpected behavior

This section provides two options for dealing with unexpected behavior changes in .NET 5.

Enable code analyzers

Code analyzers can detect possibly buggy call sites. To help guard against any surprising behaviors, we recommend enabling .NET compiler platform (Roslyn) analyzers in your project. The analyzers help flag code that might inadvertently be using a linguistic comparer when an ordinal comparer was likely intended. The following rules should help flag these issues:

- [CA1307: Specify StringComparison for clarity](#)
- [CA1309: Use ordinal StringComparison](#)
- [CA1310: Specify StringComparison for correctness](#)

These specific rules aren't enabled by default. To enable them and show any violations as build errors, set the following properties in your project file:

```
<PropertyGroup>
  <AnalysisMode>All</AnalysisMode>
  <WarningsAsErrors>$WarningsAsErrors;CA1307;CA1309;CA1310</WarningsAsErrors>
</PropertyGroup>
```

The following snippet shows examples of code that produces the relevant code analyzer warnings or errors.

```
//
// Potentially incorrect code - answer might vary based on locale.
//
string s = GetString();
// Produces analyzer warning CA1310 for string; CA1307 matches on char ','
int idx = s.IndexOf(",");
Console.WriteLine(idx);

//
// Corrected code - matches the literal substring ",".
//
string s = GetString();
int idx = s.IndexOf(",", StringComparison.Ordinal);
Console.WriteLine(idx);

//
// Corrected code (alternative) - searches for the literal ',' character.
//
string s = GetString();
int idx = s.IndexOf(',');
Console.WriteLine(idx);
```

Similarly, when instantiating a sorted collection of strings or sorting an existing string-based collection, specify an explicit comparer.

```
//
// Potentially incorrect code - behavior might vary based on locale.
//
SortedSet<string> mySet = new SortedSet<string>();
List<string> list = GetListOfStrings();
list.Sort();

//
// Corrected code - uses ordinal sorting; doesn't vary by locale.
//
SortedSet<string> mySet = new SortedSet<string>(StringComparer.OrdinalIgnoreCase);
List<string> list = GetListOfStrings();
list.Sort(StringComparer.OrdinalIgnoreCase);
```

Revert back to NLS behaviors

To revert .NET 5+ applications back to older NLS behaviors when running on Windows, follow the steps in [.NET Globalization and ICU](#). This application-wide compatibility switch must be set at the application level. Individual libraries cannot opt-in or opt-out of this behavior.

TIP

We strongly recommend you enable the [CA1307](#), [CA1309](#), and [CA1310](#) code analysis rules to help improve code hygiene and discover any existing latent bugs. For more information, see [Enable code analyzers](#).

Affected APIs

Most .NET applications won't encounter any unexpected behaviors due to the changes in .NET 5. However, due to the number of affected APIs and how foundational these APIs are to the wider .NET ecosystem, you should be aware of the potential for .NET 5 to introduce unwanted behaviors or to expose latent bugs that already exist in your application.

The affected APIs include:

- [System.String.Compare](#)
- [System.String.EndsWith](#)
- [System.String.IndexOf](#)
- [System.String.StartsWith](#)
- [System.String.ToLower](#)
- [System.String.ToLowerInvariant](#)
- [System.String.ToUpper](#)
- [System.String.ToUpperInvariant](#)
- [System.Globalization.TextInfo](#) (most members)
- [System.Globalization.CompareInfo](#) (most members)
- [System.Array.Sort](#) (when sorting arrays of strings)
- [System.Collections.Generic.List<T>.Sort\(\)](#) (when the list elements are strings)
- [System.Collections.Generic.SortedDictionary< TKey, TValue >](#) (when the keys are strings)
- [System.Collections.Generic.SortedList< TKey, TValue >](#) (when the keys are strings)
- [System.Collections.Generic.SortedSet< T >](#) (when the set contains strings)

NOTE

This is not an exhaustive list of affected APIs.

All of the above APIs use *linguistic* string searching and comparison using the thread's [current culture](#), by default. The differences between *linguistic* and *ordinal* search and comparison are called out in the [Ordinal vs. linguistic search and comparison](#).

Because ICU implements linguistic string comparisons differently from NLS, Windows-based applications that upgrade to .NET 5 from an earlier version of .NET Core or .NET Framework and that call one of the affected APIs may notice that the APIs begin exhibiting different behaviors.

Exceptions

- If an API accepts an explicit `StringComparison` or `CultureInfo` parameter, that parameter overrides the API's default behavior.
- `System.String` members where the first parameter is of type `char` (for example, `String.IndexOf(Char)`) use

ordinal searching, unless the caller passes an explicit `StringComparison` argument that specifies `CurrentCulture[IgnoreCase]` or `InvariantCulture[IgnoreCase]` .

For a more detailed analysis of the default behavior of each `String` API, see the [Default search and comparison types](#) section.

Ordinal vs. linguistic search and comparison

Ordinal (also known as *non-linguistic*) search and comparison decomposes a string into its individual `char` elements and performs a char-by-char search or comparison. For example, the strings `"dog"` and `"dog"` compare as *equal* under an `Ordinal` comparer, since the two strings consist of the exact same sequence of chars. However, `"dog"` and `"Dog"` compare as *not equal* under an `Ordinal` comparer, because they don't consist of the exact same sequence of chars. That is, uppercase `'D'` 's code point `U+0044` occurs before lowercase `'d'` 's code point `U+0064` , resulting in `"dog"` sorting before `"Dog"` .

An `ordinalIgnoreCase` comparer still operates on a char-by-char basis, but it eliminates case differences while performing the operation. Under an `OrdinalIgnoreCase` comparer, the char pairs `'d'` and `'D'` compare as *equal*, as do the char pairs `'á'` and `'Á'` . But the unaccented char `'a'` compares as *not equal* to the accented char `'á'` .

Some examples of this are provided in the following table:

STRING 1	STRING 2	ORDINAL COMPARISON	ORDINALIGNORECASE COMPARISON
<code> "dog" </code>	<code> "dog" </code>	equal	equal
<code> "dog" </code>	<code> "Dog" </code>	not equal	equal
<code> "resume" </code>	<code> "résumé" </code>	not equal	not equal

Unicode also allows strings to have several different in-memory representations. For example, an e-acute (é) can be represented in two possible ways:

- A single literal `'é'` character (also written as `'\u00E9'`).
- A literal unaccented `'e'` character followed by a combining accent modifier character `'\u0301'` .

This means that the following *four* strings all display as `"résumé"` , even though their constituent pieces are different. The strings use a combination of literal `'é'` characters or literal unaccented `'e'` characters plus the combining accent modifier `'\u0301'` .

- `"r\u00E9sum\u00E9"`
- `"r\u00E9sume\u0301"`
- `"re\u0301sum\u00E9"`
- `"re\u0301sume\u0301"`

Under an ordinal comparer, none of these strings compare as equal to each other. This is because they all contain different underlying char sequences, even though when they're rendered to the screen, they all look the same.

When performing a `string.IndexOf(..., StringComparison.Ordinal)` operation, the runtime looks for an exact substring match. The results are as follows.

```

Console.WriteLine("resume".IndexOf("e", StringComparison.OrdinalIgnoreCase)); // prints '1'
Console.WriteLine("r\u00E9sum\u00E9".IndexOf("e", StringComparison.OrdinalIgnoreCase)); // prints '-1'
Console.WriteLine("r\u00E9sum\u0301".IndexOf("e", StringComparison.OrdinalIgnoreCase)); // prints '5'
Console.WriteLine("r\u00E9sum\u00E9".IndexOf("e", StringComparison.OrdinalIgnoreCase)); // prints '1'
Console.WriteLine("r\u00E9sum\u0301".IndexOf("e", StringComparison.OrdinalIgnoreCase)); // prints '1'
Console.WriteLine("r\u00E9sum\u00E9".IndexOf("E", StringComparison.OrdinalIgnoreCase)); // prints '1'
Console.WriteLine("r\u00E9sum\u0301".IndexOf("E", StringComparison.OrdinalIgnoreCase)); // prints '-1'
Console.WriteLine("r\u00E9sum\u0301".IndexOf("E", StringComparison.OrdinalIgnoreCase)); // prints '5'
Console.WriteLine("r\u00E9sum\u00E9".IndexOf("E", StringComparison.OrdinalIgnoreCase)); // prints '1'
Console.WriteLine("r\u00E9sum\u0301".IndexOf("E", StringComparison.OrdinalIgnoreCase)); // prints '1'

```

Ordinal search and comparison routines are never affected by the current thread's culture setting.

Linguistic search and comparison routines decompose a string into *collation elements* and perform searches or comparisons on these elements. There's not necessarily a 1:1 mapping between a string's characters and its constituent collation elements. For example, a string of length 2 may consist of only a single collation element. When two strings are compared in a linguistic-aware fashion, the comparer checks whether the two strings' collation elements have the same semantic meaning, even if the string's literal characters are different.

Consider again the string "résumé" and its four different representations. The following table shows each representation broken down into its collation elements.

STRING	AS COLLATION ELEMENTS
"r\u00E9sum\u00E9"	"r" + "\u00E9" + "s" + "u" + "m" + "\u00E9"
"r\u00E9sum\u0301"	"r" + "\u00E9" + "s" + "u" + "m" + "e\u0301"
"r\u00E9sum\u00E9"	"r" + "e\u0301" + "s" + "u" + "m" + "\u00E9"
"r\u00E9sum\u0301"	"r" + "e\u0301" + "s" + "u" + "m" + "e\u0301"

A collation element corresponds loosely to what readers would think of as a single character or cluster of characters. It's conceptually similar to a [grapheme cluster](#) but encompasses a somewhat larger umbrella.

Under a linguistic comparer, exact matches aren't necessary. Collation elements are instead compared based on their semantic meaning. For example, a linguistic comparer treats the substrings "\u00E9" and "e\u0301" as equal since they both semantically mean "a lowercase e with an acute accent modifier." This allows the `IndexOf` method to match the substring "e\u0301" within a larger string that contains the semantically equivalent substring "\u00E9", as shown in the following code sample.

```

Console.WriteLine("r\u00E9sum\u00E9".IndexOf("e")); // prints '-1' (not found)
Console.WriteLine("r\u00E9sum\u00E9".IndexOf("\u00E9")); // prints '1'
Console.WriteLine("\u00E9".IndexOf("e\u0301")); // prints '0'

```

As a consequence of this, two strings of different lengths may compare as equal if a linguistic comparison is used. Callers should take care not to special-case logic that deals with string length in such scenarios.

Culture-aware search and comparison routines are a special form of linguistic search and comparison routines. Under a culture-aware comparer, the concept of a collation element is extended to include information specific to the specified culture.

For example, [in the Hungarian alphabet](#), when the two characters <dz> appear back-to-back, they are considered their own unique letter distinct from either <d> or <z>. This means that when <dz> is seen in a string, a Hungarian culture-aware comparer treats it as a single collation element.

STRING	AS COLLATION ELEMENTS	REMARKS
"endz"	"e" + "n" + "d" + "z"	(using a standard linguistic comparer)
"endz"	"e" + "n" + "dz"	(using a Hungarian culture-aware comparer)

When using a Hungarian culture-aware comparer, this means that the string "endz" *does not* end with the substring "z", as <\dz> and <\z> are considered collation elements with different semantic meaning.

```
// Set thread culture to Hungarian
CultureInfo.CurrentCulture = CultureInfo.GetCultureInfo("hu-HU");
Console.WriteLine("endz".EndsWith("z")); // Prints 'False'

// Set thread culture to invariant culture
CultureInfo.CurrentCulture = CultureInfo.InvariantCulture;
Console.WriteLine("endz".EndsWith("z")); // Prints 'True'
```

NOTE

- Behavior: Linguistic and culture-aware comparers can undergo behavioral adjustments from time to time. Both ICU and the older Windows NLS facility are updated to account for how world languages change. For more information, see the blog post [Locale \(culture\) data churn](#). The *Ordinal* comparer's behavior will never change since it performs exact bitwise searching and comparison. However, the *OrdinalIgnoreCase* comparer's behavior may change as Unicode grows to encompass more character sets and corrects omissions in existing casing data.
- Usage: The comparers `StringComparison.InvariantCulture` and `StringComparison.InvariantCultureIgnoreCase` are linguistic comparers that are not culture-aware. That is, these comparers understand concepts such as the accented character é having multiple possible underlying representations, and that all such representations should be treated equal. But non-culture-aware linguistic comparers won't contain special handling for <\dz> as distinct from <\d> or <\z>, as shown above. They also won't special-case characters like the German Eszett (ß).

.NET also offers the *invariant globalization mode*. This opt-in mode disables code paths that deal with linguistic search and comparison routines. In this mode, all operations use *Ordinal* or *OrdinalIgnoreCase* behaviors, regardless of what `CultureInfo` or `StringComparison` argument the caller provides. For more information, see [Runtime configuration options for globalization](#) and [.NET Core Globalization Invariant Mode](#).

For more information, see [Best practices for comparing strings in .NET](#).

Security implications

If your app uses an affected API for filtering, we recommend enabling the CA1307 and CA1309 code analysis rules to help locate places where a linguistic search may have inadvertently been used instead of an ordinal search. Code patterns like the following may be susceptible to security exploits.

```
//
// THIS SAMPLE CODE IS INCORRECT.
// DO NOT USE IT IN PRODUCTION.
//
public bool ContainsHtmlSensitiveCharacters(string input)
{
    if (input.IndexOf("<") >= 0) { return true; }
    if (input.IndexOf("&") >= 0) { return true; }
    return false;
}
```

Because the `string.IndexOf(string)` method uses a linguistic search by default, it's possible for a string to contain a literal '`<`' or '`&`' character and for the `string.IndexOf(string)` routine to return `-1`, indicating that the search substring was not found. Code analysis rules CA1307 and CA1309 flag such call sites and alert the developer that there's a potential problem.

Default search and comparison types

The following table lists the default search and comparison types for various string and string-like APIs. If the caller provides an explicit `CultureInfo` or `StringComparison` parameter, that parameter will be honored over any default.

API	DEFAULT BEHAVIOR	REMARKS
<code>string.Compare</code>	CurrentCulture	
<code>string.CompareTo</code>	CurrentCulture	
<code>string.Contains</code>	Ordinal	
<code>string.EndsWith</code>	Ordinal	(when the first parameter is a <code>char</code>)
<code>string.EndsWith</code>	CurrentCulture	(when the first parameter is a <code>string</code>)
<code>string.Equals</code>	Ordinal	
<code>string.GetHashCode</code>	Ordinal	
<code>string.IndexOf</code>	Ordinal	(when the first parameter is a <code>char</code>)
<code>string.IndexOf</code>	CurrentCulture	(when the first parameter is a <code>string</code>)
<code>string.IndexOfAny</code>	Ordinal	
<code>string.LastIndexOf</code>	Ordinal	(when the first parameter is a <code>char</code>)
<code>string.LastIndexOf</code>	CurrentCulture	(when the first parameter is a <code>string</code>)
<code>string.LastIndexOfAny</code>	Ordinal	
<code>string.Replace</code>	Ordinal	
<code>string.Split</code>	Ordinal	
<code>string.StartsWith</code>	Ordinal	(when the first parameter is a <code>char</code>)
<code>string.StartsWith</code>	CurrentCulture	(when the first parameter is a <code>string</code>)
<code>string.ToLower</code>	CurrentCulture	

API	DEFAULT BEHAVIOR	REMARKS
<code>string.ToLowerInvariant</code>	InvariantCulture	
<code>string.ToUpper</code>	CurrentCulture	
<code>string.ToUpperInvariant</code>	InvariantCulture	
<code>string.Trim</code>	Ordinal	
<code>string.TrimEnd</code>	Ordinal	
<code>string.TrimStart</code>	Ordinal	
<code>string == string</code>	Ordinal	
<code>string != string</code>	Ordinal	

Unlike `string` APIs, all `MemoryExtensions` APIs perform *Ordinal* searches and comparisons by default, with the following exceptions.

API	DEFAULT BEHAVIOR	REMARKS
<code>MemoryExtensions.ToLower</code>	CurrentCulture	(when passed a null <code>CultureInfo</code> argument)
<code>MemoryExtensions.ToLowerInvariant</code>	InvariantCulture	
<code>MemoryExtensions.ToUpper</code>	CurrentCulture	(when passed a null <code>CultureInfo</code> argument)
<code>MemoryExtensions.ToUpperInvariant</code>	InvariantCulture	

A consequence is that when converting code from consuming `string` to consuming `ReadOnlySpan<char>`, behavioral changes may be introduced inadvertently. An example of this follows.

```
string str = GetString();
if (str.StartsWith("Hello")) { /* do something */ } // this is a CULTURE-AWARE (linguistic) comparison

ReadOnlySpan<char> span = s.AsSpan();
if (span.StartsWith("Hello")) { /* do something */ } // this is an ORDINAL (non-linguistic) comparison
```

The recommended way to address this is to pass an explicit `StringComparison` parameter to these APIs. The code analysis rules CA1307 and CA1309 can assist with this.

```
string str = GetString();
if (str.StartsWith("Hello", StringComparison.OrdinalIgnoreCase)) { /* do something */ } // ordinal comparison

ReadOnlySpan<char> span = s.AsSpan();
if (span.StartsWith("Hello", StringComparison.OrdinalIgnoreCase)) { /* do something */ } // ordinal comparison
```

See also

- [Globalization breaking changes](#)
- [Best practices for comparing strings in .NET](#)
- [How to compare strings in C#](#)
- [.NET globalization and ICU](#)
- [Ordinal vs. culture-sensitive string operations](#)
- [Overview of .NET source code analysis](#)

Basic string operations in .NET

9/20/2022 • 2 minutes to read • [Edit Online](#)

Applications often respond to users by constructing messages based on user input. For example, it is not uncommon for websites to respond to a newly logged-on user with a specialized greeting that includes the user's name.

Several methods in the [System.String](#) and [System.Text.StringBuilder](#) classes allow you to dynamically construct custom strings to display in your user interface. These methods also help you perform a number of basic string operations like creating new strings from arrays of bytes, comparing the values of strings, and modifying existing strings.

Related sections

[Type Conversion in .NET](#)

Describes how to convert one type into another type.

[Formatting Types](#)

Describes how to format strings using format specifiers.

Creating New Strings in .NET

9/20/2022 • 4 minutes to read • [Edit Online](#)

.NET allows strings to be created using simple assignment, and also overloads a class constructor to support string creation using a number of different parameters. .NET also provides several methods in the [System.String](#) class that create new string objects by combining several strings, arrays of strings, or objects.

Creating Strings Using Assignment

The easiest way to create a new [String](#) object is simply to assign a string literal to a [String](#) object.

Creating Strings Using a Class Constructor

You can use overloads of the [String](#) class constructor to create strings from character arrays. You can also create a new string by duplicating a particular character a specified number of times.

Methods that Return Strings

The following table lists several useful methods that return new string objects.

METHOD NAME	USE
String.Format	Builds a formatted string from a set of input objects.
String.Concat	Builds strings from two or more strings.
String.Join	Builds a new string by combining an array of strings.
String.Insert	Builds a new string by inserting a string into the specified index of an existing string.
String.CopyTo	Copies specified characters in a string into a specified position in an array of characters.

Format

You can use the [String.Format](#) method to create formatted strings and concatenate strings representing multiple objects. This method automatically converts any passed object into a string. For example, if your application must display an [Int32](#) value and a [DateTime](#) value to the user, you can easily construct a string to represent these values using the [Format](#) method. For information about formatting conventions used with this method, see the section on [composite formatting](#).

The following example uses the [Format](#) method to create a string that uses an integer variable.

```

int numberOffleas = 12;
string miscInfo = String.Format("Your dog has {0} fleas. " +
                                "It is time to get a flea collar. " +
                                "The current universal date is: {1:u}.",
                                numberOffleas, DateTime.Now);

Console.WriteLine(miscInfo);
// The example displays the following output:
//      Your dog has 12 fleas. It is time to get a flea collar.
//      The current universal date is: 2008-03-28 13:31:40Z.

```

```

Dim numberOffleas As Integer = 12
Dim miscInfo As String = String.Format("Your dog has {0} fleas. " & _
                                         "It is time to get a flea collar. " & _
                                         "The current universal date is: {1:u}.",
                                         numberOffleas, Date.Now)

Console.WriteLine(miscInfo)
' The example displays the following output:
'      Your dog has 12 fleas. It is time to get a flea collar.
'      The current universal date is: 2008-03-28 13:31:40Z.

```

In this example, `DateTime.Now` displays the current date and time in a manner specified by the culture associated with the current thread.

Concat

The `String.Concat` method can be used to easily create a new string object from two or more existing objects. It provides a language-independent way to concatenate strings. This method accepts any class that derives from `System.Object`. The following example creates a string from two existing string objects and a separating character.

```

string helloString1 = "Hello";
string helloString2 = "World!";
Console.WriteLine(String.Concat(helloString1, ' ', helloString2));
// The example displays the following output:
//      Hello World!

```

```

Dim helloString1 As String = "Hello"
Dim helloString2 As String = "World!"
Console.WriteLine(String.Concat(helloString1, " "c, helloString2))
' The example displays the following output:
'      Hello World!

```

Join

The `String.Join` method creates a new string from an array of strings and a separator string. This method is useful if you want to concatenate multiple strings together, making a list perhaps separated by a comma.

The following example uses a space to bind a string array.

```

string[] words = {"Hello", "and", "welcome", "to", "my" , "world!"};
Console.WriteLine(String.Join(" ", words));
// The example displays the following output:
//      Hello and welcome to my world!

```

```
Dim words() As String = {"Hello", "and", "welcome", "to", "my", "world!"}
Console.WriteLine(String.Join(" ", words))
' The example displays the following output:
'     Hello and welcome to my world!
```

Insert

The **String.Insert** method creates a new string by inserting a string into a specified position in another string.

This method uses a zero-based index. The following example inserts a string into the fifth index position of

`MyString` and creates a new string with this value.

```
string sentence = "Once a time.";
Console.WriteLine(sentence.Insert(4, " upon"));
// The example displays the following output:
//     Once upon a time.
```

```
Dim sentence As String = "Once a time."
Console.WriteLine(sentence.Insert(4, " upon"))
' The example displays the following output:
'     Once upon a time.
```

CopyTo

The **String.CopyTo** method copies portions of a string into an array of characters. You can specify both the beginning index of the string and the number of characters to be copied. This method takes the source index, an array of characters, the destination index, and the number of characters to copy. All indexes are zero-based.

The following example uses the **CopyTo** method to copy the characters of the word "Hello" from a string object to the first index position of an array of characters.

```
string greeting = "Hello World!";
char[] charArray = {'W','h','e','r','e'};
Console.WriteLine("The original character array: {0}", new string(charArray));
greeting.CopyTo(0, charArray, 0, 5);
Console.WriteLine("The new character array: {0}", new string(charArray));
// The example displays the following output:
//     The original character array: Where
//     The new character array: Hello
```

```
Dim greeting As String = "Hello World!"
Dim charArray() As Char = {"W"c, "h"c, "e"c, "r"c, "e"c}
Console.WriteLine("The original character array: {0}", New String(charArray))
greeting.CopyTo(0, charArray, 0, 5)
Console.WriteLine("The new character array: {0}", New String(charArray))
' The example displays the following output:
'     The original character array: Where
'     The new character array: Hello
```

See also

- [Basic String Operations](#)
- [Composite Formatting](#)

Trimming and Removing Characters from Strings in .NET

9/20/2022 • 5 minutes to read • [Edit Online](#)

If you are parsing a sentence into individual words, you might end up with words that have blank spaces (also called white spaces) on either end of the word. In this situation, you can use one of the trim methods in the **System.String** class to remove any number of spaces or other characters from a specified position in the string. The following table describes the available trim methods.

METHOD NAME	USE
String.Trim	Removes white spaces or characters specified in an array of characters from the beginning and end of a string.
String.TrimEnd	Removes characters specified in an array of characters from the end of a string.
String.TrimStart	Removes characters specified in an array of characters from the beginning of a string.
String.Remove	Removes a specified number of characters from a specified index position in a string.

Trim

You can easily remove white spaces from both ends of a string by using the [String.Trim](#) method, as shown in the following example.

```
String^ MyString = " Big   ";
Console::WriteLine("Hello{0}World!", MyString);
String^ TrimString = MyString->Trim();
Console::WriteLine("Hello{0}World!", TrimString);
// The example displays the following output:
//      Hello Big   World!
//      HelloBigWorld!
```

```
string MyString = " Big   ";
Console.WriteLine("Hello{0}World!", MyString);
string TrimString = MyString.Trim();
Console.WriteLine("Hello{0}World!", TrimString);
// The example displays the following output:
//      Hello Big   World!
//      HelloBigWorld!
```

```

Dim MyString As String = " Big "
Console.WriteLine("Hello{0}World!", MyString)
Dim TrimString As String = MyString.Trim()
Console.WriteLine("Hello{0}World!", TrimString)
' The example displays the following output:
'     Hello Big  World!
'     HelloBigWorld!

```

You can also remove characters that you specify in a character array from the beginning and end of a string. The following example removes white-space characters, periods, and asterisks.

```

using System;

public class Example
{
    public static void Main()
    {
        String header = "* A Short String. *";
        Console.WriteLine(header);
        Console.WriteLine(header.Trim( new Char[] { ' ', '*', '.' } ));
    }
}
// The example displays the following output:
//      * A Short String. *
//      A Short String

```

```

Module Example
    Public Sub Main()
        Dim header As String = "* A Short String. *"
        Console.WriteLine(header)
        Console.WriteLine(header.Trim({" "c, "*"c, ".c"}))
    End Sub
End Module
' The example displays the following output:
'      * A Short String. *
'      A Short String

```

TrimEnd

The **String.TrimEnd** method removes characters from the end of a string, creating a new string object. An array of characters is passed to this method to specify the characters to be removed. The order of the elements in the character array does not affect the trim operation. The trim stops when a character not specified in the array is found.

The following example removes the last letters of a string using the **TrimEnd** method. In this example, the position of the `'r'` character and the `'w'` character are reversed to illustrate that the order of characters in the array does not matter. Notice that this code removes the last word of `MyString` plus part of the first.

```

String^ MyString = "Hello World!";
array<Char>^ MyChar = {'r','o','W','l','d','!', ' '};
String^ NewString = MyString->TrimEnd(MyChar);
Console::WriteLine(NewString);

```

```
string MyString = "Hello World!";
char[] MyChar = {'r','o','W','l','d','!', ' '};
string NewString = MyString.TrimEnd(MyChar);
Console.WriteLine(NewString);
```

```
Dim MyString As String = "Hello World!"
Dim MyChar() As Char = {"r", "o", "W", "l", "d", "!", " "}
Dim NewString As String = MyString.TrimEnd(MyChar)
Console.WriteLine(NewString)
```

This code displays `Hello` to the console.

The following example removes the last word of a string using the **TrimEnd** method. In this code, a comma follows the word `Hello` and, because the comma is not specified in the array of characters to trim, the trim ends at the comma.

```
String^ MyString = "Hello, World!";
array<Char>^ MyChar = {'r','o','W','l','d','!', ' '};
String^ NewString = MyString->TrimEnd(MyChar);
Console::WriteLine(NewString);
```

```
string MyString = "Hello, World!";
char[] MyChar = {'r','o','W','l','d','!', ' '};
string NewString = MyString.TrimEnd(MyChar);
Console.WriteLine(NewString);
```

```
Dim MyString As String = "Hello, World!"
Dim MyChar() As Char = {"r", "o", "W", "l", "d", "!", " "}
Dim NewString As String = MyString.TrimEnd(MyChar)
Console.WriteLine(NewString)
```

This code displays `Hello,` to the console.

TrimStart

The **String.TrimStart** method is similar to the **String.TrimEnd** method except that it creates a new string by removing characters from the beginning of an existing string object. An array of characters is passed to the **TrimStart** method to specify the characters to be removed. As with the **TrimEnd** method, the order of the elements in the character array does not affect the trim operation. The trim stops when a character not specified in the array is found.

The following example removes the first word of a string. In this example, the position of the `'l'` character and the `'H'` character are reversed to illustrate that the order of characters in the array does not matter.

```
String^ MyString = "Hello World!";
array<Char>^ MyChar = {'e', 'H','l','o',' ' };
String^ NewString = MyString->TrimStart(MyChar);
Console::WriteLine(NewString);
```

```
string MyString = "Hello World!";
char[] MyChar = {'e', 'H','l','o',' ' };
string NewString = MyString.TrimStart(MyChar);
Console.WriteLine(NewString);
```

```
Dim MyString As String = "Hello World!"
Dim MyChar() As Char = {"e", "H", "l", "o", " "}
Dim NewString As String = MyString.TrimStart(MyChar)
Console.WriteLine(NewString)
```

This code displays `World!` to the console.

Remove

The [String.Remove](#) method removes a specified number of characters that begin at a specified position in an existing string. This method assumes a zero-based index.

The following example removes ten characters from a string beginning at position five of a zero-based index of the string.

```
String^ MyString = "Hello Beautiful World!";
Console::WriteLine(MyString->Remove(5,10));
// The example displays the following output:
//      Hello World!
```

```
string MyString = "Hello Beautiful World!";
Console.WriteLine(MyString.Remove(5,10));
// The example displays the following output:
//      Hello World!
```

```
Dim MyString As String = "Hello Beautiful World!"
Console.WriteLine(MyString.Remove(5, 10))
' The example displays the following output:
'      Hello World!
```

Replace

You can also remove a specified character or substring from a string by calling the [String.Replace\(String, String\)](#) method and specifying an empty string ([String.Empty](#)) as the replacement. The following example removes all commas from a string.

```
using System;

public class Example
{
    public static void Main()
    {
        String phrase = "a cold, dark night";
        Console.WriteLine("Before: {0}", phrase);
        phrase = phrase.Replace(",", "");
        Console.WriteLine("After: {0}", phrase);
    }
}
// The example displays the following output:
//      Before: a cold, dark night
//      After: a cold dark night
```

```
Module Example
Public Sub Main()
    Dim phrase As String = "a cold, dark night"
    Console.WriteLine("Before: {0}", phrase)
    phrase = phrase.Replace(",", "")
    Console.WriteLine("After: {0}", phrase)
End Sub
End Module
' The example displays the following output:
'      Before: a cold, dark night
'      After: a cold dark night
```

See also

- [Basic String Operations](#)

Padding Strings in .NET

9/20/2022 • 2 minutes to read • [Edit Online](#)

Use one of the following [String](#) methods to create a new string that consists of an original string that is padded with leading or trailing characters to a specified total length. The padding character can be a space or a specified character. The resulting string appears to be either right-aligned or left-aligned. If the original string's length is already equal to or greater than the desired total length, the padding methods return the original string unchanged; for more information, see the **Returns** sections of the two overloads of the [String.PadLeft](#) and [String.PadRight](#) methods.

METHOD NAME	USE
String.PadLeft	Pads a string with leading characters to a specified total length.
String.PadRight	Pads a string with trailing characters to a specified total length.

PadLeft

The [String.PadLeft](#) method creates a new string by concatenating enough leading pad characters to an original string to achieve a specified total length. The [String.PadLeft\(Int32\)](#) method uses white space as the padding character and the [String.PadLeft\(Int32, Char\)](#) method enables you to specify your own padding character.

The following code example uses the [PadLeft](#) method to create a new string that is twenty characters long. The example displays "-----Hello World!" to the console.

```
String^ MyString = "Hello World!";
Console::WriteLine(MyString->PadLeft(20, '-'));
```

```
string MyString = "Hello World!";
Console.WriteLine(MyString.PadLeft(20, '-'));
```

```
Dim MyString As String = "Hello World!"
Console.WriteLine(MyString.PadLeft(20, "-c"))
```

PadRight

The [String.PadRight](#) method creates a new string by concatenating enough trailing pad characters to an original string to achieve a specified total length. The [String.PadRight\(Int32\)](#) method uses white space as the padding character and the [String.PadRight\(Int32, Char\)](#) method enables you to specify your own padding character.

The following code example uses the [PadRight](#) method to create a new string that is twenty characters long. The example displays "Hello World!-----" to the console.

```
String^ MyString = "Hello World!";
Console::WriteLine(MyString->PadRight(20, '-'));
```

```
string MyString = "Hello World!";
Console.WriteLine(MyString.PadRight(20, '-'));
```

```
Dim MyString As String = "Hello World!"
Console.WriteLine(MyString.PadRight(20, "-c"))
```

See also

- [Basic String Operations](#)

Compare strings in .NET

9/20/2022 • 7 minutes to read • [Edit Online](#)

.NET provides several methods to compare the values of strings. The following table lists and describes the value-comparison methods.

METHOD NAME	USE
<code>String.Compare</code>	Compares the values of two strings. Returns an integer value.
<code>String.CompareOrdinal</code>	Compares two strings without regard to local culture. Returns an integer value.
<code>String.CompareTo</code>	Compares the current string object to another string. Returns an integer value.
<code>String.StartsWith</code>	Determines whether a string begins with the string passed. Returns a Boolean value.
<code>String.EndsWith</code>	Determines whether a string ends with the string passed. Returns a Boolean value.
<code>String.Contains</code>	Determines whether a character or string occurs within another string. Returns a Boolean value.
<code>String.Equals</code>	Determines whether two strings are the same. Returns a Boolean value.
<code>String.IndexOf</code>	Returns the index position of a character or string, starting from the beginning of the string you are examining. Returns an integer value.
<code>String.LastIndexOf</code>	Returns the index position of a character or string, starting from the end of the string you are examining. Returns an integer value.

Compare method

The static `String.Compare` method provides a thorough way of comparing two strings. This method is culturally aware. You can use this function to compare two strings or substrings of two strings. Additionally, overloads are provided that regard or disregard case and cultural variance. The following table shows the three integer values that this method might return.

RETURN VALUE	CONDITION
A negative integer	The first string precedes the second string in the sort order. -or- The first string is <code>null</code> .

RETURN VALUE	CONDITION
0	The first string and the second string are equal. -or- Both strings are <code>null</code> .
A positive integer -or- 1	The first string follows the second string in the sort order. -or- The second string is <code>null</code> .

IMPORTANT

The [String.Compare](#) method is primarily intended for use when ordering or sorting strings. You should not use the [String.Compare](#) method to test for equality (that is, to explicitly look for a return value of 0 with no regard for whether one string is less than or greater than the other). Instead, to determine whether two strings are equal, use the [String.Equals\(String, String, StringComparison\)](#) method.

The following example uses the [String.Compare](#) method to determine the relative values of two strings.

```
String^ string1 = "Hello World!";
Console::WriteLine(String::Compare(string1, "Hello World?"));
```

```
string string1 = "Hello World!";
Console.WriteLine(String.Compare(string1, "Hello World?"));
```

```
Dim string1 As String = "Hello World!"
Console.WriteLine(String.Compare(string1, "Hello World?"))
```

This example displays `-1` to the console.

The preceding example is culture-sensitive by default. To perform a culture-insensitive string comparison, use an overload of the [String.Compare](#) method that allows you to specify the culture to use by supplying a *culture* parameter. For an example that demonstrates how to use the [String.Compare](#) method to perform a culture-insensitive comparison, see [Culture-insensitive string comparisons](#).

CompareOrdinal method

The [String.CompareOrdinal](#) method compares two string objects without considering the local culture. The return values of this method are identical to the values returned by the `Compare` method in the previous table.

IMPORTANT

The [String.CompareOrdinal](#) method is primarily intended for use when ordering or sorting strings. You should not use the [String.CompareOrdinal](#) method to test for equality (that is, to explicitly look for a return value of 0 with no regard for whether one string is less than or greater than the other). Instead, to determine whether two strings are equal, use the [String.Equals\(String, String, StringComparison\)](#) method.

The following example uses the `CompareOrdinal` method to compare the values of two strings.

```
String^ string1 = "Hello World!";
Console::WriteLine(String::CompareOrdinal(string1, "hello world!"));
```

```
string string1 = "Hello World!";
Console.WriteLine(String.CompareOrdinal(string1, "hello world!"));
```

```
Dim string1 As String = "Hello World!"
Console.WriteLine(String.CompareOrdinal(string1, "hello world!"))
```

This example displays `-32` to the console.

CompareTo method

The [String.CompareTo](#) method compares the string that the current string object encapsulates to another string or object. The return values of this method are identical to the values returned by the [String.Compare](#) method in the previous table.

IMPORTANT

The [String.CompareTo](#) method is primarily intended for use when ordering or sorting strings. You should not use the [String.CompareTo](#) method to test for equality (that is, to explicitly look for a return value of 0 with no regard for whether one string is less than or greater than the other). Instead, to determine whether two strings are equal, use the [String.Equals\(String, String, StringComparison\)](#) method.

The following example uses the [String.CompareTo](#) method to compare the `string1` object to the `string2` object.

```
String^ string1 = "Hello World";
String^ string2 = "Hello World!";
int MyInt = string1->CompareTo(string2);
Console::WriteLine( MyInt );
```

```
string string1 = "Hello World";
string string2 = "Hello World!";
int MyInt = string1.CompareTo(string2);
Console.WriteLine( MyInt );
```

```
Dim string1 As String = "Hello World"
Dim string2 As String = "Hello World!"
Dim MyInt As Integer = string1.CompareTo(string2)
Console.WriteLine(MyInt)
```

This example displays `-1` to the console.

All overloads of the [String.CompareTo](#) method perform culture-sensitive and case-sensitive comparisons by default. No overloads of this method are provided that allow you to perform a culture-insensitive comparison. For code clarity, we recommend that you use the [String.Compare](#) method instead, specifying [CultureInfo.CurrentCulture](#) for culture-sensitive operations or [CultureInfo.InvariantCulture](#) for culture-insensitive operations. For examples that demonstrate how to use the [String.Compare](#) method to perform both culture-sensitive and culture-insensitive comparisons, see [Performing Culture-Insensitive String Comparisons](#).

Equals method

The [String.Equals](#) method can easily determine if two strings are the same. This case-sensitive method returns a `true` or `false` Boolean value. It can be used from an existing class, as illustrated in the next example. The following example uses the `Equals` method to determine whether a string object contains the phrase "Hello World".

```
String^ string1 = "Hello World";
Console::WriteLine(string1->Equals("Hello World"));
```

```
string string1 = "Hello World";
Console.WriteLine(string1.Equals("Hello World"));
```

```
Dim string1 As String = "Hello World"
Console.WriteLine(string1.Equals("Hello World"))
```

This example displays `True` to the console.

This method can also be used as a static method. The following example compares two string objects using a static method.

```
String^ string1 = "Hello World";
String^ string2 = "Hello World";
Console::WriteLine(String::Equals(string1, string2));
```

```
string string1 = "Hello World";
string string2 = "Hello World";
Console.WriteLine(String.Equals(string1, string2));
```

```
Dim string1 As String = "Hello World"
Dim string2 As String = "Hello World"
Console.WriteLine(String.Equals(string1, string2))
```

This example displays `True` to the console.

StartsWith and EndsWith methods

You can use the [String.StartsWith](#) method to determine whether a string object begins with the same characters that encompass another string. This case-sensitive method returns `true` if the current string object begins with the passed string and `false` if it does not. The following example uses this method to determine if a string object begins with "Hello".

```
String^ string1 = "Hello World";
Console::WriteLine(string1->StartsWith("Hello"));
```

```
string string1 = "Hello World";
Console.WriteLine(string1.StartsWith("Hello"));
```

```
Dim string1 As String = "Hello World!"  
Console.WriteLine(string1.StartsWith("Hello"))
```

This example displays `True` to the console.

The [String.EndsWith](#) method compares a passed string to the characters that exist at the end of the current string object. It also returns a Boolean value. The following example checks the end of a string using the `EndsWith` method.

```
String^ string1 = "Hello World";  
Console::WriteLine(string1->EndsWith("Hello"));
```

```
string string1 = "Hello World";  
Console.WriteLine(string1.EndsWith("Hello"));
```

```
Dim string1 As String = "Hello World!"  
Console.WriteLine(string1.EndsWith("Hello"))
```

This example displays `False` to the console.

`IndexOf` and `LastIndexOf` methods

You can use the [String.IndexOf](#) method to determine the position of the first occurrence of a particular character within a string. This case-sensitive method starts counting from the beginning of a string and returns the position of a passed character using a zero-based index. If the character cannot be found, a value of `-1` is returned.

The following example uses the `IndexOf` method to search for the first occurrence of the '`l`' character in a string.

```
String^ string1 = "Hello World";  
Console::WriteLine(string1->IndexOf('l'));
```

```
string string1 = "Hello World";  
Console.WriteLine(string1.IndexOf('l'));
```

```
Dim string1 As String = "Hello World!"  
Console.WriteLine(string1.IndexOf("l"))
```

This example displays `2` to the console.

The [String.LastIndexOf](#) method is similar to the `String.IndexOf` method except that it returns the position of the last occurrence of a particular character within a string. It is case-sensitive and uses a zero-based index.

The following example uses the `LastIndexOf` method to search for the last occurrence of the '`l`' character in a string.

```
String^ string1 = "Hello World";  
Console::WriteLine(string1->LastIndexOf('l'));
```

```
string string1 = "Hello World";
Console.WriteLine(string1.LastIndexOf('l'));
```

```
Dim string1 As String = "Hello World!"
Console.WriteLine(string1.LastIndexOf("l"))
```

This example displays `9` to the console.

Both methods are useful when used in conjunction with the [String.Remove](#) method. You can use either the `IndexOf` or `LastIndexOf` methods to retrieve the position of a character, and then supply that position to the `Remove` method in order to remove a character or a word that begins with that character.

See also

- [Best practices for using strings in .NET](#)
- [Basic string operations](#)
- [Perform culture-insensitive string operations](#)
- [Sorting weight tables](#) - used by .NET Framework and .NET Core 1.0-3.1 on Windows
- [Default Unicode collation element table](#) - used by .NET 5 on all platforms, and by .NET Core on Linux and macOS

Change case in .NET

9/20/2022 • 4 minutes to read • [Edit Online](#)

If you write an application that accepts input from a user, you can never be sure what case (upper or lower) they will use to enter the data. Often, you want strings to be cased consistently, particularly if you are displaying them in the user interface. The following table describes three case-changing methods. The first two methods provide an overload that accepts a culture.

METHOD NAME	USE
<code>String.ToUpper</code>	Converts all characters in a string to uppercase.
<code>String.ToLower</code>	Converts all characters in a string to lowercase.
<code>TextInfo.ToTitleCase</code>	Converts a string to title case.

WARNING

The `String.ToUpper` and `String.ToLower` methods should not be used to convert strings in order to compare them or test them for equality. For more information, see the [Compare strings of mixed case](#) section.

Compare strings of mixed case

To compare strings of mixed case to determine their ordering, call one of the overloads of the `String.CompareTo` method with a `comparisonType` parameter, and provide a value of either `StringComparison.CurrentCultureIgnoreCase`, `StringComparison.InvariantCultureIgnoreCase`, or `StringComparison.OrdinalIgnoreCase` for the `comparisonType` argument. For a comparison using a specific culture other than the current culture, call an overload of the `String.CompareTo` method with both a `culture` and `options` parameter, and provide a value of `CompareOptions.IgnoreCase` as the `options` argument.

To compare strings of mixed case to determine whether they're equal, call one of the overloads of the `String.Equals` method with a `comparisonType` parameter, and provide a value of either `StringComparison.CurrentCultureIgnoreCase`, `StringComparison.InvariantCultureIgnoreCase`, or `StringComparison.OrdinalIgnoreCase` for the `comparisonType` argument.

For more information, see [Best practices for using strings](#).

`ToUpper` method

The `String.ToUpper` method changes all characters in a string to uppercase. The following example converts the string "Hello World!" from mixed case to uppercase.

```
string properString = "Hello World!";
Console.WriteLine(properString.ToUpper());
// This example displays the following output:
//      HELLO WORLD!
```

```
Dim MyString As String = "Hello World!"
Console.WriteLine(MyString.ToUpper())
' This example displays the following output:
'      HELLO WORLD!
```

The preceding example is culture-sensitive by default; it applies the casing conventions of the current culture. To perform a culture-insensitive case change or to apply the casing conventions of a particular culture, use the [String.ToUpper\(CultureInfo\)](#) method overload and supply a value of [CultureInfo.InvariantCulture](#) or a [System.Globalization.CultureInfo](#) object that represents the specified culture to the `culture` parameter. For an example that demonstrates how to use the [ToUpper](#) method to perform a culture-insensitive case change, see [Perform culture-insensitive case changes](#).

ToLower method

The [String.ToLower](#) method is similar to the previous method, but instead converts all the characters in a string to lowercase. The following example converts the string "Hello World!" to lowercase.

```
string properString = "Hello World!";
Console.WriteLine(properString.ToLower());
// This example displays the following output:
//      hello world!
```

```
Dim MyString As String = "Hello World!"
Console.WriteLine(MyString.ToLower())
' This example displays the following output:
'      hello world!
```

The preceding example is culture-sensitive by default; it applies the casing conventions of the current culture. To perform a culture-insensitive case change or to apply the casing conventions of a particular culture, use the [String.ToLower\(CultureInfo\)](#) method overload and supply a value of [CultureInfo.InvariantCulture](#) or a [System.Globalization.CultureInfo](#) object that represents the specified culture to the `culture` parameter. For an example that demonstrates how to use the [ToLower\(CultureInfo\)](#) method to perform a culture-insensitive case change, see [Perform culture-insensitive case changes](#).

ToTitleCase method

The [TextInfo.ToTitleCase](#) converts the first character of each word to uppercase and the remaining characters to lowercase. However, words that are entirely uppercase are assumed to be acronyms and are not converted.

The [TextInfo.ToTitleCase](#) method is culture-sensitive; that is, it uses the casing conventions of a particular culture. In order to call the method, you first retrieve the [TextInfo](#) object that represents the casing conventions of the particular culture from the [CultureInfo.TextInfo](#) property of a particular culture.

The following example passes each string in an array to the [TextInfo.ToTitleCase](#) method. The strings include proper title strings as well as acronyms. The strings are converted to title case by using the casing conventions of the English (United States) culture.

```

using System;
using System.Globalization;

public class Example
{
    public static void Main()
    {
        string[] values = { "a tale of two cities", "gROWL to the rescue",
                           "inside the US government", "sports and MLB baseball",
                           "The Return of Sherlock Holmes", "UNICEF and children"};

        CultureInfo ti = CultureInfo.CurrentCulture.TextInfo;
        foreach (var value in values)
            Console.WriteLine("{0} --> {1}", value, ti.ToTitleCase(value));
    }
}

// The example displays the following output:
//   a tale of two cities --> A Tale Of Two Cities
//   gROWL to the rescue --> Growl To The Rescue
//   inside the US government --> Inside The US Government
//   sports and MLB baseball --> Sports And MLB Baseball
//   The Return of Sherlock Holmes --> The Return Of Sherlock Holmes
//   UNICEF and children --> UNICEF And Children

```

```

Imports System.Globalization

Module Example
    Public Sub Main()
        Dim values() As String = {"a tale of two cities", "gROWL to the rescue",
                                 "inside the US government", "sports and MLB baseball",
                                 "The Return of Sherlock Holmes", "UNICEF and children"}

        Dim ti As TextInfo = CultureInfo.CurrentCulture.TextInfo
        For Each value In values
            Console.WriteLine("{0} --> {1}", value, ti.ToTitleCase(value))
        Next
    End Sub
End Module

' The example displays the following output:
'   a tale of two cities --> A Tale Of Two Cities
'   gROWL to the rescue --> Growl To The Rescue
'   inside the US government --> Inside The US Government
'   sports and MLB baseball --> Sports And MLB Baseball
'   The Return of Sherlock Holmes --> The Return Of Sherlock Holmes
'   UNICEF and children --> UNICEF And Children

```

Note that although it is culture-sensitive, the [TextInfo.ToTitleCase](#) method does not provide linguistically correct casing rules. For instance, in the previous example, the method converts "a tale of two cities" to "A Tale Of Two Cities". However, the linguistically correct title casing for the en-US culture is "A Tale of Two Cities."

See also

- [Basic String Operations](#)
- [Perform culture-insensitive string operations](#)

Extract substrings from a string

9/20/2022 • 10 minutes to read • [Edit Online](#)

This article covers some different techniques for extracting parts of a string.

- Use the [Split method](#) when the substrings you want are separated by a known delimiting character (or characters).
- [Regular expressions](#) are useful when the string conforms to a fixed pattern.
- Use the [IndexOf](#) and [Substring methods](#) in conjunction when you don't want to extract *all* of the substrings in a string.

String.Split method

[String.Split](#) provides a handful of overloads to help you break up a string into a group of substrings based on one or more delimiting characters that you specify. You can choose to limit the total number of substrings in the final result, trim white-space characters from substrings, or exclude empty substrings.

The following examples show three different overloads of `String.Split()`. The first example calls the [Split\(Char\[\]\)](#) overload without passing any separator characters. When you don't specify any delimiting characters, `String.Split()` uses default delimiters, which are white-space characters, to split up the string.

```
string s = "You win some. You lose some.";

string[] subs = s.Split();

foreach (string sub in subs)
{
    Console.WriteLine($"Substring: {sub}");
}

// This example produces the following output:
//
// Substring: You
// Substring: win
// Substring: some.
// Substring: You
// Substring: lose
// Substring: some.
```

```
Dim s As String = "You win some. You lose some."
Dim subs As String() = s.Split()

For Each substring As String In subs
    Console.WriteLine("Substring: {0}", substring)
Next

' This example produces the following output:
'
' Substring: You
' Substring: win
' Substring: some.
' Substring: You
' Substring: lose
' Substring: some.
```

As you can see, the period characters (.) are included in two of the substrings. If you want to exclude the period characters, you can add the period character as an additional delimiting character. The next example shows how to do this.

```
string s = "You win some. You lose some.";

string[] subs = s.Split(' ', '.');

foreach (string sub in subs)
{
    Console.WriteLine($"Substring: {sub}");
}

// This example produces the following output:
//
// Substring: You
// Substring: win
// Substring: some
// Substring:
// Substring: You
// Substring: lose
// Substring: some
// Substring:
```

```
Dim s As String = "You win some. You lose some."
Dim subs As String() = s.Split(" "c, ".c")

For Each substring As String In subs
    Console.WriteLine("Substring: {0}", substring)
Next

' This example produces the following output:
'
' Substring: You
' Substring: win
' Substring: some
' Substring:
' Substring: You
' Substring: lose
' Substring: some
' Substring:
```

The periods are gone from the substrings, but now two extra empty substrings have been included. These empty substrings represent the substring between the word and the period that follows it. To omit empty substrings from the resulting array, you can call the [Split\(Char\[\], StringSplitOptions\)](#) overload and specify [StringSplitOptions.RemoveEmptyEntries](#) for the `options` parameter.

```

string s = "You win some. You lose some.";
char[] separators = new char[] { ' ', '.' };

string[] subs = s.Split(separators, StringSplitOptions.RemoveEmptyEntries);

foreach (string sub in subs)
{
    Console.WriteLine($"Substring: {sub}");
}

// This example produces the following output:
//
// Substring: You
// Substring: win
// Substring: some
// Substring: You
// Substring: lose
// Substring: some

```

```

Dim s As String = "You win some. You lose some."
Dim separators As Char() = New Char() {" ", "."}
Dim subs As String() = s.Split(separators, StringSplitOptions.RemoveEmptyEntries)

For Each substring As String In subs
    Console.WriteLine("Substring: {0}", substring)
Next

' This example produces the following output:
'
' Substring: You
' Substring: win
' Substring: some
' Substring: You
' Substring: lose
' Substring: some

```

Regular expressions

If your string conforms to a fixed pattern, you can use a regular expression to extract and handle its elements. For example, if strings take the form "*number operand number*", you can use a [regular expression](#) to extract and handle the string's elements. Here's an example:

```

String[] expressions = { "16 + 21", "31 * 3", "28 / 3",
                       "42 - 18", "12 * 7",
                       "2, 4, 6, 8" };
String pattern = @"(\d+)\s+([-*/])\s+(\d+)";

foreach (string expression in expressions)
{
    foreach (System.Text.RegularExpressions.Match m in
        System.Text.RegularExpressions.Regex.Matches(expression, pattern))
    {
        int value1 = Int32.Parse(m.Groups[1].Value);
        int value2 = Int32.Parse(m.Groups[3].Value);
        switch (m.Groups[2].Value)
        {
            case "+":
                Console.WriteLine("{0} = {1}", m.Value, value1 + value2);
                break;
            case "-":
                Console.WriteLine("{0} = {1}", m.Value, value1 - value2);
                break;
            case "*":
                Console.WriteLine("{0} = {1}", m.Value, value1 * value2);
                break;
            case "/":
                Console.WriteLine("{0} = {1:N2}", m.Value, value1 / value2);
                break;
        }
    }
}

// The example displays the following output:
//      16 + 21 = 37
//      31 * 3 = 93
//      28 / 3 = 9.33
//      42 - 18 = 24
//      12 * 7 = 84

```

```

Dim expressions() As String = {"16 + 21", "31 * 3", "28 / 3",
                               "42 - 18", "12 * 7",
                               "2, 4, 6, 8"}

Dim pattern As String = "(\d+)\s+([-*/])\s+(\d+)"
For Each expression In expressions
    For Each m As Match In Regex.Matches(expression, pattern)
        Dim value1 As Integer = Int32.Parse(m.Groups(1).Value)
        Dim value2 As Integer = Int32.Parse(m.Groups(3).Value)
        Select Case m.Groups(2).Value
            Case "+"
                Console.WriteLine("{0} = {1}", m.Value, value1 + value2)
            Case "-"
                Console.WriteLine("{0} = {1}", m.Value, value1 - value2)
            Case "*"
                Console.WriteLine("{0} = {1}", m.Value, value1 * value2)
            Case "/"
                Console.WriteLine("{0} = {1:N2}", m.Value, value1 / value2)
        End Select
    Next
Next

' The example displays the following output:
'      16 + 21 = 37
'      31 * 3 = 93
'      28 / 3 = 9.33
'      42 - 18 = 24
'      12 * 7 = 84

```

The regular expression pattern `(\d+)\s+([-*/])\s+(\d+)` is defined like this:

PATTERN	DESCRIPTION
<code>(\d+)</code>	Match one or more decimal digits. This is the first capturing group.
<code>\s+</code>	Match one or more white-space characters.
<code>([-*/])</code>	Match an arithmetic operator sign (+, -, *, or /). This is the second capturing group.
<code>\s+</code>	Match one or more white-space characters.
<code>(\d+)</code>	Match one or more decimal digits. This is the third capturing group.

You can also use a regular expression to extract substrings from a string based on a pattern rather than a fixed set of characters. This is a common scenario when either of these conditions occurs:

- One or more of the delimiter characters does not *always* serve as a delimiter in the [String](#) instance.
- The sequence and number of delimiter characters is variable or unknown.

For example, the [Split](#) method cannot be used to split the following string, because the number of `\n` (newline) characters is variable, and they don't always serve as delimiters.

```
[This is captured\n text.]\\n\\n[\\n[This is more captured text.]\\n]\\n[Some more captured text:\\n    Option1\\n    Option2][Terse text.]
```

A regular expression can split this string easily, as the following example shows.

```
String input = "[This is captured\\ntext.]\\n\\n[\\n" +
    "[This is more captured text.]\\n]\\n" +
    "[Some more captured text:\\n    Option1" +
    "\\n    Option2][Terse text.]";
String pattern = @"\[(^\[\]]+)\]";
int ctr = 0;

foreach (System.Text.RegularExpressions.Match m in
    System.Text.RegularExpressions.Regex.Matches(input, pattern))
{
    Console.WriteLine("{0}: {1}", ++ctr, m.Groups[1].Value);
}

// The example displays the following output:
//      1: This is captured
//      text.
//      2: This is more captured text.
//      3: Some more captured text:
//          Option1
//          Option2
//      4: Terse text.
```

```

Dim input As String = String.Format("[This is captured{0}text.]" +
    "{0}{0}[{0}[This is more " +
    "captured text.]{0}{0}" +
    "[Some more captured text:" +
    "{0}    Option1" +
    "{0}    Option2][Terse text.]",
    vbCrLf)
Dim pattern As String = "\[([^\[\]]+)\]"
Dim ctr As Integer = 0
For Each m As Match In Regex.Matches(input, pattern)
    ctr += 1
    Console.WriteLine("{0}: {1}", ctr, m.Groups(1).Value)
Next

' The example displays the following output:
' 1: This is captured
' text.
' 2: This is more captured text.
' 3: Some more captured text:
'     Option1
'     Option2
' 4: Terse text.

```

The regular expression pattern `\[([^\[\]]+)\]` is defined like this:

PATTERN	DESCRIPTION
<code>\[</code>	Match an opening bracket.
<code>([^[\]]+)</code>	Match any character that is not an opening or a closing bracket one or more times. This is the first capturing group.
<code>\]</code>	Match a closing bracket.

The [Regex.Split](#) method is almost identical to [String.Split](#), except that it splits a string based on a regular expression pattern instead of a fixed character set. For example, the following example uses the [Regex.Split](#) method to split a string that contains substrings delimited by various combinations of hyphens and other characters.

```

String input = "abacus -- alabaster - * - atrium -- " +
    "any -* actual - + - armoire - - alarm";
String pattern = @"\s-\s?[*]?\s?-`s";
String[] elements = System.Text.RegularExpressions.Regex.Split(input, pattern);

foreach (string element in elements)
    Console.WriteLine(element);

// The example displays the following output:
//      abacus
//      alabaster
//      atrium
//      any
//      actual
//      armoire
//      alarm

```

```

Dim input As String = "abacus -- alabaster - * - atrium +- " +
    "any -* actual - + - armoire - - alarm"
Dim pattern As String = "\s-\s?[+*]?\s?-\s"
Dim elements() As String = Regex.Split(input, pattern)
For Each element In elements
    Console.WriteLine(element)
Next

' The example displays the following output:
'     abacus
'     alabaster
'     atrium
'     any
'     actual
'     armoire
'     alarm

```

The regular expression pattern `\s-\s?[+*]?\s?-\s` is defined like this:

PATTERN	DESCRIPTION
<code>\s-</code>	Match a white-space character followed by a hyphen.
<code>\s?</code>	Match zero or one white-space character.
<code>[+*]?</code>	Match zero or one occurrence of either the + or * character.
<code>\s?</code>	Match zero or one white-space character.
<code>-\s</code>	Match a hyphen followed by a white-space character.

String.IndexOf and String.Substring methods

If you aren't interested in all of the substrings in a string, you might prefer to work with one of the string comparison methods that returns the index at which the match begins. You can then call the [Substring](#) method to extract the substring that you want. The string comparison methods include:

- [IndexOf](#), which returns the zero-based index of the first occurrence of a character or string in a string instance.
- [IndexOfAny](#), which returns the zero-based index in the current string instance of the first occurrence of any character in a character array.
- [LastIndexOf](#), which returns the zero-based index of the last occurrence of a character or string in a string instance.
- [LastIndexOfAny](#), which returns a zero-based index in the current string instance of the last occurrence of any character in a character array.

The following example uses the [IndexOf](#) method to find the periods in a string. It then uses the [Substring](#) method to return full sentences.

```

String s = "This is the first sentence in a string. " +
    "More sentences will follow. For example, " +
    "this is the third sentence. This is the " +
    "fourth. And this is the fifth and final " +
    "sentence.";
var sentences = new List<String>();
int start = 0;
int position;

// Extract sentences from the string.
do
{
    position = s.IndexOf('.', start);
    if (position >= 0)
    {
        sentences.Add(s.Substring(start, position - start + 1).Trim());
        start = position + 1;
    }
} while (position > 0);

// Display the sentences.
foreach (var sentence in sentences)
    Console.WriteLine(sentence);

// The example displays the following output:
//      This is the first sentence in a string.
//      More sentences will follow.
//      For example, this is the third sentence.
//      This is the fourth.
//      And this is the fifth and final sentence.

```

```

Dim input As String = "This is the first sentence in a string. " +
    "More sentences will follow. For example, " +
    "this is the third sentence. This is the " +
    "fourth. And this is the fifth and final " +
    "sentence."
Dim sentences As New List(Of String)
Dim start As Integer = 0
Dim position As Integer

' Extract sentences from the string.
Do
    position = input.IndexOf("."c, start)
    If position >= 0 Then
        sentences.Add(input.Substring(start, position - start + 1).Trim())
        start = position + 1
    End If
Loop While position > 0

' Display the sentences.
For Each sentence In sentences
    Console.WriteLine(sentence)
Next
End Sub

' The example displays the following output:
'      This is the first sentence in a string.
'      More sentences will follow.
'      For example, this is the third sentence.
'      This is the fourth.
'      And this is the fifth and final sentence.

```

See also

- Basic string operations in .NET
- .NET regular expressions
- How to parse strings using String.Split in C#

Using the `StringBuilder` Class in .NET

9/20/2022 • 7 minutes to read • [Edit Online](#)

The `String` object is immutable. Every time you use one of the methods in the `System.String` class, you create a new string object in memory, which requires a new allocation of space for that new object. In situations where you need to perform repeated modifications to a string, the overhead associated with creating a new `String` object can be costly. The `System.Text.StringBuilder` class can be used when you want to modify a string without creating a new object. For example, using the `StringBuilder` class can boost performance when concatenating many strings together in a loop.

Importing the `System.Text` Namespace

The `StringBuilder` class is found in the `System.Text` namespace. To avoid having to provide a fully qualified type name in your code, you can import the `System.Text` namespace:

```
using namespace System;
using namespace System::Text;
```

```
using System;
using System.Text;
```

```
Imports System.Text
```

Instantiating a `StringBuilder` Object

You can create a new instance of the `StringBuilder` class by initializing your variable with one of the overloaded constructor methods, as illustrated in the following example.

```
StringBuilder^ myStringBuilder = gcnew StringBuilder("Hello World!");
```

```
StringBuilder myStringBuilder = new StringBuilder("Hello World!");
```

```
Dim myStringBuilder As New StringBuilder("Hello World!")
```

Setting the Capacity and Length

Although the `StringBuilder` is a dynamic object that allows you to expand the number of characters in the string that it encapsulates, you can specify a value for the maximum number of characters that it can hold. This value is called the capacity of the object and should not be confused with the length of the string that the current `StringBuilder` holds. For example, you might create a new instance of the `StringBuilder` class with the string "Hello", which has a length of 5, and you might specify that the object has a maximum capacity of 25. When you modify the `StringBuilder`, it does not reallocate size for itself until the capacity is reached. When this occurs, the new space is allocated automatically and the capacity is doubled. You can specify the capacity of the `StringBuilder` class using one of the overloaded constructors. The following example specifies that the

`myStringBuilder` object can be expanded to a maximum of 25 spaces.

```
StringBuilder^ myStringBuilder = gcnew StringBuilder("Hello World!", 25);
```

```
StringBuilder myStringBuilder = new StringBuilder("Hello World!", 25);
```

```
Dim myStringBuilder As New StringBuilder("Hello World!", 25)
```

Additionally, you can use the read/write [Capacity](#) property to set the maximum length of your object. The following example uses the [Capacity](#) property to define the maximum object length.

```
myStringBuilder->Capacity = 25;
```

```
myStringBuilder.Capacity = 25;
```

```
myStringBuilder.Capacity = 25
```

The [EnsureCapacity](#) method can be used to check the capacity of the current [StringBuilder](#). If the capacity is greater than the passed value, no change is made; however, if the capacity is smaller than the passed value, the current capacity is changed to match the passed value.

The [Length](#) property can also be viewed or set. If you set the [Length](#) property to a value that is greater than the [Capacity](#) property, the [Capacity](#) property is automatically changed to the same value as the [Length](#) property. Setting the [Length](#) property to a value that is less than the length of the string within the current [StringBuilder](#) shortens the string.

Modifying the [StringBuilder](#) String

The following table lists the methods you can use to modify the contents of a [StringBuilder](#).

METHOD NAME	USE
StringBuilder.Append	Appends information to the end of the current StringBuilder .
StringBuilder.AppendFormat	Replaces a format specifier passed in a string with formatted text.
StringBuilder.Insert	Inserts a string or object into the specified index of the current StringBuilder .
StringBuilder.Remove	Removes a specified number of characters from the current StringBuilder .
StringBuilder.Replace	Replaces all occurrences of a specified character or string in the current StringBuilder with another specified character or string.

Append

The [Append](#) method can be used to add text or a string representation of an object to the end of a string

represented by the current `StringBuilder`. The following example initializes a `StringBuilder` to "Hello World" and then appends some text to the end of the object. Space is allocated automatically as needed.

```
StringBuilder^ myStringBuilder = gcnew StringBuilder("Hello World!");
myStringBuilder->Append(" What a beautiful day.");
Console::WriteLine(myStringBuilder);
// The example displays the following output:
//      Hello World! What a beautiful day.
```

```
StringBuilder myStringBuilder = new StringBuilder("Hello World!");
myStringBuilder.Append(" What a beautiful day.");
Console.WriteLine(myStringBuilder);
// The example displays the following output:
//      Hello World! What a beautiful day.
```

```
Dim myStringBuilder As New StringBuilder("Hello World!")
myStringBuilder.Append(" What a beautiful day.")
Console.WriteLine(myStringBuilder)
' The example displays the following output:
'      Hello World! What a beautiful day.
```

AppendFormat

The `StringBuilder.AppendFormat` method adds text to the end of the `StringBuilder` object. It supports the composite formatting feature (for more information, see [Composite Formatting](#)) by calling the `IFormattable` implementation of the object or objects to be formatted. Therefore, it accepts the standard format strings for numeric, date and time, and enumeration values, the custom format strings for numeric and date and time values, and the format strings defined for custom types. (For information about formatting, see [Formatting Types](#).) You can use this method to customize the format of variables and append those values to a `StringBuilder`. The following example uses the `AppendFormat` method to place an integer value formatted as a currency value at the end of a `StringBuilder` object.

```
int MyInt = 25;
StringBuilder^ myStringBuilder = gcnew StringBuilder("Your total is ");
myStringBuilder->AppendFormat("{0:C} ", MyInt);
Console::WriteLine(myStringBuilder);
// The example displays the following output:
//      Your total is $25.00
```

```
int MyInt = 25;
StringBuilder myStringBuilder = new StringBuilder("Your total is ");
myStringBuilder.AppendFormat("{0:C} ", MyInt);
Console.WriteLine(myStringBuilder);
// The example displays the following output:
//      Your total is $25.00
```

```
Dim MyInt As Integer = 25
Dim myStringBuilder As New StringBuilder("Your total is ")
myStringBuilder.AppendFormat("{0:C} ", MyInt)
Console.WriteLine(myStringBuilder)
' The example displays the following output:
'      Your total is $25.00
```

Insert

The `Insert` method adds a string or object to a specified position in the current `StringBuilder` object. The

following example uses this method to insert a word into the sixth position of a [StringBuilder](#) object.

```
StringBuilder^ myStringBuilder = gcnew StringBuilder("Hello World!");
myStringBuilder->Insert(6,"Beautiful ");
Console::WriteLine(myStringBuilder);
// The example displays the following output:
//      Hello Beautiful World!
```

```
StringBuilder myStringBuilder = new StringBuilder("Hello World!");
myStringBuilder.Insert(6,"Beautiful ");
Console.WriteLine(myStringBuilder);
// The example displays the following output:
//      Hello Beautiful World!
```

```
Dim myStringBuilder As New StringBuilder("Hello World!")
myStringBuilder.Insert(6, "Beautiful ")
Console.WriteLine(myStringBuilder)
' The example displays the following output:
'      Hello Beautiful World!
```

Remove

You can use the **Remove** method to remove a specified number of characters from the current [StringBuilder](#) object, beginning at a specified zero-based index. The following example uses the **Remove** method to shorten a [StringBuilder](#) object.

```
StringBuilder^ myStringBuilder = gcnew StringBuilder("Hello World!");
myStringBuilder->Remove(5,7);
Console::WriteLine(myStringBuilder);
// The example displays the following output:
//      Hello
```

```
StringBuilder myStringBuilder = new StringBuilder("Hello World!");
myStringBuilder.Remove(5,7);
Console.WriteLine(myStringBuilder);
// The example displays the following output:
//      Hello
```

```
Dim myStringBuilder As New StringBuilder("Hello World!")
myStringBuilder.Remove(5, 7)
Console.WriteLine(myStringBuilder)
' The example displays the following output:
'      Hello
```

Replace

The **Replace** method can be used to replace characters within the [StringBuilder](#) object with another specified character. The following example uses the **Replace** method to search a [StringBuilder](#) object for all instances of the exclamation point character (!) and replace them with the question mark character (?).

```
StringBuilder^ myStringBuilder = gcnew StringBuilder("Hello World!");
myStringBuilder->Replace('!', '?');
Console::WriteLine(myStringBuilder);
// The example displays the following output:
//      Hello World?
```

```
StringBuilder myStringBuilder = new StringBuilder("Hello World!");
myStringBuilder.Replace('!', '?');
Console.WriteLine(myStringBuilder);
// The example displays the following output:
//      Hello World?
```

```
Dim myStringBuilder As New StringBuilder("Hello World!")
myStringBuilder.Replace("!c", "?c")
Console.WriteLine(myStringBuilder)
' The example displays the following output:
'      Hello World?
```

Converting a `StringBuilder` Object to a `String`

You must convert the `StringBuilder` object to a `String` object before you can pass the string represented by the `StringBuilder` object to a method that has a `String` parameter or display it in the user interface. You do this conversion by calling the `StringBuilder.ToString` method. The following example calls a number of `StringBuilder` methods and then calls the `StringBuilder.ToString()` method to display the string.

```
using System;
using System.Text;

public class Example
{
    public static void Main()
    {
        StringBuilder sb = new StringBuilder();
        bool flag = true;
        string[] spellings = { "recieve", "receeve", "receive" };
        sb.AppendFormat("Which of the following spellings is {0}:", flag);
        sb.AppendLine();
        for (int ctr = 0; ctr <= spellings.GetUpperBound(0); ctr++) {
            sb.AppendFormat("  {0}. {1}", ctr, spellings[ctr]);
            sb.AppendLine();
        }
        sb.AppendLine();
        Console.WriteLine(sb.ToString());
    }
}
// The example displays the following output:
//      Which of the following spellings is True:
//          0. recieve
//          1. receeve
//          2. receive
```

```
Imports System.Text

Module Example
    Public Sub Main()
        Dim sb As New StringBuilder()
        Dim flag As Boolean = True
        Dim spellings() As String = {"recieve", "receeve", "receive"}
        sb.AppendFormat("Which of the following spellings is {0}:", flag)
        sb.AppendLine()
        For ctr As Integer = 0 To spellings.GetUpperBound(0)
            sb.AppendFormat(" {0}. {1}", ctr, spellings(ctr))
            sb.AppendLine()
        Next
        sb.AppendLine()
        Console.WriteLine(sb.ToString())
    End Sub
End Module
' The example displays the following output:
'     Which of the following spellings is True:
'         0. recieve
'         1. receeve
'         2. receive
```

See also

- [System.Text.StringBuilder](#)
- [Basic String Operations](#)
- [Formatting Types](#)

How to: Perform Basic String Manipulations in .NET

9/20/2022 • 4 minutes to read • [Edit Online](#)

The following example uses some of the methods discussed in the [Basic String Operations](#) topics to construct a class that performs string manipulations in a manner that might be found in a real-world application. The `MailToData` class stores the name and address of an individual in separate properties and provides a way to combine the `City`, `State`, and `Zip` fields into a single string for display to the user. Furthermore, the class allows the user to enter the city, state, and zip code information as a single string. The application automatically parses the single string and enters the proper information into the corresponding property.

For simplicity, this example uses a console application with a command-line interface.

Example

```
using System;

class MainClass
{
    static void Main()
    {
        MailToData MyData = new MailToData();

        Console.Write("Enter Your Name: ");
        MyData.Name = Console.ReadLine();
        Console.Write("Enter Your Address: ");
        MyData.Address = Console.ReadLine();
        Console.Write("Enter Your City, State, and ZIP Code separated by spaces: ");
        MyData.CityStateZip = Console.ReadLine();
        Console.WriteLine();

        if (MyData.Validated) {
            Console.WriteLine("Name: {0}", MyData.Name);
            Console.WriteLine("Address: {0}", MyData.Address);
            Console.WriteLine("City: {0}", MyData.City);
            Console.WriteLine("State: {0}", MyData.State);
            Console.WriteLine("Zip: {0}", MyData.Zip);

            Console.WriteLine("\nThe following address will be used:");
            Console.WriteLine(MyData.Address);
            Console.WriteLine(MyData.CityStateZip);
        }
    }
}

public class MailToData
{
    string name = "";
    string address = "";
    string citystatezip = "";
    string city = "";
    string state = "";
    string zip = "";
    bool parseSucceeded = false;

    public string Name
    {
        get{return name;}
        set{name = value;}
    }
}
```

```

public string Address
{
    get{return address;}
    set{address = value;}
}

public string CityStateZip
{
    get {
        return String.Format("{0}, {1} {2}", city, state, zip);
    }
    set {
        citystatezip = value.Trim();
        ParseCityStateZip();
    }
}

public string City
{
    get{return city;}
    set{city = value;}
}

public string State
{
    get{return state;}
    set{state = value;}
}

public string Zip
{
    get{return zip;}
    set{zip = value;}
}

public bool Validated
{
    get { return parseSucceeded; }
}

private void ParseCityStateZip()
{
    string msg = "";
    const string msgEnd = "\nYou must enter spaces between city, state, and zip code.\n";

    // Throw a FormatException if the user did not enter the necessary spaces
    // between elements.
    try
    {
        // City may consist of multiple words, so we'll have to parse the
        // string from right to left starting with the zip code.
        int zipIndex = citystatezip.LastIndexOf(" ");
        if (zipIndex == -1) {
            msg = "\nCannot identify a zip code." + msgEnd;
            throw new FormatException(msg);
        }
        zip = citystatezip.Substring(zipIndex + 1);

        int stateIndex = citystatezip.LastIndexOf(" ", zipIndex - 1);
        if (stateIndex == -1) {
            msg = "\nCannot identify a state." + msgEnd;
            throw new FormatException(msg);
        }
        state = citystatezip.Substring(stateIndex + 1, zipIndex - stateIndex - 1);
        state = state.ToUpper();

        city = citystatezip.Substring(0, stateIndex);
        if (city.Length == 0) {
            msg = "\nCity cannot be empty." + msgEnd;
            throw new FormatException(msg);
        }
    }
}

```

```

        msg = "\nCannot identify a city." + msgEnd;
        throw new FormatException(msg);
    }
    parseSucceeded = true;
}
catch (FormatException ex)
{
    Console.WriteLine(ex.Message);
}
}

private string ReturnCityStateZip()
{
    // Make state uppercase.
    state = state.ToUpper();

    // Put the value of city, state, and zip together in the proper manner.
    string MyCityStateZip = String.Concat(city, ", ", state, " ", zip);

    return MyCityStateZip;
}
}

```

```

Class MainClass
Public Shared Sub Main()
    Dim MyData As New MailToData()

    Console.Write("Enter Your Name: ")
    MyData.Name = Console.ReadLine()
    Console.Write("Enter Your Address: ")
    MyData.Address = Console.ReadLine()
    Console.Write("Enter Your City, State, and ZIP Code separated by spaces: ")
    MyData.CityStateZip = Console.ReadLine()
    Console.WriteLine()

    If MyData.Validated Then
        Console.WriteLine("Name: {0}", MyData.Name)
        Console.WriteLine("Address: {0}", MyData.Address)
        Console.WriteLine("City: {0}", MyData.City)
        Console.WriteLine("State: {0}", MyData.State)
        Console.WriteLine("ZIP Code: {0}", MyData.Zip)

        Console.WriteLine("The following address will be used:")
        Console.WriteLine(MyData.Address)
        Console.WriteLine(MyData.CityStateZip)
    End If
End Sub
End Class

Public Class MailToData
    Private strName As String = ""
    Private strAddress As String = ""
    Private strCityStateZip As String = ""
    Private strCity As String = ""
    Private strState As String = ""
    Private strZip As String = ""
    Private parseSucceeded As Boolean = False

    Public Property Name() As String
        Get
            Return strName
        End Get
        Set
            strName = value
        End Set
    End Property

```

```

    Public Property Address() As String
        Get
            Return strAddress
        End Get
        Set
            strAddress = value
        End Set
    End Property

    Public Property CityStateZip() As String
        Get
            Return String.Format("{0}, {1} {2}", strCity, strState, strZip)
        End Get
        Set
            strCityStateZip = value.Trim()
            ParseCityStateZip()
        End Set
    End Property

    Public Property City() As String
        Get
            Return strCity
        End Get
        Set
            strCity = value
        End Set
    End Property

    Public Property State() As String
        Get
            Return strState
        End Get
        Set
            strState = value
        End Set
    End Property

    Public Property Zip() As String
        Get
            Return strZip
        End Get
        Set
            strZip = value
        End Set
    End Property

    Public ReadOnly Property Validated As Boolean
        Get
            Return parseSucceeded
        End Get
    End Property

    Private Sub ParseCityStateZip()
        Dim msg As String = Nothing
        Const msgEnd As String = vbCrLf +
                               "You must enter spaces between city, state, and zip code." +
                               vbCrLf

        ' Throw a FormatException if the user did not enter the necessary spaces
        ' between elements.
        Try
            ' City may consist of multiple words, so we'll have to parse the
            ' string from right to left starting with the zip code.
            Dim zipIndex As Integer = strCityStateZip.LastIndexOf(" ")
            If zipIndex = -1 Then
                msg = vbCrLf + "Cannot identify a zip code." + msgEnd
                Throw New FormatException(msg)
            End If
            strZip = strCityStateZip.Substring(zipIndex + 1)
        End Try
    End Sub

```

```
Dim stateIndex As Integer = strCityStateZip.LastIndexOf(" ", zipIndex - 1)
If stateIndex = -1 Then
    msg = vbCrLf + "Cannot identify a state." + msgEnd
    Throw New FormatException(msg)
End If
strState = strCityStateZip.Substring(stateIndex + 1, zipIndex - stateIndex - 1)
strState = strState.ToUpper()

strCity = strCityStateZip.Substring(0, stateIndex)
If strCity.Length = 0 Then
    msg = vbCrLf + "Cannot identify a city." + msgEnd
    Throw New FormatException(msg)
End If
parseSucceeded = True
Catch ex As FormatException
    Console.WriteLine(ex.Message)
End Try
End Sub
End Class
```

When the preceding code is executed, the user is asked to enter their name and address. The application places the information in the appropriate properties and displays the information back to the user, creating a single string that displays the city, state, and zip code information.

See also

- [Basic String Operations](#)

Parse strings in .NET

9/20/2022 • 2 minutes to read • [Edit Online](#)

A *parsing* operation converts a string that represents a .NET base type into that base type. For example, a parsing operation is used to convert a string to a floating-point number or to a date-and-time value. The method most commonly used to perform a parsing operation is the `Parse` method. Because parsing is the reverse operation of formatting (which involves converting a base type into its string representation), many of the same rules and conventions apply. Just as formatting uses an object that implements the `IFormatProvider` interface to provide culture-sensitive formatting information, parsing also uses an object that implements the `IFormatProvider` interface to determine how to interpret a string representation. For more information, see [Format types](#).

In This Section

[Parsing Numeric Strings](#)

Describes how to convert strings into .NET numeric types.

[Parsing Date and Time Strings](#)

Describes how to convert strings into .NET `DateTime` types.

[Parsing Other Strings](#)

Describes how to convert strings into `Char`, `Boolean`, and `Enum` types.

Related Sections

[Formatting Types](#)

Describes basic formatting concepts like format specifiers and format providers.

[Type Conversion in .NET](#)

Describes how to convert types.

Parsing numeric strings in .NET

9/20/2022 • 8 minutes to read • [Edit Online](#)

All numeric types have two static parsing methods, `Parse` and `TryParse`, that you can use to convert the string representation of a number into a numeric type. These methods enable you to parse strings that were produced by using the format strings documented in [Standard Numeric Format Strings](#) and [Custom Numeric Format Strings](#). By default, the `Parse` and `TryParse` methods can successfully convert strings that contain integral decimal digits only to integer values. They can successfully convert strings that contain integral and fractional decimal digits, group separators, and a decimal separator to floating-point values. The `Parse` method throws an exception if the operation fails, whereas the `TryParse` method returns `false`.

NOTE

Starting in .NET 7, the numeric types in .NET also implement the `System.IParseable<TSelf>` interface, which defines the `IParseable<TSelf>.Parse` and `IParseable<TSelf>.TryParse` methods.

Parsing and format providers

Typically, the string representations of numeric values differ by culture. Elements of numeric strings, such as currency symbols, group (or thousands) separators, and decimal separators, all vary by culture. Parsing methods either implicitly or explicitly use a format provider that recognizes these culture-specific variations. If no format provider is specified in a call to the `Parse` or `TryParse` method, the format provider associated with the current culture (the `NumberFormatInfo` object returned by the `NumberFormatInfo.CurrentInfo` property) is used.

A format provider is represented by an `IFormatProvider` implementation. This interface has a single member, the `GetFormat` method, whose single parameter is a `Type` object that represents the type to be formatted. This method returns the object that provides formatting information. .NET supports the following two `IFormatProvider` implementations for parsing numeric strings:

- A `CultureInfo` object whose `CultureInfo.GetFormat` method returns a `NumberFormatInfo` object that provides culture-specific formatting information.
- A `NumberFormatInfo` object whose `NumberFormatInfo.GetFormat` method returns itself.

The following example tries to convert each string in an array to a `Double` value. It first tries to parse the string by using a format provider that reflects the conventions of the English (United States) culture. If this operation throws a `FormatException`, it tries to parse the string by using a format provider that reflects the conventions of the French (France) culture.

```
using System;
using System.Globalization;

public class Example
{
    public static void Main()
    {
        string[] values = { "1,304.16", "$1,456.78", "1,094", "152",
                            "123,45 €", "1 304,16", "Ae9f" };
        double number;
        CultureInfo culture = null;

        foreach (string value in values) {
            try {
                culture = CultureInfo.CreateSpecificCulture("en-US");
                number = Double.Parse(value, culture);
                Console.WriteLine("{0}: {1} --> {2}", culture.Name, value, number);
            }
            catch (FormatException) {
                Console.WriteLine("{0}: Unable to parse '{1}'.",
                                  culture.Name, value);
                culture = CultureInfo.CreateSpecificCulture("fr-FR");
                try {
                    number = Double.Parse(value, culture);
                    Console.WriteLine("{0}: {1} --> {2}", culture.Name, value, number);
                }
                catch (FormatException) {
                    Console.WriteLine("{0}: Unable to parse '{1}'.",
                                      culture.Name, value);
                }
            }
            Console.WriteLine();
        }
    }
}

// The example displays the following output:
//      en-US: 1,304.16 --> 1304.16
//
//      en-US: Unable to parse '$1,456.78'.
//      fr-FR: Unable to parse '$1,456.78'.
//
//      en-US: 1,094 --> 1094
//
//      en-US: 152 --> 152
//
//      en-US: Unable to parse '123,45 €'.
//      fr-FR: Unable to parse '123,45 €'.
//
//      en-US: Unable to parse '1 304,16'.
//      fr-FR: 1 304,16 --> 1304.16
//
//      en-US: Unable to parse 'Ae9f'.
//      fr-FR: Unable to parse 'Ae9f'.
```

```

Imports System.Globalization

Module Example
    Public Sub Main()
        Dim values() As String = {"1,304.16", "$1,456.78", "1,094", "152",
                                "123,45 €", "1 304,16", "Ae9f"}
        Dim number As Double
        Dim culture As CultureInfo = Nothing

        For Each value As String In values
            Try
                culture = CultureInfo.CreateSpecificCulture("en-US")
                number = Double.Parse(value, culture)
                Console.WriteLine("{0}: {1} --> {2}", culture.Name, value, number)
            Catch e As FormatException
                Console.WriteLine("{0}: Unable to parse '{1}'.",
                                 culture.Name, value)
                culture = CultureInfo.CreateSpecificCulture("fr-FR")
                Try
                    number = Double.Parse(value, culture)
                    Console.WriteLine("{0}: {1} --> {2}", culture.Name, value, number)
                Catch ex As FormatException
                    Console.WriteLine("{0}: Unable to parse '{1}'.",
                                     culture.Name, value)
                End Try
            End Try
        Next
    End Sub
End Module
' The example displays the following output:
' en-US: 1,304.16 --> 1304.16
'
' en-US: Unable to parse '$1,456.78'.
' fr-FR: Unable to parse '$1,456.78'.
'
' en-US: 1,094 --> 1094
'
' en-US: 152 --> 152
'
' en-US: Unable to parse '123,45 €'.
' fr-FR: Unable to parse '123,45 €'.
'
' en-US: Unable to parse '1 304,16'.
' fr-FR: 1 304,16 --> 1304.16
'
' en-US: Unable to parse 'Ae9f'.
' fr-FR: Unable to parse 'Ae9f'.

```

Parsing and NumberStyles Values

The style elements (such as white space, group separators, and decimal separator) that the parse operation can handle are defined by a [NumberStyles](#) enumeration value. By default, strings that represent integer values are parsed by using the [NumberStyles.Integer](#) value, which permits only numeric digits, leading and trailing white space, and a leading sign. Strings that represent floating-point values are parsed using a combination of the [NumberStyles.Float](#) and [NumberStyles.AllowThousands](#) values; this composite style permits decimal digits along with leading and trailing white space, a leading sign, a decimal separator, a group separator, and an exponent. By calling an overload of the [Parse](#) or [TryParse](#) method that includes a parameter of type [NumberStyles](#) and setting one or more [NumberStyles](#) flags, you can control the style elements that can be present in the string for the parse operation to succeed.

For example, a string that contains a group separator can't be converted to an [Int32](#) value by using the

`Int32.Parse(String)` method. However, the conversion succeeds if you use the `NumberStyles.AllowThousands` flag, as the following example illustrates.

```
using System;
using System.Globalization;

public class Example
{
    public static void Main()
    {
        string value = "1,304";
        int number;
        IFormatProvider provider = CultureInfo.CreateSpecificCulture("en-US");
        if (Int32.TryParse(value, out number))
            Console.WriteLine("{0} --> {1}", value, number);
        else
            Console.WriteLine("Unable to convert '{0}'", value);

        if (Int32.TryParse(value, NumberStyles.Integer | NumberStyles.AllowThousands,
                           provider, out number))
            Console.WriteLine("{0} --> {1}", value, number);
        else
            Console.WriteLine("Unable to convert '{0}'", value);
    }
}
// The example displays the following output:
//      Unable to convert '1,304'
//      1,304 --> 1304
```

```
Imports System.Globalization

Module Example
    Public Sub Main()
        Dim value As String = "1,304"
        Dim number As Integer
        Dim provider As IFormatProvider = CultureInfo.CreateSpecificCulture("en-US")
        If Int32.TryParse(value, number) Then
            Console.WriteLine("{0} --> {1}", value, number)
        Else
            Console.WriteLine("Unable to convert '{0}'", value)
        End If

        If Int32.TryParse(value, NumberStyles.Integer Or NumberStyles.AllowThousands,
                           provider, number) Then
            Console.WriteLine("{0} --> {1}", value, number)
        Else
            Console.WriteLine("Unable to convert '{0}'", value)
        End If
    End Sub
End Module
' The example displays the following output:
'      Unable to convert '1,304'
'      1,304 --> 1304
```

WARNING

The parse operation always uses the formatting conventions of a particular culture. If you do not specify a culture by passing a `CultureInfo` or `NumberFormatInfo` object, the culture associated with the current thread is used.

The following table lists the members of the `NumberStyles` enumeration and describes the effect that they have on the parsing operation.

NUMBERSTYLES VALUE	EFFECT ON THE STRING TO BE PARSED
NumberStyles.None	Only numeric digits are permitted.
NumberStyles.AllowDecimalPoint	The decimal separator and fractional digits are permitted. For integer values, only zero is permitted as a fractional digit. Valid decimal separators are determined by the NumberFormatInfo.NumberDecimalSeparator or NumberFormatInfo.CurrencyDecimalSeparator property.
NumberStyles.AllowExponent	The "e" or "E" character can be used to indicate exponential notation. For additional information, see NumberStyles .
NumberStyles.AllowLeadingWhite	Leading white space is permitted.
NumberStyles.AllowTrailingWhite	Trailing white space is permitted.
NumberStyles.AllowLeadingSign	A positive or negative sign can precede numeric digits.
NumberStyles.AllowTrailingSign	A positive or negative sign can follow numeric digits.
NumberStyles.AllowParentheses	Parentheses can be used to indicate negative values.
NumberStyles.AllowThousands	The group separator is permitted. The group separator character is determined by the NumberFormatInfo.NumberGroupSeparator or NumberFormatInfo.CurrencyGroupSeparator property.
NumberStyles.AllowCurrencySymbol	The currency symbol is permitted. The currency symbol is defined by the NumberFormatInfo.CurrencySymbol property.
NumberStyles.AllowHexSpecifier	The string to be parsed is interpreted as a hexadecimal number. It can include the hexadecimal digits 0-9, A-F, and a-f. This flag can be used only to parse integer values.

In addition, the [NumberStyles](#) enumeration provides the following composite styles, which include multiple [NumberStyles](#) flags.

COMPOSITE NUMBERSTYLES VALUE	INCLUDES MEMBERS
NumberStyles.Integer	Includes the NumberStyles.AllowLeadingWhite , NumberStyles.AllowTrailingWhite , and NumberStyles.AllowLeadingSign styles. This is the default style used to parse integer values.
NumberStyles.Number	Includes the NumberStyles.AllowLeadingWhite , NumberStyles.AllowTrailingWhite , NumberStyles.AllowLeadingSign , NumberStyles.AllowTrailingSign , NumberStyles.AllowDecimalPoint , and NumberStyles.AllowThousands styles.

COMPOSITE NUMBERSTYLES VALUE	INCLUDES MEMBERS
NumberStyles.Float	Includes the NumberStyles.AllowLeadingWhite , NumberStyles.AllowTrailingWhite , NumberStyles.AllowLeadingSign , NumberStyles.AllowDecimalPoint , and NumberStyles.AllowExponent styles.
NumberStyles.Currency	Includes all styles except NumberStyles.AllowExponent and NumberStyles.AllowHexSpecifier .
NumberStyles.Any	Includes all styles except NumberStyles.AllowHexSpecifier .
NumberStyles.HexNumber	Includes the NumberStyles.AllowLeadingWhite , NumberStyles.AllowTrailingWhite , and NumberStyles.AllowHexSpecifier styles.

Parsing and Unicode Digits

The Unicode standard defines code points for digits in various writing systems. For example, code points from U+0030 to U+0039 represent the basic Latin digits 0 through 9, code points from U+09E6 to U+09EF represent the Bangla digits 0 through 9, and code points from U+FF10 to U+FF19 represent the Fullwidth digits 0 through 9. However, the only numeric digits recognized by parsing methods are the basic Latin digits 0-9 with code points from U+0030 to U+0039. If a numeric parsing method is passed a string that contains any other digits, the method throws a [FormatException](#).

The following example uses the [Int32.Parse](#) method to parse strings that consist of digits in different writing systems. As the output from the example shows, the attempt to parse the basic Latin digits succeeds, but the attempt to parse the Fullwidth, Arabic-Indic, and Bangla digits fails.

```
using System;

public class Example
{
    public static void Main()
    {
        string value;
        // Define a string of basic Latin digits 1-5.
        value = "\u0031\u0032\u0033\u0034\u0035";
        ParseDigits(value);

        // Define a string of Fullwidth digits 1-5.
        value = "\uFF11\uFF12\uFF13\uFF14\uFF15";
        ParseDigits(value);

        // Define a string of Arabic-Indic digits 1-5.
        value = "\u0661\u0662\u0663\u0664\u0665";
        ParseDigits(value);

        // Define a string of Bangla digits 1-5.
        value = "\u09e7\u09e8\u09e9\u09ea\u09eb";
        ParseDigits(value);
    }

    static void ParseDigits(string value)
    {
        try {
            int number = Int32.Parse(value);
            Console.WriteLine("{0} --> {1}", value, number);
        }
        catch (FormatException) {
            Console.WriteLine("Unable to parse '{0}'.", value);
        }
    }
}

// The example displays the following output:
//      '12345' --> 12345
//      Unable to parse '12345'.
//      Unable to parse '\u09e7'.
//      Unable to parse '\u09e8'.
//      Unable to parse '\u09e9'.
//      Unable to parse '\u09ea'.
//      Unable to parse '\u09eb'.
```

```

Module Example
    Public Sub Main()
        Dim value As String
        ' Define a string of basic Latin digits 1-5.
        value = ChrW(&h31) + ChrW(&h32) + ChrW(&h33) + ChrW(&h34) + ChrW(&h35)
        ParseDigits(value)

        ' Define a string of Fullwidth digits 1-5.
        value = ChrW(&hff11) + ChrW(&hff12) + ChrW(&hff13) + ChrW(&hff14) + ChrW(&hff15)
        ParseDigits(value)

        ' Define a string of Arabic-Indic digits 1-5.
        value = ChrW(&h661) + ChrW(&h662) + ChrW(&h663) + ChrW(&h664) + ChrW(&h665)
        ParseDigits(value)

        ' Define a string of Bangla digits 1-5.
        value = ChrW(&h09e7) + ChrW(&h09e8) + ChrW(&h09e9) + ChrW(&h09ea) + ChrW(&h09eb)
        ParseDigits(value)
    End Sub

    Sub ParseDigits(value As String)
        Try
            Dim number As Integer = Int32.Parse(value)
            Console.WriteLine("'{0}' --> {1}", value, number)
        Catch e As FormatException
            Console.WriteLine("Unable to parse '{0}'.", value)
        End Try
    End Sub
End Module
' The example displays the following output:
'      '12345' --> 12345
'      Unable to parse '12345'.
'      Unable to parse 'ଠୟ୍ୟୋ'.
'      Unable to parse '■■■■■'.

```

See also

- [NumberStyles](#)
- [Parsing Strings](#)
- [Formatting Types](#)

Parse date and time strings in .NET

9/20/2022 • 7 minutes to read • [Edit Online](#)

Parsing strings to convert them to [DateTime](#) objects requires you to specify information about how the dates and times are represented as text. Different cultures use different orders for day, month, and year. Some time representations use a 24-hour clock, others specify "AM" and "PM." Some applications need only the date. Others need only the time. Still others need to specify both the date and the time. The methods that convert strings to [DateTime](#) objects enable you to provide detailed information about the formats you expect and the elements of a date and time your application needs. There are three subtasks to correctly converting text into a [DateTime](#):

1. You must specify the expected format of the text representing a date and time.
2. You may specify the culture for the format of a date time.
3. You may specify how missing components in the text representation are set in the date and time.

The [Parse](#) and [TryParse](#) methods convert many common representations of a date and time. The [ParseExact](#) and [TryParseExact](#) methods convert a string representation that conforms to the pattern specified by a date and time format string. (See the articles on [standard date and time format strings](#) and [custom date and time format strings](#) for details.)

The current [DateTimeFormatInfo](#) object provides more control over how text should be interpreted as a date and time. Properties of a [DateTimeFormatInfo](#) describe the date and time separators, the names of months, days, and eras, and the format for the "AM" and "PM" designations. The [CultureInfo](#) returned by [CultureInfo.CurrentCulture](#) has a [CultureInfo.DateTimeFormat](#) property that represents the current culture. If you want a specific culture or custom settings, you specify the [IFormatProvider](#) parameter of a parsing method. For the [IFormatProvider](#) parameter, specify a [CultureInfo](#) object, which represents a culture, or a [DateTimeFormatInfo](#) object.

The text representing a date or time may be missing some information. For example, most people would assume the date "March 12" represents the current year. Similarly, "March 2018" represents the month of March in the year 2018. Text representing time often does only includes hours, minutes, and an AM/PM designation. Parsing methods handle this missing information by using reasonable defaults:

- When only the time is present, the date portion uses the current date.
- When only the date is present, the time portion is midnight.
- When the year isn't specified in a date, the current year is used.
- When the day of the month isn't specified, the first of the month is used.

If the date is present in the string, it must include the month and one of the day or year. If the time is present, it must include the hour, and either the minutes or the AM/PM designator.

You can specify the [NoCurrentDateDefault](#) constant to override these defaults. When you use that constant, any missing year, month, or day properties are set to the value [1](#). The [last example](#) using [Parse](#) demonstrates this behavior.

In addition to a date and a time component, the string representation of a date and time can include an offset that indicates how much the time differs from Coordinated Universal Time (UTC). For example, the string "2/14/2007 5:32:00 -7:00" defines a time that is seven hours earlier than UTC. If an offset is omitted from the string representation of a time, parsing returns a [DateTime](#) object with its [Kind](#) property set to [DateTimeKind.Unspecified](#). If an offset is specified, parsing returns a [DateTime](#) object with its [Kind](#) property set to [DateTimeKind.Local](#) and its value adjusted to the local time zone of your machine. You can modify this

behavior by using a [DateTimeStyles](#) value with the parsing method.

The format provider is also used to interpret an ambiguous numeric date. It is not clear which components of the date represented by the string "02/03/04" are the month, day, and year. The components are interpreted according to the order of similar date formats in the format provider.

Parse

The following example illustrates the use of the [DateTime.Parse](#) method to convert a `string` into a [DateTime](#).

This example uses the culture associated with the current thread. If the [CultureInfo](#) associated with the current culture cannot parse the input string, a [FormatException](#) is thrown.

TIP

All the C# samples in this article run in your browser. Press the [Run](#) button to see the output. You can also edit them to experiment yourself.

NOTE

These examples are available in the GitHub docs repo for both [C#](#) and [Visual Basic](#).

```
string dateInput = "Jan 1, 2009";
var parsedDate = DateTime.Parse(dateInput);
Console.WriteLine(parsedDate);
// Displays the following output on a system whose culture is en-US:
//      1/1/2009 00:00:00
```

```
Dim MyString As String = "Jan 1, 2009"
Dim MyDateTime As DateTime = DateTime.Parse(MyString)
Console.WriteLine(MyDateTime)
' Displays the following output on a system whose culture is en-US:
'      1/1/2009 00:00:00
```

You can also explicitly define the culture whose formatting conventions are used when you parse a string. You specify one of the standard [DateTimeFormatInfo](#) objects returned by the [CultureInfo.DateTimeFormat](#) property. The following example uses a format provider to parse a German string into a [DateTime](#). It creates a [CultureInfo](#) representing the `de-DE` culture. That `CultureInfo` object ensures successful parsing of this particular string. This precludes whatever setting is in the [CurrentCulture](#) of the [CurrentThread](#).

```
var cultureInfo = new CultureInfo("de-DE");
string dateString = "12 Juni 2008";
var dateTime = DateTime.Parse(dateString, cultureInfo);
Console.WriteLine(dateTime);
// The example displays the following output:
//      6/12/2008 00:00:00
```

```
Dim MyCultureInfo As New CultureInfo("de-DE")
Dim MyString As String = "12 Juni 2008"
Dim MyDateTime As DateTime = DateTime.Parse(MyString, MyCultureInfo)
Console.WriteLine(MyDateTime)
' The example displays the following output:
'      6/12/2008 00:00:00
```

However, although you can use overloads of the [Parse](#) method to specify custom format providers, the method does not support parsing non-standard formats. To parse a date and time expressed in a non-standard format, use the [ParseExact](#) method instead.

The following example uses the [DateTimeStyles](#) enumeration to specify that the current date and time information should not be added to the [DateTime](#) for unspecified fields.

```
var cultureInfo = new CultureInfo("de-DE");
string dateString = "12 Juni 2008";
var dateTime = DateTime.Parse(dateString, cultureInfo,
                             DateTimeStyles.NoCurrentDateDefault);
Console.WriteLine(dateTime);
// The example displays the following output if the current culture is en-US:
//      6/12/2008 00:00:00
```

```
Dim MyCultureInfo As New CultureInfo("de-DE")
Dim MyString As String = "12 Juni 2008"
Dim MyDateTime As DateTime = DateTime.Parse(MyString, MyCultureInfo,
                                             DateTimeStyles.NoCurrentDateDefault)
Console.WriteLine(MyDateTime)
' The example displays the following output if the current culture is en-US:
'      6/12/2008 00:00:00
```

ParseExact

The [DateTime.ParseExact](#) method converts a string to a [DateTime](#) object if it conforms to one of the specified string patterns. When a string that is not one of the forms specified is passed to this method, a [FormatException](#) is thrown. You can specify one of the standard date and time format specifiers or a combination of the custom format specifiers. Using the custom format specifiers, it is possible for you to construct a custom recognition string. For an explanation of the specifiers, see the topics on [standard date and time format strings](#) and [custom date and time format strings](#).

In the following example, the [DateTime.ParseExact](#) method is passed a string object to parse, followed by a format specifier, followed by a [CultureInfo](#) object. This [ParseExact](#) method can only parse strings that follow the long date pattern in the `en-US` culture.

```
var cultureInfo = new CultureInfo("en-US");
string[] dateStrings = { " Friday, April 10, 2009", "Friday, April 10, 2009" };
foreach (string dateString in dateStrings)
{
    try
    {
        var dateTime = DateTime.ParseExact(dateString, "D", cultureInfo);
        Console.WriteLine(dateTime);
    }
    catch (FormatException)
    {
        Console.WriteLine("Unable to parse '{0}'", dateString);
    }
}
// The example displays the following output:
//      Unable to parse ' Friday, April 10, 2009'
//      4/10/2009 00:00:00
```

```
Dim MyCultureInfo As New CultureInfo("en-US")
Dim MyString() As String = {" Friday, April 10, 2009", "Friday, April 10, 2009"}
For Each dateString As String In MyString
    Try
        Dim MyDateTime As DateTime = DateTime.ParseExact(dateString, "D",
                                                       MyCultureInfo)
        Console.WriteLine(MyDateTime)
    Catch e As FormatException
        Console.WriteLine("Unable to parse '{0}'", dateString)
    End Try
Next
' The example displays the following output:
'     Unable to parse ' Friday, April 10, 2009'
'     4/10/2009 00:00:00
```

Each overload of the [Parse](#) and [ParseExact](#) methods also has an [IFormatProvider](#) parameter that provides culture-specific information about the formatting of the string. This [IFormatProvider](#) object is a [CultureInfo](#) object that represents a standard culture or a [DateTimeFormatInfo](#) object that is returned by the [CultureInfo.DateTimeFormat](#) property. [ParseExact](#) also uses an additional string or string array argument that defines one or more custom date and time formats.

See also

- [Parsing strings](#)
- [Formatting types](#)
- [Type conversion in .NET](#)
- [Standard date and time formats](#)
- [Custom date and time format strings](#)

Parsing Other Strings in .NET

9/20/2022 • 2 minutes to read • [Edit Online](#)

In addition to numeric and [DateTime](#) strings, you can also parse strings that represent the types [Char](#), [Boolean](#), and [Enum](#) into data types.

Char

The static `Parse` method associated with the **Char** data type is useful for converting a string that contains a single character into its Unicode value. The following code example parses a string into a Unicode character.

```
String^ MyString1 = "A";
char MyChar = Char::Parse(MyString1);
// MyChar now contains a Unicode "A" character.
```

```
string MyString1 = "A";
char MyChar = Char.Parse(MyString1);
// MyChar now contains a Unicode "A" character.
```

```
Dim MyString1 As String = "A"
Dim MyChar As Char = Char.Parse(MyString1)
' MyChar now contains a Unicode "A" character.
```

Boolean

The **Boolean** data type contains a `Parse` method that you can use to convert a string that represents a Boolean value into an actual **Boolean** type. This method is not case-sensitive and can successfully parse a string containing "True" or "False." The `Parse` method associated with the **Boolean** type can also parse strings that are surrounded by white spaces. If any other string is passed, a [FormatException](#) is thrown.

The following code example uses the `Parse` method to convert a string into a Boolean value.

```
String^ MyString2 = "True";
bool MyBool = bool::Parse(MyString2);
// MyBool now contains a True Boolean value.
```

```
string MyString2 = "True";
bool MyBool = bool.Parse(MyString2);
// MyBool now contains a True Boolean value.
```

```
Dim MyString2 As String = "True"
Dim MyBool As Boolean = Boolean.Parse(MyString2)
' MyBool now contains a True Boolean value.
```

Enumeration

You can use the static `Parse` method to initialize an enumeration type to the value of a string. This method

accepts the enumeration type you are parsing, the string to parse, and an optional Boolean flag indicating whether or not the parse is case-sensitive. The string you are parsing can contain several values separated by commas, which can be preceded or followed by one or more empty spaces (also called white spaces). When the string contains multiple values, the value of the returned object is the value of all specified values combined with a bitwise OR operation.

The following example uses the **Parse** method to convert a string representation into an enumeration value. The **DayOfWeek** enumeration is initialized to **Thursday** from a string.

```
String^ MyString3 = "Thursday";
DayOfWeek MyDays = (DayOfWeek)Enum::Parse(DayOfWeek::typeid, MyString3);
Console::WriteLine(MyDays);
// The result is Thursday.
```

```
string MyString3 = "Thursday";
DayOfWeek MyDays = (DayOfWeek)Enum.Parse(typeof(DayOfWeek), MyString3);
Console.WriteLine(MyDays);
// The result is Thursday.
```

```
Dim MyString3 As String = "Thursday"
Dim MyDays As DayOfWeek = CType([Enum].Parse(GetType(DayOfWeek), MyString3), DayOfWeek)
Console.WriteLine("{0:G}", MyDays)
' The result is Thursday.
```

See also

- [Parsing Strings](#)
- [Formatting Types](#)
- [Type Conversion in .NET](#)

.NET regular expressions

9/20/2022 • 10 minutes to read • [Edit Online](#)

Regular expressions provide a powerful, flexible, and efficient method for processing text. The extensive pattern-matching notation of regular expressions enables you to quickly parse large amounts of text to:

- Find specific character patterns.
- Validate text to ensure that it matches a predefined pattern (such as an email address).
- Extract, edit, replace, or delete text substrings.
- Add extracted strings to a collection in order to generate a report.

For many applications that deal with strings or that parse large blocks of text, regular expressions are an indispensable tool.

How regular expressions work

The centerpiece of text processing with regular expressions is the regular expression engine, which is represented by the [System.Text.RegularExpressions.Regex](#) object in .NET. At a minimum, processing text using regular expressions requires that the regular expression engine be provided with the following two items of information:

- The regular expression pattern to identify in the text.

In .NET, regular expression patterns are defined by a special syntax or language, which is compatible with Perl 5 regular expressions and adds some additional features such as right-to-left matching. For more information, see [Regular Expression Language - Quick Reference](#).

- The text to parse for the regular expression pattern.

The methods of the [Regex](#) class let you perform the following operations:

- Determine whether the regular expression pattern occurs in the input text by calling the [Regex.IsMatch](#) method. For an example that uses the [IsMatch](#) method for validating text, see [How to: Verify that Strings Are in Valid Email Format](#).
- Retrieve one or all occurrences of text that matches the regular expression pattern by calling the [Regex.Match](#) or [Regex.Matches](#) method. The former method returns a [System.Text.RegularExpressions.Match](#) object that provides information about the matching text. The latter returns a [MatchCollection](#) object that contains one [System.Text.RegularExpressions.Match](#) object for each match found in the parsed text.
- Replace text that matches the regular expression pattern by calling the [Regex.Replace](#) method. For examples that use the [Replace](#) method to change date formats and remove invalid characters from a string, see [How to: Strip Invalid Characters from a String](#) and [Example: Changing Date Formats](#).

For an overview of the regular expression object model, see [The Regular Expression Object Model](#).

For more information about the regular expression language, see [Regular Expression Language - Quick Reference](#) or download and print one of the following brochures:

- [Quick Reference in Word \(.docx\) format](#)
- [Quick Reference in PDF \(.pdf\) format](#)

Regular expression examples

The [String](#) class includes string search and replacement methods that you can use when you want to locate literal strings in a larger string. Regular expressions are most useful either when you want to locate one of several substrings in a larger string, or when you want to identify patterns in a string, as the following examples illustrate.

WARNING

When using [System.Text.RegularExpressions](#) to process untrusted input, pass a timeout. A malicious user can provide input to `RegularExpression` causing a Denial-of-Service attack. ASP.NET Core framework APIs that use `RegularExpression` pass a timeout.

TIP

The [System.Web.RegularExpressions](#) namespace contains a number of regular expression objects that implement predefined regular expression patterns for parsing strings from HTML, XML, and ASP.NET documents. For example, the [TagRegex](#) class identifies start tags in a string, and the [CommentRegex](#) class identifies ASP.NET comments in a string.

Example 1: Replace substrings

Assume that a mailing list contains names that sometimes include a title (Mr., Mrs., Miss, or Ms.) along with a first and last name. Suppose you don't want to include the titles when you generate envelope labels from the list. In that case, you can use a regular expression to remove the titles, as the following example illustrates:

```
using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string pattern = "(Mr\\\\.? |Mrs\\\\.? |Miss |Ms\\\\.? )";
        string[] names = { "Mr. Henry Hunt", "Ms. Sara Samuels",
                           "Abraham Adams", "Ms. Nicole Norris" };
        foreach (string name in names)
            Console.WriteLine(Regex.Replace(name, pattern, String.Empty));
    }
}
// The example displays the following output:
//   Henry Hunt
//   Sara Samuels
//   Abraham Adams
//   Nicole Norris
```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim pattern As String = "(Mr\.\.? |Mrs\.\.? |Miss |Ms\.\.? )"
        Dim names() As String = {"Mr. Henry Hunt", "Ms. Sara Samuels", _
                               "Abraham Adams", "Ms. Nicole Norris"}
        For Each name As String In names
            Console.WriteLine(Regex.Replace(name, pattern, String.Empty))
        Next
    End Sub
End Module
' The example displays the following output:
'   Henry Hunt
'   Sara Samuels
'   Abraham Adams
'   Nicole Norris

```

The regular expression pattern `(Mr\.\.? |Mrs\.\.? |Miss |Ms\.\.?)` matches any occurrence of "Mr ", "Mr.", "Mrs ", "Mrs.", "Miss ", "Ms or "Ms ". The call to the `Regex.Replace` method replaces the matched string with `String.Empty`; in other words, it removes it from the original string.

Example 2: Identify duplicated words

Accidentally duplicating words is a common error that writers make. Use a regular expression to identify duplicated words, as the following example shows:

```

using System;
using System.Text.RegularExpressions;

public class Class1
{
    public static void Main()
    {
        string pattern = @"\b(\w+?)\s\1\b";
        string input = "This this is a nice day. What about this? This tastes good. I saw a a dog.";
        foreach (Match match in Regex.Matches(input, pattern, RegexOptions.IgnoreCase))
            Console.WriteLine("{0} (duplicates '{1}') at position {2}",
                             match.Value, match.Groups[1].Value, match.Index);
    }
}
// The example displays the following output:
//   This this (duplicates 'This') at position 0
//   a a (duplicates 'a') at position 66

```

```

Imports System.Text.RegularExpressions

Module modMain
    Public Sub Main()
        Dim pattern As String = "\b(\w+?)\s\1\b"
        Dim input As String = "This this is a nice day. What about this? This tastes good. I saw a a dog."
        For Each match As Match In Regex.Matches(input, pattern, RegexOptions.IgnoreCase)
            Console.WriteLine("{0} (duplicates '{1}') at position {2}",
                             match.Value, match.Groups(1).Value, match.Index)
        Next
    End Sub
End Module
' The example displays the following output:
'   This this (duplicates 'This') at position 0
'   a a (duplicates 'a') at position 66

```

The regular expression pattern `\b(\w+?)\s\1\b` can be interpreted as follows:

PATTERN	INTERPRETATION
\b	Start at a word boundary.
(\w+?)	Match one or more word characters, but as few characters as possible. Together, they form a group that can be referred to as \1.
\s	Match a white-space character.
\1	Match the substring that's equal to the group named \1.
\b	Match a word boundary.

The [Regex.Matches](#) method is called with regular expression options set to [RegexOptions.IgnoreCase](#). Therefore, the match operation is case-insensitive, and the example identifies the substring "This this" as a duplication.

The input string includes the substring "this? This". However, because of the intervening punctuation mark, it isn't identified as a duplication.

Example 3: Dynamically build a culture-sensitive regular expression

The following example illustrates the power of regular expressions combined with the flexibility offered by .NET's globalization features. It uses the [NumberFormatInfo](#) object to determine the format of currency values in the system's current culture. It then uses that information to dynamically construct a regular expression that extracts currency values from the text. For each match, it extracts the subgroup that contains the numeric string only, converts it to a [Decimal](#) value, and calculates a running total.

```

using System;
using System.Collections.Generic;
using System.Globalization;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        // Define text to be parsed.
        string input = "Office expenses on 2/13/2008:\n" +
            "Paper (500 sheets)           $3.95\n" +
            "Pencils (box of 10)         $1.00\n" +
            "Pens (box of 10)            $4.49\n" +
            "Erasers                      $2.19\n" +
            "Ink jet printer              $69.95\n\n" +
            "Total Expenses                $ 81.58\n";

        // Get current culture's NumberFormatInfo object.
        NumberFormatInfo nfi = CultureInfo.CurrentCulture.NumberFormat;
        // Assign needed property values to variables.
        string currencySymbol = nfi.CurrencySymbol;
        bool symbolPrecedesIfPositive = nfi.CurrencyPositivePattern % 2 == 0;
        string groupSeparator = nfi.CurrencyGroupSeparator;
        string decimalSeparator = nfi.CurrencyDecimalSeparator;

        // Form regular expression pattern.
        string pattern = Regex.Escape( symbolPrecedesIfPositive ? currencySymbol : "" ) +
            @"\s*[-+]?([0-9]{0,3}(." + groupSeparator + "[0-9]{3})*(" +
            Regex.Escape(decimalSeparator) + "[0-9]+)?)" +
            (! symbolPrecedesIfPositive ? currencySymbol : "");

        Console.WriteLine( "The regular expression pattern is:" );
        Console.WriteLine( "    " + pattern );

        // Get text that matches regular expression pattern.
        MatchCollection matches = Regex.Matches(input, pattern,
            RegexOptions.IgnorePatternWhitespace);
        Console.WriteLine("Found {0} matches.", matches.Count);

        // Get numeric string, convert it to a value, and add it to List object.
        List<decimal> expenses = new List<Decimal>();

        foreach (Match match in matches)
            expenses.Add(Decimal.Parse(match.Groups[1].Value));

        // Determine whether total is present and if present, whether it is correct.
        decimal total = 0;
        foreach (decimal value in expenses)
            total += value;

        if (total / 2 == expenses[expenses.Count - 1])
            Console.WriteLine("The expenses total {0:C2}.", expenses[expenses.Count - 1]);
        else
            Console.WriteLine("The expenses total {0:C2}.", total);
    }
}

// The example displays the following output:
//      The regular expression pattern is:
//      \$\s*[-+]?([0-9]{0,3}(,[0-9]{3})*(\.[0-9]+)?)
//      Found 6 matches.
//      The expenses total $81.58.

```

```

Imports System.Collections.Generic
Imports System.Globalization
Imports System.Text.RegularExpressions

Public Module Example
    Public Sub Main()
        ' Define text to be parsed.
        Dim input As String = "Office expenses on 2/13/2008:" + vbCrLf + _
            "Paper (500 sheets)                 $3.95" + vbCrLf + _
            "Pencils (box of 10)               $1.00" + vbCrLf + _
            "Pens (box of 10)                  $4.49" + vbCrLf + _
            "Erasers                          $2.19" + vbCrLf + _
            "Ink jet printer                  $69.95" + vbCrLf + vbCrLf + _
            "Total Expenses                   $ 81.58" + vbCrLf

        ' Get current culture's NumberFormatInfo object.
        Dim nfi As NumberFormatInfo = CultureInfo.CurrentCulture.NumberFormat
        ' Assign needed property values to variables.
        Dim currencySymbol As String = nfi.CurrencySymbol
        Dim symbolPrecedesIfPositive As Boolean = CBool(nfi.CurrencyPositivePattern Mod 2 = 0)
        Dim groupSeparator As String = nfi.CurrencyGroupSeparator
        Dim decimalSeparator As String = nfi.CurrencyDecimalSeparator

        ' Form regular expression pattern.
        Dim pattern As String = Regex.Escape(CStr(IIf(symbolPrecedesIfPositive, currencySymbol, ""))) + _
            "\s*[-+]?([0-9]{0,3}([.,][0-9]{3})*([.,][0-9]+))?" + _
            Regex.Escape(decimalSeparator) + "[0-9]+)?)" + _
            CStr(IIf(Not symbolPrecedesIfPositive, currencySymbol, ""))
        Console.WriteLine("The regular expression pattern is: ")
        Console.WriteLine("    " + pattern)

        ' Get text that matches regular expression pattern.
        Dim matches As MatchCollection = Regex.Matches(input, pattern, RegexOptions.IgnorePatternWhitespace)
        Console.WriteLine("Found {0} matches. ", matches.Count)

        ' Get numeric string, convert it to a value, and add it to List object.
        Dim expenses As New List(Of Decimal)

        For Each match As Match In matches
            expenses.Add(Decimal.Parse(match.Groups.Item(1).Value))
        Next

        ' Determine whether total is present and if present, whether it is correct.
        Dim total As Decimal
        For Each value As Decimal In expenses
            total += value
        Next

        If total / 2 = expenses(expenses.Count - 1) Then
            Console.WriteLine("The expenses total {0:C2}.", expenses(expenses.Count - 1))
        Else
            Console.WriteLine("The expenses total {0:C2}.", total)
        End If
    End Sub
End Module
' The example displays the following output:
'     The regular expression pattern is:
'     \$.s*[-+]?([0-9]{0,3}([.,][0-9]{3})*([.,][0-9]+))?
'     Found 6 matches.
'     The expenses total $81.58.

```

On a computer whose current culture is English - United States (en-US), the example dynamically builds the regular expression `\$.s*[-+]?([0-9]{0,3}([.,][0-9]{3})*([.,][0-9]+))?`. This regular expression pattern can be interpreted as follows:

PATTERN	INTERPRETATION
\\$	Look for a single occurrence of the dollar symbol (\$) in the input string. The regular expression pattern string includes a backslash to indicate that the dollar symbol is to be interpreted literally rather than as a regular expression anchor. The \$ symbol alone would indicate that the regular expression engine should try to begin its match at the end of a string. To ensure that the current culture's currency symbol isn't misinterpreted as a regular expression symbol, the example calls the Regex.Escape method to escape the character.
\s*	Look for zero or more occurrences of a white-space character.
[-+]?	Look for zero or one occurrence of either a positive or negative sign.
([0-9]{0,3}(,[0-9]{3})*(.[0-9]+)?)	The outer parentheses define this expression as a capturing group or a subexpression. If a match is found, information about this part of the matching string can be retrieved from the second Group object in the GroupCollection object returned by the Match.Groups property. The first element in the collection represents the entire match.
[0-9]{0,3}	Look for zero to three occurrences of the decimal digits 0 through 9.
(,[0-9]{3})*	Look for zero or more occurrences of a group separator followed by three decimal digits.
\.	Look for a single occurrence of the decimal separator.
[0-9]+	Look for one or more decimal digits.
(.[0-9]+)?	Look for zero or one occurrence of the decimal separator followed by at least one decimal digit.

If each subpattern is found in the input string, the match succeeds, and a [Match](#) object that contains information about the match is added to the [MatchCollection](#) object.

Related articles

TITLE	DESCRIPTION
Regular Expression Language - Quick Reference	Provides information on the set of characters, operators, and constructs that you can use to define regular expressions.
The Regular Expression Object Model	Provides information and code examples that illustrate how to use the regular expression classes.
Details of Regular Expression Behavior	Provides information about the capabilities and behavior of .NET regular expressions.
Use regular expressions in Visual Studio	

Reference

- [System.Text.RegularExpressions](#)
- [System.Text.RegularExpressions.Regex](#)
- [Regular Expressions - Quick Reference \(download in Word format\)](#)
- [Regular Expressions - Quick Reference \(download in PDF format\)](#)

Regular Expression Language - Quick Reference

9/20/2022 • 12 minutes to read • [Edit Online](#)

A regular expression is a pattern that the regular expression engine attempts to match in input text. A pattern consists of one or more character literals, operators, or constructs. For a brief introduction, see [.NET Regular Expressions](#).

Each section in this quick reference lists a particular category of characters, operators, and constructs that you can use to define regular expressions.

We've also provided this information in two formats that you can download and print for easy reference:

- [Download in Word \(.docx\) format](#)
- [Download in PDF \(.pdf\) format](#)

Character Escapes

The backslash character (\) in a regular expression indicates that the character that follows it either is a special character (as shown in the following table), or should be interpreted literally. For more information, see [Character Escapes](#).

ESCAPED CHARACTER	DESCRIPTION	PATTERN	MATCHES
\a	Matches a bell character, \u0007.	\a	"\u0007" in "Error!" + '\u0007'
\b	In a character class, matches a backspace, \u0008.	[\b]{3,}	"\b\b\b\b" in "\b\b\b\b"
\t	Matches a tab, \u0009.	(\w+)\t	"item1\t", "item2\t" in "item1\item2\t"
\r	Matches a carriage return, \u000D. (\r is not equivalent to the newline character, \n.)	\r\n(\w+)	"\r\nThese" in "\r\nThese are\ntwo lines."
\v	Matches a vertical tab, \u000B.	[\v]{2,}	"\v\v\v" in "\v\v\v"
\f	Matches a form feed, \u000C.	[\f]{2,}	"\f\f\f" in "\f\f\f"
\n	Matches a new line, \u000A.	\r\n(\w+)	"\r\nThese" in "\r\nThese are\ntwo lines."
\e	Matches an escape, \u001B.	\e	"\x001B" in "\x001B"

ESCAPED CHARACTER	DESCRIPTION	PATTERN	MATCHES
<code>\ nnn</code>	Uses octal representation to specify a character (<i>nnn</i> consists of two or three digits).	<code>\w\040\w</code>	"a b" , "c d" in "a bc d"
<code>\x nn</code>	Uses hexadecimal representation to specify a character (<i>nn</i> consists of exactly two digits).	<code>\w\x20\w</code>	"a b" , "c d" in "a bc d"
<code>\c X</code> <code>\c x</code>	Matches the ASCII control character that is specified by <i>X</i> or <i>x</i> , where <i>X</i> or <i>x</i> is the letter of the control character.	<code>\cC</code>	"\x0003" in "\x0003" (Ctrl-C)
<code>\u nnnn</code>	Matches a Unicode character by using hexadecimal representation (exactly four digits, as represented by <i>nnnn</i>).	<code>\w\u0020\w</code>	"a b" , "c d" in "a bc d"
<code>\</code>	When followed by a character that is not recognized as an escaped character in this and other tables in this topic, matches that character. For example, <code>*</code> is the same as <code>\x2A</code> , and <code>\.</code> is the same as <code>\x2E</code> . This allows the regular expression engine to disambiguate language elements (such as * or ?) and character literals (represented by <code>*</code> or <code>\?</code>).	<code>\d+[\+-\x*]\d+</code>	"2+2" and "3*9" in "(2+2) * 3*9"

Character Classes

A character class matches any one of a set of characters. Character classes include the language elements listed in the following table. For more information, see [Character Classes](#).

CHARACTER CLASS	DESCRIPTION	PATTERN	MATCHES
<code>[character_group]</code>	Matches any single character in <i>character_group</i> . By default, the match is case-sensitive.	<code>[ae]</code>	"a" in "gray" "a" , "e" in "lane"
<code>[^ character_group]</code>	Negation: Matches any single character that is not in <i>character_group</i> . By default, characters in <i>character_group</i> are case-sensitive.	<code>[^aei]</code>	"r" , "g" , "n" in "reign"

CHARACTER CLASS	DESCRIPTION	PATTERN	MATCHES
[first - last]	Character range: Matches any single character in the range from <i>first</i> to <i>last</i> .	[A-Z]	"A" , "B" in "AB123"
.	Wildcard: Matches any single character except \n . To match a literal period character (. or \u002E), you must precede it with the escape character (\.).	a.e	"ave" in "nave" "ate" in "water"
\p{ name }	Matches any single character in the Unicode general category or named block specified by <i>name</i> .	\p{Lu} \p{IsCyrillic}	"C" , "L" in "City Lights" "Д" , "Х" in "дЖем"
\P{ name }	Matches any single character that is not in the Unicode general category or named block specified by <i>name</i> .	\P{Lu} \P{IsCyrillic}	"i" , "t" , "y" in "City" "e" , "m" in "дЖем"
\w	Matches any word character .	\w	"I" , "D" , "A" , "1" , "3" in "ID A1.3"
\W	Matches any non-word character .	\W	" " , "." in "ID A1.3"
\s	Matches any white-space character .	\w\s	"D " in "ID A1.3"
\S	Matches any non-white-space character .	\s\S	"_" in "int __ctr"
\d	Matches any decimal digit .	\d	"4" in "4 = IV"
\D	Matches any character other than a decimal digit .	\D	" " , "=" , " " , "I" , "v" in "4 = IV"

Anchors

Anchors, or atomic zero-width assertions, cause a match to succeed or fail depending on the current position in the string, but they do not cause the engine to advance through the string or consume characters. The metacharacters listed in the following table are anchors. For more information, see [Anchors](#).

ASSERTION	DESCRIPTION	PATTERN	MATCHES
^	By default, the match must start at the beginning of the string; in multiline mode, it must start at the beginning of the line.	^\d{3}	"901" in "901-333-"

ASSERTION	DESCRIPTION	PATTERN	MATCHES
\$	By default, the match must occur at the end of the string or before <code>\n</code> at the end of the string; in multiline mode, it must occur before the end of the line or before <code>\n</code> at the end of the line.	<code>-\d{3}\$</code>	"-333" in "-901-333"
\A	The match must occur at the start of the string.	<code>\A\d{3}</code>	"901" in "901-333-"
\z	The match must occur at the end of the string or before <code>\n</code> at the end of the string.	<code>-\d{3}\z</code>	"-333" in "-901-333"
\z	The match must occur at the end of the string.	<code>-\d{3}\z</code>	"-333" in "-901-333"
\G	The match must occur at the point where the previous match ended, or if there was no previous match, at the position in the string where matching started.	<code>\G(\d\)</code>	"(1)" , "(3)" , "(5)" in "(1)(3)(5)[7](9)"
\b	The match must occur on a boundary between a <code>\w</code> (alphanumeric) and a <code>\W</code> (nonalphanumeric) character.	<code>\b\w+\s\w+\b</code>	"them theme" , "them them" in "them theme them them"
\B	The match must not occur on a <code>\b</code> boundary.	<code>\Bend\w*\b</code>	"ends" , "ender" in "end sends endure lender"

Grouping Constructs

Grouping constructs delineate subexpressions of a regular expression and typically capture substrings of an input string. Grouping constructs include the language elements listed in the following table. For more information, see [Grouping Constructs](#).

GROUPING CONSTRUCT	DESCRIPTION	PATTERN	MATCHES
(<code>subexpression</code>)	Captures the matched subexpression and assigns it a one-based ordinal number.	<code>(\w)\1</code>	"ee" in "deep"

GROUPING CONSTRUCT	DESCRIPTION	PATTERN	MATCHES
(?< <i>name</i> > <i>subexpression</i>) or (?' <i>name</i> ' <i>subexpression</i>)	Captures the matched subexpression into a named group.	(? <i>double</i>)\w)\k<double>	"ee" in "deep"
(?< <i>name1</i> - <i>name2</i> > <i>subexpression</i>) or (?' <i>name1</i> - <i>name2</i> ' <i>subexpression</i>)	Defines a balancing group definition. For more information, see the "Balancing Group Definition" section in Grouping Constructs .	((('Open')()[^(\())*]+('?Close-Open'))[^(\())]*+)*((Open)(?!))\$	"((1-3)*(3-1))" in "3+2^((1-3)*(3-1))"
(?: <i>subexpression</i>)	Defines a noncapturing group.	Write(?:Line)?	"WriteLine" in "Console.WriteLine()" "Write" in "Console.WriteLine(value)"
(?imsx-imnsx: <i>subexpression</i>)	Applies or disables the specified options within <i>subexpression</i> . For more information, see Regular Expression Options .	A\d{2}(?i:\w+)\b	"A12x1" , "A12XL" in "A12x1 A12XL a12x1"
(?= <i>subexpression</i>)	Zero-width positive lookahead assertion.	\b\w+\b(?=.=and.+)	"cats" , "dogs" in "cats, dogs and some mice."
(?! <i>subexpression</i>)	Zero-width negative lookahead assertion.	\b\w+\b(?!=.=and.+)	"and" , "some" , "mice" in "cats, dogs and some mice."
(?<= <i>subexpression</i>)	Zero-width positive lookbehind assertion.	\b\w+\b(?<=.+and.+) \b\w+\b(?<=.+and.*)	"some" , "mice" in "cats, dogs and some mice." "and" , "some" , "mice" in "cats, dogs and some mice."
(?<! <i>subexpression</i>)	Zero-width negative lookbehind assertion.	\b\w+\b(?<! .=and.+) \b\w+\b(?<! .=and.*)	"cats" , "dogs" , "and" in "cats, dogs and some mice." "cats" , "dogs" in "cats, dogs and some mice."

GROUPING CONSTRUCT	DESCRIPTION	PATTERN	MATCHES
<code>(?> subexpression)</code>	Atomic group.	<code>(?>a ab)c</code>	"ac" in "ac" nothing in "abc"

Lookarounds at a glance

When the regular expression engine hits a **lookaround expression**, it takes a substring reaching from the current position to the start (lookbehind) or end (lookahead) of the original string, and then runs `Regex.IsMatch` on that substring using the lookaround pattern. Success of this subexpression's result is then determined by whether it's a positive or negative assertion.

LOOKAROUND	NAME	FUNCTION
<code>(?=check)</code>	Positive Lookahead	Asserts that what immediately follows the current position in the string is "check"
<code>(?<=check)</code>	Positive Lookbehind	Asserts that what immediately precedes the current position in the string is "check"
<code>(?!check)</code>	Negative Lookahead	Asserts that what immediately follows the current position in the string is not "check"
<code>(?<!check)</code>	Negative Lookbehind	Asserts that what immediately precedes the current position in the string is not "check"

Once they have matched, **atomic groups** won't be re-evaluated again, even when the remainder of the pattern fails due to the match. This can significantly improve performance when quantifiers occur within the atomic group or the remainder of the pattern.

Quantifiers

A quantifier specifies how many instances of the previous element (which can be a character, a group, or a character class) must be present in the input string for a match to occur. Quantifiers include the language elements listed in the following table. For more information, see [Quantifiers](#).

QUANTIFIER	DESCRIPTION	PATTERN	MATCHES
<code>*</code>	Matches the previous element zero or more times.	<code>a.*c</code>	"abcabc" in "abcabc"
<code>+</code>	Matches the previous element one or more times.	<code>"be+"</code>	"bee" in "been", "be" in "bent"
<code>?</code>	Matches the previous element zero or one time.	<code>"rai?"</code>	"rai" in "rain"

QUANTIFIER	DESCRIPTION	PATTERN	MATCHES
{ n }	Matches the previous element exactly n times.	" , \d{3}"	" ,043" in "1,043.6" , " ,876" , " ,543" , and " ,210" in "9,876,543,210"
{ n , }	Matches the previous element at least n times.	"\d{2,}"	"166" , "29" , "1930"
{ n , m }	Matches the previous element at least n times, but no more than m times.	"\d{3,5}"	"166" , "17668" "19302" in "193024"
?	Matches the previous element zero or more times, but as few times as possible.	a.?c	"abc" in "abcbc"
+?	Matches the previous element one or more times, but as few times as possible.	"be+?"	"be" in "been" , "be" in "bent"
??	Matches the previous element zero or one time, but as few times as possible.	"rai??"	"ra" in "rain"
{ n }?	Matches the preceding element exactly n times.	" , \d{3}?"	" ,043" in "1,043.6" , " ,876" , " ,543" , and " ,210" in "9,876,543,210"
{ n , }?	Matches the previous element at least n times, but as few times as possible.	"\d{2,}?"	"166" , "29" , "1930"
{ n , m }?	Matches the previous element between n and m times, but as few times as possible.	"\d{3,5}?"	"166" , "17668" "193" , "024" in "193024"

Backreference Constructs

A backreference allows a previously matched subexpression to be identified subsequently in the same regular expression. The following table lists the backreference constructs supported by regular expressions in .NET. For more information, see [Backreference Constructs](#).

BACKREFERENCE CONSTRUCT	DESCRIPTION	PATTERN	MATCHES
\ $number$	Backreference. Matches the value of a numbered subexpression.	(\w)\1	"ee" in "seek"

BACKREFERENCE CONSTRUCT	DESCRIPTION	PATTERN	MATCHES
<code>\k< name ></code>	Named backreference. Matches the value of a named expression.	<code>(?<char>\w)\k<char></code>	"ee" in "seek"

Alternation Constructs

Alternation constructs modify a regular expression to enable either/or matching. These constructs include the language elements listed in the following table. For more information, see [Alternation Constructs](#).

ALTERNATION CONSTRUCT	DESCRIPTION	PATTERN	MATCHES
<code> </code>	Matches any one element separated by the vertical bar (<code> </code>) character.	<code>th(e is at)</code>	"the", "this" in "this is the day."
<code>(?(< expression >) yes no)</code> or <code>(?(< expression >) yes)</code>	Matches <i>yes</i> if the regular expression pattern designated by <i>expression</i> matches; otherwise, matches the optional <i>no</i> part. <i>expression</i> is interpreted as a zero-width assertion. To avoid ambiguity with a named or numbered capturing group, you can optionally use an explicit assertion, like this: <code>(?(< ?= expression >))</code> <i>yes no</i>	<code>(? (A)A\d{2}\b \b\d{3}\b)</code>	"A10", "910" in "A10 C103 910"
<code>(?(< name >) yes no)</code> or <code>(?(< name >) yes)</code>	Matches <i>yes</i> if <i>name</i> , a named or numbered capturing group, has a match; otherwise, matches the optional <i>no</i> .	<code>(?<quoted>)?(?<quoted>.+?" \s+\s)</code>	"Dogs.jpg ", "\Yiska playing.jpg\"" in "Dogs.jpg \"Yiska playing.jpg\""

Substitutions

Substitutions are regular expression language elements that are supported in replacement patterns. For more information, see [Substitutions](#). The metacharacters listed in the following table are atomic zero-width assertions.

CHARACTER	DESCRIPTION	PATTERN	REPLACEMENT PATTERN	INPUT STRING	RESULT STRING
<code>\$ number</code>	Substitutes the substring matched by group <i>number</i> .	<code>\b(\w+)(\s)(\w+)\b</code>	<code>\$3\$2\$1</code>	"one two"	"two one"

CHARACTER	DESCRIPTION	PATTERN	REPLACEMENT PATTERN	INPUT STRING	RESULT STRING
<code> \${ name }</code>	Substitutes the substring matched by the named group <i>name</i> .	<code>\b(?(<word1>\w+)(\s)(?<word2>\w+)\b</code>	<code> \${word2} \${word1}</code>	<code>"one two"</code>	<code>"two one"</code>
<code> \$\$</code>	Substitutes a literal "\$".	<code>\b(\d+)\s?USD</code>	<code> \$\$\\$1</code>	<code>"103 USD"</code>	<code>"\$103"</code>
<code> \$&</code>	Substitutes a copy of the whole match.	<code>\\$?\d*\.\?\d+</code>	<code>**\$&**</code>	<code>"\$1.30"</code>	<code>"**\$1.30**"</code>
<code> \$`</code>	Substitutes all the text of the input string before the match.	<code>B+</code>	<code>\$`</code>	<code>"AABBCC"</code>	<code>"AAACC"</code>
<code> \$'</code>	Substitutes all the text of the input string after the match.	<code>B+</code>	<code>\$'</code>	<code>"AABBCC"</code>	<code>"AACCCC"</code>
<code> \$+</code>	Substitutes the last group that was captured.	<code>B+(C+)</code>	<code>\$+</code>	<code>"AABBCCDD"</code>	<code>"AACDDD"</code>
<code> \$_</code>	Substitutes the entire input string.	<code>B+</code>	<code>\$_</code>	<code>"AABBCC"</code>	<code>"AAAABBBBBB"</code>

Regular Expression Options

You can specify options that control how the regular expression engine interprets a regular expression pattern. Many of these options can be specified either inline (in the regular expression pattern) or as one or more [RegexOptions](#) constants. This quick reference lists only inline options. For more information about inline and [RegexOptions](#) options, see the article [Regular Expression Options](#).

You can specify an inline option in two ways:

- By using the [miscellaneous construct](#) `(?imnsx-imnsx)`, where a minus sign (-) before an option or set of options turns those options off. For example, `(?i-mn)` turns case-insensitive matching (`i`) on, turns multiline mode (`m`) off, and turns unnamed group captures (`n`) off. The option applies to the regular expression pattern from the point at which the option is defined, and is effective either to the end of the pattern or to the point where another construct reverses the option.
- By using the [grouping construct](#) `(?imnsx-imnsx: subexpression)`, which defines options for the specified group only.

The .NET regular expression engine supports the following inline options:

OPTION	DESCRIPTION	PATTERN	MATCHES
i	Use case-insensitive matching.	\b(?i)a(?-i)a\w+\b	"aardvark" , "aaaAuto" in "aardvark AAAuto aaaAuto Adam breakfast"
m	Use multiline mode. <code>^</code> and <code>\$</code> match the beginning and end of a line, instead of the beginning and end of a string.	For an example, see the "Multiline Mode" section in Regular Expression Options .	
n	Do not capture unnamed groups.	For an example, see the "Explicit Captures Only" section in Regular Expression Options .	
s	Use single-line mode.	For an example, see the "Single-line Mode" section in Regular Expression Options .	
x	Ignore unescaped white space in the regular expression pattern.	\b(?x) \d+ \s \w+	"1 aardvark" , "2 cats" in "1 aardvark 2 cats IV centurions"

Miscellaneous Constructs

Miscellaneous constructs either modify a regular expression pattern or provide information about it. The following table lists the miscellaneous constructs supported by .NET. For more information, see [Miscellaneous Constructs](#).

CONSTRUCT	DEFINITION	EXAMPLE
(?imnsx-imnsx)	Sets or disables options such as case insensitivity in the middle of a pattern. For more information, see Regular Expression Options .	\bA(?i)b\w+\b matches "ABA" , "Able" in "ABA Able Act"
(?# comment)	Inline comment. The comment ends at the first closing parenthesis.	\bA(?#Matches words starting with A)\w+\b
# [to end of line]	X-mode comment. The comment starts at an unescaped <code>#</code> and continues to the end of the line.	(?x)\bA\w+\b#Matches words starting with A

See also

- [System.Text.RegularExpressions](#)
- [System.Text.RegularExpressions.Regex](#)
- [Regular Expressions](#)
- [Regular Expression Classes](#)
- [Regular Expressions - Quick Reference \(download in Word format\)](#)

- Regular Expressions - Quick Reference (download in PDF format)

Character Escapes in Regular Expressions

9/20/2022 • 4 minutes to read • [Edit Online](#)

The backslash (\) in a regular expression indicates one of the following:

- The character that follows it is a special character, as shown in the table in the following section. For example, `\b` is an anchor that indicates that a regular expression match should begin on a word boundary, `\t` represents a tab, and `\x020` represents a space.
- A character that otherwise would be interpreted as an unescaped language construct should be interpreted literally. For example, a brace (`{`) begins the definition of a quantifier, but a backslash followed by a brace (`\{`) indicates that the regular expression engine should match the brace. Similarly, a single backslash marks the beginning of an escaped language construct, but two backslashes (`\\"`) indicate that the regular expression engine should match the backslash.

NOTE

Character escapes are recognized in regular expression patterns but not in replacement patterns.

Character Escapes in .NET

The following table lists the character escapes supported by regular expressions in .NET.

CHARACTER OR SEQUENCE	DESCRIPTION
All characters except for the following: .\$^{}[({})*+?\\	Characters other than those listed in the Character or sequence column have no special meaning in regular expressions; they match themselves. The characters included in the Character or sequence column are special regular expression language elements. To match them in a regular expression, they must be escaped or included in a positive character group . For example, the regular expression <code>\\$\d+</code> or <code>[\$]\d+</code> matches "\$1200".
<code>\a</code>	Matches a bell (alarm) character, <code>\u0007</code> .
<code>\b</code>	In a <code>[character_group]</code> character class, matches a backspace, <code>\u0008</code> . (See Character Classes .) Outside a character class, <code>\b</code> is an anchor that matches a word boundary. (See Anchors .)
<code>\t</code>	Matches a tab, <code>\u0009</code> .
<code>\r</code>	Matches a carriage return, <code>\u000D</code> . Note that <code>\r</code> is not equivalent to the newline character, <code>\n</code> .
<code>\v</code>	Matches a vertical tab, <code>\u000B</code> .
<code>\f</code>	Matches a form feed, <code>\u000C</code> .

CHARACTER OR SEQUENCE	DESCRIPTION
<code>\n</code>	Matches a new line, <code>\u000A</code> .
<code>\e</code>	Matches an escape, <code>\u001B</code> .
<code>\ nnn</code>	Matches an ASCII character, where <i>nnn</i> consists of two or three digits that represent the octal character code. For example, <code>\040</code> represents a space character. This construct is interpreted as a backreference if it has only one digit (for example, <code>\2</code>) or if it corresponds to the number of a capturing group. (See Backreference Constructs .)
<code>\x nn</code>	Matches an ASCII character, where <i>nn</i> is a two-digit hexadecimal character code.
<code>\c X</code>	Matches an ASCII control character, where X is the letter of the control character. For example, <code>\cc</code> is CTRL-C.
<code>\u nnnn</code>	Matches a UTF-16 code unit whose value is <i>nnnn</i> hexadecimal. Note: The Perl 5 character escape that is used to specify Unicode is not supported by .NET. The Perl 5 character escape has the form <code>\x{ ##### ... }</code> , where ##### ... is a series of hexadecimal digits. Instead, use <code>\u nnnn</code> .
<code>\</code>	When followed by a character that is not recognized as an escaped character, matches that character. For example, <code>*</code> matches an asterisk (*) and is the same as <code>\x2A</code> .

An Example

The following example illustrates the use of character escapes in a regular expression. It parses a string that contains the names of the world's largest cities and their populations in 2009. Each city name is separated from its population by a tab (`\t`) or a vertical bar (`|` or `\u007C`). Individual cities and their populations are separated from each other by a carriage return and line feed.

```

using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string delimited = @"\G(.+)[\t\u007c](.+)\r?\n";
        string input = "Mumbai, India|13,922,125\t\n" +
                      "Shanghai, China\t13,831,900\n" +
                      "Karachi, Pakistan|12,991,000\n" +
                      "Delhi, India\t12,259,230\n" +
                      "Istanbul, Turkey|11,372,613\n";
        Console.WriteLine("Population of the World's Largest Cities, 2009");
        Console.WriteLine();
        Console.WriteLine("{0,-20} {1,10}", "City", "Population");
        Console.WriteLine();
        foreach (Match match in Regex.Matches(input, delimited))
            Console.WriteLine("{0,-20} {1,10}", match.Groups[1].Value,
                             match.Groups[2].Value);
    }
}

// The example displays the following output:
//      Population of the World's Largest Cities, 2009
//
//      City          Population
//
//      Mumbai, India      13,922,125
//      Shanghai, China     13,831,900
//      Karachi, Pakistan   12,991,000
//      Delhi, India        12,259,230
//      Istanbul, Turkey    11,372,613

```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim delimited As String = "\G(.+)[\t\u007c](.+)\r?\n"
        Dim input As String = "Mumbai, India|13,922,125" + vbCrLf + _
                             "Shanghai, China" + vbTab + "13,831,900" + vbCrLf + _
                             "Karachi, Pakistan|12,991,000" + vbCrLf + _
                             "Delhi, India" + vbTab + "12,259,230" + vbCrLf + _
                             "Istanbul, Turkey|11,372,613" + vbCrLf
        Console.WriteLine("Population of the World's Largest Cities, 2009")
        Console.WriteLine()
        Console.WriteLine("{0,-20} {1,10}", "City", "Population")
        Console.WriteLine()
        For Each match As Match In Regex.Matches(input, delimited)
            Console.WriteLine("{0,-20} {1,10}", match.Groups(1).Value, _
                             match.Groups(2).Value)
        Next
    End Sub
End Module
' The example displays the following output:
'      Population of the World's Largest Cities, 2009
'
'      City          Population
'
'      Mumbai, India      13,922,125
'      Shanghai, China     13,831,900
'      Karachi, Pakistan   12,991,000
'      Delhi, India        12,259,230
'      Istanbul, Turkey    11,372,613

```

The regular expression `\G(.+)[\t\u007c](.+)\r?\n` is interpreted as shown in the following table.

PATTERN	DESCRIPTION
\G	Begin the match where the last match ended.
(.+)	Match any character one or more times. This is the first capturing group.
[\t\u007c]	Match a tab (\t) or a vertical bar ().
(.+)	Match any character one or more times. This is the second capturing group.
\r?\n	Match zero or one occurrence of a carriage return followed by a new line.

See also

- [Regular Expression Language - Quick Reference](#)

Character classes in regular expressions

9/20/2022 • 33 minutes to read • [Edit Online](#)

A character class defines a set of characters, any one of which can occur in an input string for a match to succeed. The regular expression language in .NET supports the following character classes:

- Positive character groups. A character in the input string must match one of a specified set of characters. For more information, see [Positive Character Group](#).
- Negative character groups. A character in the input string must not match one of a specified set of characters. For more information, see [Negative Character Group](#).
- Any character. The `.` (dot or period) character in a regular expression is a wildcard character that matches any character except `\n`. For more information, see [Any Character](#).
- A general Unicode category or named block. A character in the input string must be a member of a particular Unicode category or must fall within a contiguous range of Unicode characters for a match to succeed. For more information, see [Unicode Category or Unicode Block](#).
- A negative general Unicode category or named block. A character in the input string must not be a member of a particular Unicode category or must not fall within a contiguous range of Unicode characters for a match to succeed. For more information, see [Negative Unicode Category or Unicode Block](#).
- A word character. A character in the input string can belong to any of the Unicode categories that are appropriate for characters in words. For more information, see [Word Character](#).
- A non-word character. A character in the input string can belong to any Unicode category that is not a word character. For more information, see [Non-Word Character](#).
- A white-space character. A character in the input string can be any Unicode separator character, as well as any one of a number of control characters. For more information, see [White-Space Character](#).
- A non-white-space character. A character in the input string can be any character that is not a white-space character. For more information, see [Non-White-Space Character](#).
- A decimal digit. A character in the input string can be any of a number of characters classified as Unicode decimal digits. For more information, see [Decimal Digit Character](#).
- A non-decimal digit. A character in the input string can be anything other than a Unicode decimal digit. For more information, see [Decimal Digit Character](#).

.NET supports character class subtraction expressions, which enables you to define a set of characters as the result of excluding one character class from another character class. For more information, see [Character Class Subtraction](#).

NOTE

Character classes that match characters by category, such as `\w` to match word characters or `\p{}` to match a Unicode category, rely on the [CharUnicodeInfo](#) class to provide information about character categories. In .NET Framework 4.6.2 and later versions, character categories are based on [The Unicode Standard, Version 8.0.0](#).

Positive character group: []

A positive character group specifies a list of characters, any one of which may appear in an input string for a match to occur. This list of characters may be specified individually, as a range, or both.

The syntax for specifying a list of individual characters is as follows:

```
[*character_group*]
```

where *character_group* is a list of the individual characters that can appear in the input string for a match to succeed. *character_group* can consist of any combination of one or more literal characters, [escape characters](#), or character classes.

The syntax for specifying a range of characters is as follows:

```
[firstCharacter-lastCharacter]
```

where *firstCharacter* is the character that begins the range and *lastCharacter* is the character that ends the range. A character range is a contiguous series of characters defined by specifying the first character in the series, a hyphen (-), and then the last character in the series. Two characters are contiguous if they have adjacent Unicode code points. *firstCharacter* must be the character with the lower code point, and *lastCharacter* must be the character with the higher code point.

NOTE

Because a positive character group can include both a set of characters and a character range, a hyphen character (-) is always interpreted as the range separator unless it is the first or last character of the group.

Some common regular expression patterns that contain positive character classes are listed in the following table.

PATTERN	DESCRIPTION
[aeiou]	Match all vowels.
[\p{P}\d]	Match all punctuation and decimal digit characters.
[\s\p{P}]	Match all white space and punctuation.

The following example defines a positive character group that contains the characters "a" and "e" so that the input string must contain the words "grey" or "gray" followed by another word for a match to occur.

```

using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string pattern = @"gr[ae]y\s\S+?[\s\p{P}]";
        string input = "The gray wolf jumped over the grey wall.";
        MatchCollection matches = Regex.Matches(input, pattern);
        foreach (Match match in matches)
            Console.WriteLine($"'{match.Value}'");
    }
}
// The example displays the following output:
//      'gray wolf '
//      'grey wall.'

```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim pattern As String = "gr[ae]y\s\S+?[\s\p{P}]"
        Dim input As String = "The gray wolf jumped over the grey wall."
        Dim matches As MatchCollection = Regex.Matches(input, pattern)
        For Each match As Match In matches
            Console.WriteLine($"'{match.Value}'")
        Next
    End Sub
End Module
' The example displays the following output:
'      'gray wolf '
'      'grey wall.'

```

The regular expression `gr[ae]y\s\S+?[\s\p{P}]` is defined as follows:

PATTERN	DESCRIPTION
<code>gr</code>	Match the literal characters "gr".
<code>[ae]</code>	Match either an "a" or an "e".
<code>y\s</code>	Match the literal character "y" followed by a white-space character.
<code>\S+?</code>	Match one or more non-white-space characters, but as few as possible.
<code>[\s\p{P}]</code>	Match either a white-space character or a punctuation mark.

The following example matches words that begin with any capital letter. It uses the subexpression `[A-Z]` to represent the range of capital letters from A to Z.

```

using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string pattern = @"\b[A-Z]\w*\b";
        string input = "A city Albany Zulu maritime Marseilles";
        foreach (Match match in Regex.Matches(input, pattern))
            Console.WriteLine(match.Value);
    }
}
// The example displays the following output:
//      A
//      Albany
//      Zulu
//      Marseilles

```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim pattern As String = "\b[A-Z]\w*\b"
        Dim input As String = "A city Albany Zulu maritime Marseilles"
        For Each match As Match In Regex.Matches(input, pattern)
            Console.WriteLine(match.Value)
        Next
    End Sub
End Module

```

The regular expression `\b[A-Z]\w*\b` is defined as shown in the following table.

PATTERN	DESCRIPTION
<code>\b</code>	Start at a word boundary.
<code>[A-Z]</code>	Match any uppercase character from A to Z.
<code>\w*</code>	Match zero or more word characters.
<code>\b</code>	Match a word boundary.

Negative character group: [^]

A negative character group specifies a list of characters that must not appear in an input string for a match to occur. The list of characters may be specified individually, as a range, or both.

The syntax for specifying a list of individual characters is as follows:

`[*^character_group*]`

where *character_group* is a list of the individual characters that cannot appear in the input string for a match to succeed. *character_group* can consist of any combination of one or more literal characters, [escape characters](#), or character classes.

The syntax for specifying a range of characters is as follows:

[^*firstCharacter*-*lastCharacter*]

where *firstCharacter* is the character that begins the range and *lastCharacter* is the character that ends the range. A character range is a contiguous series of characters defined by specifying the first character in the series, a hyphen (-), and then the last character in the series. Two characters are contiguous if they have adjacent Unicode code points. *firstCharacter* must be the character with the lower code point, and *lastCharacter* must be the character with the higher code point.

NOTE

Because a negative character group can include both a set of characters and a character range, a hyphen character (-) is always interpreted as the range separator unless it is the first or last character of the group.

Two or more character ranges can be concatenated. For example, to specify the range of decimal digits from "0" through "9", the range of lowercase letters from "a" through "f", and the range of uppercase letters from "A" through "F", use [0-9a-fA-F].

The leading caret character (^) in a negative character group is mandatory and indicates the character group is a negative character group instead of a positive character group.

IMPORTANT

A negative character group in a larger regular expression pattern is not a zero-width assertion. That is, after evaluating the negative character group, the regular expression engine advances one character in the input string.

Some common regular expression patterns that contain negative character groups are listed in the following table.

PATTERN	DESCRIPTION
[^aeiou]	Match all characters except vowels.
[^\p{P}\d]	Match all characters except punctuation and decimal digit characters.

The following example matches any word that begins with the characters "th" and is not followed by an "o".

```
using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string pattern = @"\bth[^o]\w+\b";
        string input = "thought thing though them through thus thorough this";
        foreach (Match match in Regex.Matches(input, pattern))
            Console.WriteLine(match.Value);
    }
}
// The example displays the following output:
//      thing
//      them
//      through
//      thus
//      this
```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim pattern As String = "\bth[^o]\w+\b"
        Dim input As String = "thought thing though them through thus " + _
                             "thorough this"
        For Each match As Match In Regex.Matches(input, pattern)
            Console.WriteLine(match.Value)
        Next
    End Sub
End Module
' The example displays the following output:
'     thing
'     them
'     through
'     thus
'     this

```

The regular expression `\bth[^o]\w+\b` is defined as shown in the following table.

PATTERN	DESCRIPTION
<code>\b</code>	Start at a word boundary.
<code>th</code>	Match the literal characters "th".
<code>[^o]</code>	Match any character that is not an "o".
<code>\w+</code>	Match one or more word characters.
<code>\b</code>	End at a word boundary.

Any character: .

The period character (.) matches any character except `\n` (the newline character, \u000A), with the following two qualifications:

- If a regular expression pattern is modified by the [RegexOptions.Singleline](#) option, or if the portion of the pattern that contains the `.` character class is modified by the `s` option, `.` matches any character. For more information, see [Regular Expression Options](#).

The following example illustrates the different behavior of the `.` character class by default and with the [RegexOptions.Singleline](#) option. The regular expression `^.+` starts at the beginning of the string and matches every character. By default, the match ends at the end of the first line; the regular expression pattern matches the carriage return character, `\r` or \u000D, but it does not match `\n`. Because the [RegexOptions.Singleline](#) option interprets the entire input string as a single line, it matches every character in the input string, including `\n`.

```

using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string pattern = "^.+";
        string input = "This is one line and" + Environment.NewLine + "this is the second.";
        foreach (Match match in Regex.Matches(input, pattern))
            Console.WriteLine(Regex.Escape(match.Value));

        Console.WriteLine();
        foreach (Match match in Regex.Matches(input, pattern, RegexOptions.Singleline))
            Console.WriteLine(Regex.Escape(match.Value));
    }
}

// The example displays the following output:
//      This\ is\ one\ line\ and\r
//
//      This\ is\ one\ line\ and\r\nthis\ is\ the\ second\.

```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim pattern As String = "^.+"
        Dim input As String = "This is one line and" + vbCrLf + "this is the second."
        For Each match As Match In Regex.Matches(input, pattern)
            Console.WriteLine(Regex.Escape(match.Value))
        Next
        Console.WriteLine()
        For Each match As Match In Regex.Matches(input, pattern, RegexOptions.SingleLine)
            Console.WriteLine(Regex.Escape(match.Value))
        Next
    End Sub
End Module
' The example displays the following output:
'      This\ is\ one\ line\ and\r
'
'      This\ is\ one\ line\ and\r\nthis\ is\ the\ second\.

```

NOTE

Because it matches any character except `\n`, the `[.]` character class also matches `\r` (the carriage return character, \u000D).

- In a positive or negative character group, a period is treated as a literal period character, and not as a character class. For more information, see [Positive Character Group](#) and [Negative Character Group](#) earlier in this topic. The following example provides an illustration by defining a regular expression that includes the period character (`.`) both as a character class and as a member of a positive character group. The regular expression `\b.*[.?!;:](\s|\z)` begins at a word boundary, matches any character until it encounters one of five punctuation marks, including a period, and then matches either a white-space character or the end of the string.

```

using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string pattern = @"\b.*[.?!;:](\s|\z)";
        string input = "this. what: is? go, thing.";
        foreach (Match match in Regex.Matches(input, pattern))
            Console.WriteLine(match.Value);
    }
}
// The example displays the following output:
//      this. what: is? go, thing.

```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim pattern As String = "\b.*[.?!;:](\s|\z)"
        Dim input As String = "this. what: is? go, thing."
        For Each match As Match In Regex.Matches(input, pattern)
            Console.WriteLine(match.Value)
        Next
    End Sub
End Module
' The example displays the following output:
'      this. what: is? go, thing.

```

NOTE

Because it matches any character, the `.` language element is often used with a lazy quantifier if a regular expression pattern attempts to match any character multiple times. For more information, see [Quantifiers](#).

Unicode category or Unicode block: \p{}

The Unicode standard assigns each character a general category. For example, a particular character can be an uppercase letter (represented by the `Lu` category), a decimal digit (the `Nd` category), a math symbol (the `Sm` category), or a paragraph separator (the `Zl` category). Specific character sets in the Unicode standard also occupy a specific range or block of consecutive code points. For example, the basic Latin character set is found from `\u0000` through `\u007F`, while the Arabic character set is found from `\u0600` through `\u06FF`.

The regular expression construct

`\p{ name }`

matches any character that belongs to a Unicode general category or named block, where `name` is the category abbreviation or named block name. For a list of category abbreviations, see the [Supported Unicode General Categories](#) section later in this topic. For a list of named blocks, see the [Supported Named Blocks](#) section later in this topic.

TIP

Matching may be improved if the string is first normalized by calling the `String.Normalize` method.

The following example uses the `\p{ name }` construct to match both a Unicode general category (in this case, the `Pd`, or Punctuation, Dash category) and a named block (the `IsGreek` and `IsBasicLatin` named blocks).

```
using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string pattern = @"\b(\p{IsGreek}+(\s)?)+\p{Pd}\s(\p{IsBasicLatin}+(\s)?)+";
        string input = "Kata Maθθaiov - The Gospel of Matthew";

        Console.WriteLine(Regex.IsMatch(input, pattern));           // Displays True.
    }
}
```

```
Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim pattern As String = "\b(\p{IsGreek}+(\s)?)+\p{Pd}\s(\p{IsBasicLatin}+(\s)?)+"
        Dim input As String = "Kata Maθθaiov - The Gospel of Matthew"

        Console.WriteLine(Regex.IsMatch(input, pattern))           ' Displays True.
    End Sub
End Module
```

The regular expression `\b(\p{IsGreek}+(\s)?)+\p{Pd}\s(\p{IsBasicLatin}+(\s)?)+` is defined as shown in the following table.

PATTERN	DESCRIPTION
<code>\b</code>	Start at a word boundary.
<code>\p{IsGreek}+</code>	Match one or more Greek characters.
<code>(\s)?</code>	Match zero or one white-space character.
<code>(\p{IsGreek}+(\s)?)+</code>	Match the pattern of one or more Greek characters followed by zero or one white-space characters one or more times.
<code>\p{Pd}</code>	Match a Punctuation, Dash character.
<code>\s</code>	Match a white-space character.
<code>\p{IsBasicLatin}+</code>	Match one or more basic Latin characters.
<code>(\s)?</code>	Match zero or one white-space character.
<code>(\p{IsBasicLatin}+(\s)?)+</code>	Match the pattern of one or more basic Latin characters followed by zero or one white-space characters one or more times.

Negative Unicode category or Unicode block: \P{}

The Unicode standard assigns each character a general category. For example, a particular character can be an uppercase letter (represented by the `Lu` category), a decimal digit (the `Nd` category), a math symbol (the `Sm` category), or a paragraph separator (the `Zl` category). Specific character sets in the Unicode standard also occupy a specific range or block of consecutive code points. For example, the basic Latin character set is found from `\u0000` through `\u007F`, while the Arabic character set is found from `\u0600` through `\u06FF`.

The regular expression construct

```
\P{ name }
```

matches any character that does not belong to a Unicode general category or named block, where *name* is the category abbreviation or named block name. For a list of category abbreviations, see the [Supported Unicode General Categories](#) section later in this topic. For a list of named blocks, see the [Supported Named Blocks](#) section later in this topic.

TIP

Matching may be improved if the string is first normalized by calling the [String.Normalize](#) method.

The following example uses the `\P{ name }` construct to remove any currency symbols (in this case, the `Sc`, or Symbol, Currency category) from numeric strings.

```
using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string pattern = @"(\P{Sc})+";

        string[] values = { "$164,091.78", "£1,073,142.68", "73¢", "€120" };
        foreach (string value in values)
            Console.WriteLine(Regex.Match(value, pattern).Value);
    }
}
// The example displays the following output:
//      164,091.78
//      1,073,142.68
//      73
//      120
```

```
Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim pattern As String = "(\P{Sc})+"

        Dim values() As String = {"$164,091.78", "£1,073,142.68", "73¢", "€120"}
        For Each value As String In values
            Console.WriteLine(Regex.Match(value, pattern).Value)
        Next
    End Sub
End Module
' The example displays the following output:
'      164,091.78
'      1,073,142.68
'      73
'      120
```

The regular expression pattern `(\P{Sc})+` matches one or more characters that are not currency symbols; it effectively strips any currency symbol from the result string.

Word character: \w

`\w` matches any word character. A word character is a member of any of the Unicode categories listed in the following table.

CATEGORY	DESCRIPTION
Li	Letter, Lowercase
Lu	Letter, Uppercase
Lt	Letter, Titlecase
Lo	Letter, Other
Lm	Letter, Modifier
Mn	Mark, Nonspacing
Nd	Number, Decimal Digit
Pc	Punctuation, Connector. This category includes ten characters, the most commonly used of which is the LOWLINE character (_), u+005F.

If ECMAScript-compliant behavior is specified, `\w` is equivalent to `[a-zA-Z_0-9]`. For information on ECMAScript regular expressions, see the "ECMAScript Matching Behavior" section in [Regular Expression Options](#).

NOTE

Because it matches any word character, the `\w` language element is often used with a lazy quantifier if a regular expression pattern attempts to match any word character multiple times, followed by a specific word character. For more information, see [Quantifiers](#).

The following example uses the `\w` language element to match duplicate characters in a word. The example defines a regular expression pattern, `(\w)\1`, which can be interpreted as follows.

ELEMENT	DESCRIPTION
<code>(\w)</code>	Match a word character. This is the first capturing group.
<code>\1</code>	Match the value of the first capture.

```

using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string pattern = @"(\w)\1";
        string[] words = { "trellis", "seer", "latter", "summer",
                           "hoarse", "lesser", "aardvark", "stunned" };
        foreach (string word in words)
        {
            Match match = Regex.Match(word, pattern);
            if (match.Success)
                Console.WriteLine("{0} found in '{1}' at position {2}.",
                                 match.Value, word, match.Index);
            else
                Console.WriteLine("No double characters in '{0}'.", word);
        }
    }
}

// The example displays the following output:
//      'll' found in 'trellis' at position 3.
//      'ee' found in 'seer' at position 1.
//      'tt' found in 'latter' at position 2.
//      'mm' found in 'summer' at position 2.
//      No double characters in 'hoarse'.
//      'ss' found in 'lesser' at position 2.
//      'aa' found in 'aardvark' at position 0.
//      'nn' found in 'stunned' at position 3.

```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim pattern As String = "(\w)\1"
        Dim words() As String = {"trellis", "seer", "latter", "summer", _
                               "hoarse", "lesser", "aardvark", "stunned"}
        For Each word As String In words
            Dim match As Match = Regex.Match(word, pattern)
            If match.Success Then
                Console.WriteLine("{0} found in '{1}' at position {2}.",
                                 match.Value, word, match.Index)
            Else
                Console.WriteLine("No double characters in '{0}'.", word)
            End If
        Next
    End Sub
End Module

' The example displays the following output:
'      'll' found in 'trellis' at position 3.
'      'ee' found in 'seer' at position 1.
'      'tt' found in 'latter' at position 2.
'      'mm' found in 'summer' at position 2.
'      No double characters in 'hoarse'.
'      'ss' found in 'lesser' at position 2.
'      'aa' found in 'aardvark' at position 0.
'      'nn' found in 'stunned' at position 3.

```

Non-word character: \W

\W matches any non-word character. The \W language element is equivalent to the following character class:

```
[^\p{Ll}\p{Lu}\p{Lt}\p{Lo}\p{Nd}\p{Pc}\p{Lm}]
```

In other words, it matches any character except for those in the Unicode categories listed in the following table.

CATEGORY	DESCRIPTION
Li	Letter, Lowercase
Lu	Letter, Uppercase
Lt	Letter, Titlecase
Lo	Letter, Other
Lm	Letter, Modifier
Mn	Mark, Nonspacing
Nd	Number, Decimal Digit
Pc	Punctuation, Connector. This category includes ten characters, the most commonly used of which is the LOWLINE character (_), u+005F.

If ECMAScript-compliant behavior is specified, `\w` is equivalent to `[^a-zA-Z_0-9]`. For information on ECMAScript regular expressions, see the "ECMAScript Matching Behavior" section in [Regular Expression Options](#).

NOTE

Because it matches any non-word character, the `\w` language element is often used with a lazy quantifier if a regular expression pattern attempts to match any non-word character multiple times followed by a specific non-word character. For more information, see [Quantifiers](#).

The following example illustrates the `\w` character class. It defines a regular expression pattern, `\b(\w+)(\w){1,2}`, that matches a word followed by one or two non-word characters, such as white space or punctuation. The regular expression is interpreted as shown in the following table.

ELEMENT	DESCRIPTION
\b	Begin the match at a word boundary.
(\w+)	Match one or more word characters. This is the first capturing group.
(\w){1,2}	Match a non-word character either one or two times. This is the second capturing group.

```
using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string pattern = @"\b(\w+)(\W){1,2}";
        string input = "The old, grey mare slowly walked across the narrow, green pasture.";
        foreach (Match match in Regex.Matches(input, pattern))
        {
            Console.WriteLine(match.Value);
            Console.Write("  Non-word character(s):");
            CaptureCollection captures = match.Groups[2].Captures;
            for (int ctr = 0; ctr < captures.Count; ctr++)
                Console.WriteLine(@"'{0}' (\u{1}){2}", captures[ctr].Value,
                    Convert.ToInt16(captures[ctr].Value[0]).ToString("X4"),
                    ctr < captures.Count - 1 ? ", " : "");
            Console.WriteLine();
        }
    }
}

// The example displays the following output:
//     The
//         Non-word character(s):' ' (\u0020)
//     old,
//         Non-word character(s):',' (\u002C), ' ' (\u0020)
//     grey
//         Non-word character(s):' ' (\u0020)
//     mare
//         Non-word character(s):' ' (\u0020)
//     slowly
//         Non-word character(s):' ' (\u0020)
//     Non-word character(s):' ' (\u0020)
//     walked
//         Non-word character(s):' ' (\u0020)
//     Non-word character(s):' ' (\u0020)
//     Non-word character(s):' ' (\u0020)
//     across
//         Non-word character(s):' ' (\u0020)
//     the
//         Non-word character(s):' ' (\u0020)
//     narrow,
//         Non-word character(s):',' (\u002C), ' ' (\u0020)
//     green
//         Non-word character(s):' ' (\u0020)
//     pasture.
//         Non-word character(s):'.' (\u002E)
```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim pattern As String = "\b(\w+)(\W){1,2}"
        Dim input As String = "The old, grey mare slowly walked across the narrow, green pasture."
        For Each match As Match In Regex.Matches(input, pattern)
            Console.WriteLine(match.Value)
            Console.Write(" Non-word character(s):")
            Dim captures As CaptureCollection = match.Groups(2).Captures
            For ctr As Integer = 0 To captures.Count - 1
                Console.Write("{0}' (\u{1}){2}", captures(ctr).Value, _
                    Convert.ToInt16(captures(ctr).Value.Chars(0)).ToString("X4"), _
                    If(ctr < captures.Count - 1, ", ", ""))
            Next
            Console.WriteLine()
        Next
    End Sub
End Module
' The example displays the following output:
'     The
'         Non-word character(s):' ' (\u0020)
'     old,
'         Non-word character(s):',' (\u002C), ' ' (\u0020)
'     grey
'         Non-word character(s):' ' (\u0020)
'     mare
'         Non-word character(s):' ' (\u0020)
'     slowly
'         Non-word character(s):' ' (\u0020)
'     walked
'         Non-word character(s):' ' (\u0020)
'     across
'         Non-word character(s):' ' (\u0020)
'     the
'         Non-word character(s):' ' (\u0020)
'     narrow,
'         Non-word character(s):',' (\u002C), ' ' (\u0020)
'     green
'         Non-word character(s):' ' (\u0020)
'     pasture.
'         Non-word character(s):'.' (\u002E)

```

Because the [Group](#) object for the second capturing group contains only a single captured non-word character, the example retrieves all captured non-word characters from the [CaptureCollection](#) object that is returned by the [Group.Captures](#) property.

Whitespace character: \s

`\s` matches any whitespace character. It is equivalent to the escape sequences and Unicode categories listed in the following table.

CATEGORY	DESCRIPTION
<code>\f</code>	The form feed character, \u000C.
<code>\n</code>	The newline character, \u000A.
<code>\r</code>	The carriage return character, \u000D.

CATEGORY	DESCRIPTION
\t	The tab character, \u0009.
\v	The vertical tab character, \u000B.
\x85	The NEXT LINE (NEL) character, \u0085.
\p{Z}	Matches all separator characters . This includes the <code>zs</code> , <code>z1</code> , and <code>zp</code> categories.

If ECMAScript-compliant behavior is specified, `\s` is equivalent to `[\f\n\r\t\v]`. For information on ECMAScript regular expressions, see the "ECMAScript Matching Behavior" section in [Regular Expression Options](#).

The following example illustrates the `\s` character class. It defines a regular expression pattern, `\b\w+(e)?s(\s|$)`, that matches a word ending in either "s" or "es" followed by either a white-space character or the end of the input string. The regular expression is interpreted as shown in the following table.

ELEMENT	DESCRIPTION
\b	Begin the match at a word boundary.
\w+	Match one or more word characters.
(e)?	Match an "e" either zero or one time.
s	Match an "s".
(\s \$)	Match either a white-space character or the end of the input string.

```
using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string pattern = @"\b\w+(e)?s(\s|$)";
        string input = "matches stores stops leave leaves";
        foreach (Match match in Regex.Matches(input, pattern))
            Console.WriteLine(match.Value);
    }
}
// The example displays the following output:
//      matches
//      stores
//      stops
//      leaves
```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim pattern As String = "\b\w+(e)?s(\s|$)"
        Dim input As String = "matches stores stops leave leaves"
        For Each match As Match In Regex.Matches(input, pattern)
            Console.WriteLine(match.Value)
        Next
    End Sub
End Module
' The example displays the following output:
'     matches
'     stores
'     stops
'     leaves

```

Non-whitespace character: \S

\s matches any non-white-space character. It is equivalent to the [^\f\n\r\t\v\x85\p{Z}] regular expression pattern, or the opposite of the regular expression pattern that is equivalent to \s, which matches white-space characters. For more information, see [White-Space Character:\s](#).

If ECMAScript-compliant behavior is specified, \s is equivalent to [^\f\n\r\t\v]. For information on ECMAScript regular expressions, see the "ECMAScript Matching Behavior" section in [Regular Expression Options](#).

The following example illustrates the \s language element. The regular expression pattern \b(\s+)\s? matches strings that are delimited by white-space characters. The second element in the match's [GroupCollection](#) object contains the matched string. The regular expression can be interpreted as shown in the following table.

ELEMENT	DESCRIPTION
\b	Begin the match at a word boundary.
(\s+)	Match one or more non-white-space characters. This is the first capturing group.
\s?	Match zero or one white-space character.

```
using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string pattern = @"\b(\S+)\s?";
        string input = "This is the first sentence of the first paragraph. " +
                      "This is the second sentence.\n" +
                      "This is the only sentence of the second paragraph.";
        foreach (Match match in Regex.Matches(input, pattern))
            Console.WriteLine(match.Groups[1]);
    }
}

// The example displays the following output:
// This
// is
// the
// first
// sentence
// of
// the
// first
// paragraph.
// This
// is
// the
// second
// sentence.
// This
// is
// the
// only
// sentence
// of
// the
// second
// paragraph.
```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim pattern As String = "\b(\S+)\s?"
        Dim input As String = "This is the first sentence of the first paragraph. " + _
                            "This is the second sentence." + vbCrLf + _
                            "This is the only sentence of the second paragraph."
        For Each match As Match In Regex.Matches(input, pattern)
            Console.WriteLine(match.Groups(1))
        Next
    End Sub
End Module
' The example displays the following output:
' This
' is
' the
' first
' sentence
' of
' the
' first
' paragraph.
' This
' is
' the
' second
' sentence.
' This
' is
' the
' only
' sentence
' of
' the
' second
' paragraph.

```

Decimal digit character: \d

`\d` matches any decimal digit. It is equivalent to the `\p{Nd}` regular expression pattern, which includes the standard decimal digits 0-9 as well as the decimal digits of a number of other character sets.

If ECMAScript-compliant behavior is specified, `\d` is equivalent to `[0-9]`. For information on ECMAScript regular expressions, see the "ECMAScript Matching Behavior" section in [Regular Expression Options](#).

The following example illustrates the `\d` language element. It tests whether an input string represents a valid telephone number in the United States and Canada. The regular expression pattern

`^(\\(?\\d{3}\\)?)?[-\\s-]?\\d{3}-\\d{4}$` is defined as shown in the following table.

ELEMENT	DESCRIPTION
<code>^</code>	Begin the match at the beginning of the input string.
<code>\\(?</code>	Match zero or one literal "(" character.
<code>\\d{3}</code>	Match three decimal digits.
<code>\\)?</code>	Match zero or one literal ")" character.

ELEMENT	DESCRIPTION
[\s-]	Match a hyphen or a white-space character.
(\(? \d{3} \)? [\s-])?	Match an optional opening parenthesis followed by three decimal digits, an optional closing parenthesis, and either a white-space character or a hyphen zero or one time. This is the first capturing group.
\d{3}-\d{4}	Match three decimal digits followed by a hyphen and four more decimal digits.
\$	Match the end of the input string.

```

using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string pattern = @"^(\(? \d{3} \)? [\s-] )?\d{3}-\d{4}$";
        string[] inputs = { "111 111-1111", "222-2222", "222 333-444",
                           "(212) 111-1111", "111-AB1-1111",
                           "212-111-1111", "01 999-9999" };

        foreach (string input in inputs)
        {
            if (Regex.IsMatch(input, pattern))
                Console.WriteLine(input + ": matched");
            else
                Console.WriteLine(input + ": match failed");
        }
    }
}

// The example displays the following output:
//      111 111-1111: matched
//      222-2222: matched
//      222 333-444: match failed
//      (212) 111-1111: matched
//      111-AB1-1111: match failed
//      212-111-1111: matched
//      01 999-9999: match failed

```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim pattern As String = "^(\\(?\\d{3}\\)?)?\\s-]\\)?\\d{3}-\\d{4}$"
        Dim inputs() As String = {"111 111-1111", "222-2222", "222 333-444", _
                                "(212) 111-1111", "111-AB1-1111", _
                                "212-111-1111", "01 999-9999"}
        
        For Each input As String In inputs
            If Regex.IsMatch(input, pattern) Then
                Console.WriteLine(input + ": matched")
            Else
                Console.WriteLine(input + ": match failed")
            End If
        Next
    End Sub
End Module
' The example displays the following output:
'   111 111-1111: matched
'   222-2222: matched
'   222 333-444: match failed
'   (212) 111-1111: matched
'   111-AB1-1111: match failed
'   212-111-1111: matched
'   01 999-9999: match failed

```

Non-digit character: \D

\D matches any non-digit character. It is equivalent to the \P{Nd} regular expression pattern.

If ECMAScript-compliant behavior is specified, \D is equivalent to [^0-9]. For information on ECMAScript regular expressions, see the "ECMAScript Matching Behavior" section in [Regular Expression Options](#).

The following example illustrates the \D language element. It tests whether a string such as a part number consists of the appropriate combination of decimal and non-decimal characters. The regular expression pattern ^\D\d{1,5}\D*\$ is defined as shown in the following table.

ELEMENT	DESCRIPTION
^	Begin the match at the beginning of the input string.
\D	Match a non-digit character.
\d{1,5}	Match from one to five decimal digits.
\D*	Match zero, one, or more non-decimal characters.
\$	Match the end of the input string.

```

using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string pattern = @"\D\d{1,5}\D*";
        string[] inputs = { "A1039C", "AA0001", "C18A", "Y938518" };

        foreach (string input in inputs)
        {
            if (Regex.IsMatch(input, pattern))
                Console.WriteLine(input + ": matched");
            else
                Console.WriteLine(input + ": match failed");
        }
    }
}

// The example displays the following output:
//      A1039C: matched
//      AA0001: match failed
//      C18A: matched
//      Y938518: match failed

```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim pattern As String = "\D\d{1,5}\D*"
        Dim inputs() As String = {"A1039C", "AA0001", "C18A", "Y938518"}

        For Each input As String In inputs
            If Regex.IsMatch(input, pattern) Then
                Console.WriteLine(input + ": matched")
            Else
                Console.WriteLine(input + ": match failed")
            End If
        Next
    End Sub
End Module
' The example displays the following output:

```

Supported Unicode general categories

Unicode defines the general categories listed in the following table. For more information, see the "UCD File Format" and "General Category Values" subtopics at the [Unicode Character Database](#), Sec. 5.7.1, Table 12.

CATEGORY	DESCRIPTION
Lu	Letter, Uppercase
Ll	Letter, Lowercase
Lt	Letter, Titlecase
Lm	Letter, Modifier

CATEGORY	DESCRIPTION
Lo	Letter, Other
L	All letter characters. This includes the Lu, Ll, Lt, Lm, and Lo characters.
Mn	Mark, Nonspacing
Mc	Mark, Spacing Combining
Me	Mark, Enclosing
M	All combining marks. This includes the Mn, Mc, and Me categories.
Nd	Number, Decimal Digit
Nl	Number, Letter
No	Number, Other
N	All numbers. This includes the Nd, Nl, and No categories.
Pc	Punctuation, Connector
Pd	Punctuation, Dash
Ps	Punctuation, Open
Pe	Punctuation, Close
Pi	Punctuation, Initial quote (may behave like Ps or Pe depending on usage)
Pf	Punctuation, Final quote (may behave like Ps or Pe depending on usage)
Po	Punctuation, Other
P	All punctuation characters. This includes the Pc, Pd, Ps, Pe, Pi, Pf, and Po categories.
Sm	Symbol, Math
Sc	Symbol, Currency
Sk	Symbol, Modifier
So	Symbol, Other

CATEGORY	DESCRIPTION
S	All symbols. This includes the Sm, Sc, Sk, and So categories.
Zs	Separator, Space
Zl	Separator, Line
Zp	Separator, Paragraph
Z	All separator characters. This includes the Zs, Zl, and Zp categories.
Cc	Other, Control
Cf	Other, Format
Cs	Other, Surrogate
Co	Other, Private Use
Cn	Other, Not Assigned or Noncharacter
C	All other characters. This includes the Cc, Cf, Cs, Co, and Cn categories.

You can determine the Unicode category of any particular character by passing that character to the [GetUnicodeCategory](#) method. The following example uses the [GetUnicodeCategory](#) method to determine the category of each element in an array that contains selected Latin characters.

```
using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        char[] chars = { 'a', 'X', '8', ',', ' ', '\u0009', '!' };

        foreach (char ch in chars)
            Console.WriteLine("{0}: {1}", Regex.Escape(ch.ToString()),
                Char.GetUnicodeCategory(ch));
    }
}
// The example displays the following output:
//      'a': LowercaseLetter
//      'X': UppercaseLetter
//      '8': DecimalDigitNumber
//      ',': OtherPunctuation
//      '\u0009': SpaceSeparator
//      '\t': Control
//      '!': OtherPunctuation
```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim chars() As Char = {"a"c, "X"c, "8"c, ","c, " "c, ChrW(9), !"c}

        For Each ch As Char In chars
            Console.WriteLine("'{0}': {1}", Regex.Escape(ch.ToString()), _
                Char.GetUnicodeCategory(ch))
        Next
    End Sub
End Module
' The example displays the following output:
'      'a': LowercaseLetter
'      'X': UppercaseLetter
'      '8': DecimalDigitNumber
'      ','': OtherPunctuation
'      '\ ''': SpaceSeparator
'      '\t': Control
'      '!': OtherPunctuation

```

Supported named blocks

.NET provides the named blocks listed in the following table. The set of supported named blocks is based on Unicode 4.0 and Perl 5.6. For a regular expression that uses named blocks, see the [Unicode category or Unicode block:`\p{}`](#) section.

CODE POINT RANGE	BLOCK NAME
0000 - 007F	IsBasicLatin
0080 - 0OFF	IsLatin-1Supplement
0100 - 017F	IsLatinExtended-A
0180 - 024F	IsLatinExtended-B
0250 - 02AF	IsIPAExtensions
02B0 - 02FF	IsSpacingModifierLetters
0300 - 036F	IsCombiningDiacriticalMarks
0370 - 03FF	IsGreek -or- IsGreekandCoptic
0400 - 04FF	IsCyrilllic
0500 - 052F	IsCyrilllicSupplement
0530 - 058F	IsArmenian

CODE POINT RANGE	BLOCK NAME
0590 - 05FF	IsHebrew
0600 - 06FF	IsArabic
0700 - 074F	IsSyriac
0780 - 07BF	IsThaana
0900 - 097F	IsDevanagari
0980 - 09FF	IsBengali
0A00 - 0A7F	IsGurmukhi
0A80 - 0AFF	IsGujarati
0B00 - 0B7F	IsOriya
0B80 - 0BFF	IsTamil
0C00 - 0C7F	IsTelugu
0C80 - 0CFF	IsKannada
0D00 - 0D7F	IsMalayalam
0D80 - 0DFF	IsSinhala
0E00 - 0E7F	IsThai
0E80 - 0EFF	IsLao
0F00 - 0FFF	IsTibetan
1000 - 109F	IsMyanmar
10A0 - 10FF	IsGeorgian
1100 - 11FF	IsHangulJamo
1200 - 137F	IsEthiopic
13A0 - 13FF	IsCherokee
1400 - 167F	IsUnifiedCanadianAboriginalSyllabics
1680 - 169F	IsOgham

CODE POINT RANGE	BLOCK NAME
16A0 - 16FF	IsRunic
1700 - 171F	IsTagalog
1720 - 173F	IsHanunoo
1740 - 175F	IsBuhid
1760 - 177F	IsTagbanwa
1780 - 17FF	IsKhmer
1800 - 18AF	IsMongolian
1900 - 194F	IsLimbu
1950 - 197F	IsTaiLe
19E0 - 19FF	IsKhmerSymbols
1D00 - 1D7F	IsPhoneticExtensions
1E00 - 1EFF	IsLatinExtendedAdditional
1F00 - 1FFF	IsGreekExtended
2000 - 206F	IsGeneralPunctuation
2070 - 209F	IsSuperscriptsandSubscripts
20A0 - 20CF	IsCurrencySymbols
20D0 - 20FF	IsCombiningDiacriticalMarksforSymbols
-Or-	
	IsCombiningMarksforSymbols
2100 - 214F	IsLetterlikeSymbols
2150 - 218F	IsNumberForms
2190 - 21FF	IsArrows
2200 - 22FF	IsMathematicalOperators
2300 - 23FF	IsMiscellaneousTechnical
2400 - 243F	IsControlPictures

CODE POINT RANGE	BLOCK NAME
2440 - 245F	IsOpticalCharacterRecognition
2460 - 24FF	IsEnclosedAlphanumerics
2500 - 257F	IsBoxDrawing
2580 - 259F	IsBlockElements
25A0 - 25FF	IsGeometricShapes
2600 - 26FF	IsMiscellaneousSymbols
2700 - 27BF	IsDingbats
27C0 - 27EF	IsMiscellaneousMathematicalSymbols-A
27F0 - 27FF	IsSupplementalArrows-A
2800 - 28FF	IsBraillePatterns
2900 - 297F	IsSupplementalArrows-B
2980 - 29FF	IsMiscellaneousMathematicalSymbols-B
2A00 - 2AFF	IsSupplementalMathematicalOperators
2B00 - 2BFF	IsMiscellaneousSymbolsandArrows
2E80 - 2EFF	IsCJKRadicalsSupplement
2F00 - 2FDF	IsKangxiRadicals
2FF0 - 2FFF	IsIdeographicDescriptionCharacters
3000 - 303F	IsCJKSymbolsandPunctuation
3040 - 309F	IsHiragana
30A0 - 30FF	IsKatakana
3100 - 312F	IsBopomofo
3130 - 318F	IsHangulCompatibilityJamo
3190 - 319F	IsKanbun
31A0 - 31BF	IsBopomofoExtended

CODE POINT RANGE	BLOCK NAME
31F0 - 31FF	IsKatakanaPhoneticExtensions
3200 - 32FF	IsEnclosedCJKLettersandMonths
3300 - 33FF	IsCJKCompatibility
3400 - 4DBF	IsCJKUnifiedIdeographsExtensionA
4DC0 - 4DFF	IsYijingHexagramSymbols
4E00 - 9FFF	IsCJKUnifiedIdeographs
A000 - A48F	IsYiSyllables
A490 - A4CF	IsYiRadicals
AC00 - D7AF	IsHangulSyllables
D800 - DB7F	IsHighSurrogates
DB80 - DBFF	IsHighPrivateUseSurrogates
DC00 - DFFF	IsLowSurrogates
E000 - F8FF	IsPrivateUse or IsPrivateUseArea
F900 - FAFF	IsCJKCompatibilityIdeographs
FB00 - FB4F	IsAlphabeticPresentationForms
FB50 - FDFF	IsArabicPresentationForms-A
FE00 - FEOF	IsVariationSelectors
FE20 - FE2F	IsCombiningHalfMarks
FE30 - FE4F	IsCJKCompatibilityForms
FE50 - FE6F	IsSmallFormVariants
FE70 - FEFF	IsArabicPresentationForms-B
FF00 - FFFF	IsHalfwidthandFullwidthForms
FFFF - FFFF	IsSpecials

Character class subtraction: [base_group - [excluded_group]]

A character class defines a set of characters. Character class subtraction yields a set of characters that is the result of excluding the characters in one character class from another character class.

A character class subtraction expression has the following form:

```
[ base_group -[ excluded_group ]]
```

The square brackets ([]) and hyphen (-) are mandatory. The *base_group* is a [positive character group](#) or a [negative character group](#). The *excluded_group* component is another positive or negative character group, or another character class subtraction expression (that is, you can nest character class subtraction expressions).

For example, suppose you have a base group that consists of the character range from "a" through "z". To define the set of characters that consists of the base group except for the character "m", use `[a-z-[m]]`. To define the set of characters that consists of the base group except for the set of characters "d", "j", and "p", use `[a-z-[dp]]`. To define the set of characters that consists of the base group except for the character range from "m" through "p", use `[a-z-[m-p]]`.

Consider the nested character class subtraction expression, `[a-z-[d-w-[m-o]]]`. The expression is evaluated from the innermost character range outward. First, the character range from "m" through "o" is subtracted from the character range "d" through "w", which yields the set of characters from "d" through "l" and "p" through "w". That set is then subtracted from the character range from "a" through "z", which yields the set of characters `[abcmonxyz]`.

You can use any character class with character class subtraction. To define the set of characters that consists of all Unicode characters from \u0000 through \uFFFF except white-space characters (\s), the characters in the punctuation general category (\p{P}), the characters in the `IsGreek` named block (\p{IsGreek}), and the Unicode NEXT LINE control character (\x85), use `[\u0000-\uFFFF-\s\p{P}\p{IsGreek}\x85]`.

Choose character classes for a character class subtraction expression that will yield useful results. Avoid an expression that yields an empty set of characters, which cannot match anything, or an expression that is equivalent to the original base group. For example, the empty set is the result of the expression

`[\p{IsBasicLatin}-[\x00-\x7F]]`, which subtracts all characters in the `IsBasicLatin` character range from the `IsBasicLatin` general category. Similarly, the original base group is the result of the expression `[a-z-[0-9]]`.

This is because the base group, which is the character range of letters from "a" through "z", does not contain any characters in the excluded group, which is the character range of decimal digits from "0" through "9".

The following example defines a regular expression, `^[0-9-[2468]]+$`, that matches zero and odd digits in an input string. The regular expression is interpreted as shown in the following table.

ELEMENT	DESCRIPTION
<code>^</code>	Begin the match at the start of the input string.
<code>[0-9-[2468]]+</code>	Match one or more occurrences of any character from 0 to 9 except for 2, 4, 6, and 8. In other words, match one or more occurrences of zero or an odd digit.
<code>\$</code>	End the match at the end of the input string.

```

using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string[] inputs = { "123", "13579753", "3557798", "335599901" };
        string pattern = @"^-[0-9-[2468]]+$";

        foreach (string input in inputs)
        {
            Match match = Regex.Match(input, pattern);
            if (match.Success)
                Console.WriteLine(match.Value);
        }
    }
}

// The example displays the following output:
//      13579753
//      335599901

```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim inputs() As String = {"123", "13579753", "3557798", "335599901"}
        Dim pattern As String = "^[0-9-[2468]]+$"

        For Each input As String In inputs
            Dim match As Match = Regex.Match(input, pattern)
            If match.Success Then Console.WriteLine(match.Value)
        Next
    End Sub
End Module

' The example displays the following output:
'      13579753
'      335599901

```

See also

- [GetUnicodeCategory](#)
- [Regular Expression Language - Quick Reference](#)
- [Regular Expression Options](#)

Anchors in Regular Expressions

9/20/2022 • 21 minutes to read • [Edit Online](#)

Anchors, or atomic zero-width assertions, specify a position in the string where a match must occur. When you use an anchor in your search expression, the regular expression engine does not advance through the string or consume characters; it looks for a match in the specified position only. For example, `^` specifies that the match must start at the beginning of a line or string. Therefore, the regular expression `^http:` matches "http:" only when it occurs at the beginning of a line. The following table lists the anchors supported by the regular expressions in .NET.

ANCHOR	DESCRIPTION
<code>^</code>	By default, the match must occur at the beginning of the string; in multiline mode, it must occur at the beginning of the line. For more information, see Start of String or Line .
<code>\$</code>	By default, the match must occur at the end of the string or before <code>\n</code> at the end of the string; in multiline mode, it must occur at the end of the line or before <code>\n</code> at the end of the line. For more information, see End of String or Line .
<code>\A</code>	The match must occur at the beginning of the string only (no multiline support). For more information, see Start of String Only .
<code>\Z</code>	The match must occur at the end of the string, or before <code>\n</code> at the end of the string. For more information, see End of String or Before Ending Newline .
<code>\z</code>	The match must occur at the end of the string only. For more information, see End of String Only .
<code>\G</code>	The match must start at the position where the previous match ended, or if there was no previous match, at the position in the string where matching started. For more information, see Contiguous Matches .
<code>\b</code>	The match must occur on a word boundary. For more information, see Word Boundary .
<code>\B</code>	The match must not occur on a word boundary. For more information, see Non-Word Boundary .

Start of String or Line: ^

By default, the `^` anchor specifies that the following pattern must begin at the first character position of the string. If you use `^` with the `RegexOptions.Multiline` option (see [Regular Expression Options](#)), the match must occur at the beginning of each line.

The following example uses the `^` anchor in a regular expression that extracts information about the years during which some professional baseball teams existed. The example calls two overloads of the `Regex.Matches` method:

- The call to the [Matches\(String, String\)](#) overload finds only the first substring in the input string that matches the regular expression pattern.
- The call to the [Matches\(String, String, RegexOptions\)](#) overload with the `options` parameter set to [RegexOptions.Multiline](#) finds all five substrings.

```

using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string input = "Brooklyn Dodgers, National League, 1911, 1912, 1932-1957\n" +
                      "Chicago Cubs, National League, 1903-present\n" +
                      "Detroit Tigers, American League, 1901-present\n" +
                      "New York Giants, National League, 1885-1957\n" +
                      "Washington Senators, American League, 1901-1960\n";
        string pattern = @"^((\w+(\s?)){2,}),\s(\w+\s\w+),(\s\d{4}(-(\d{4}|present))?,?)+" ;
        Match match;

        match = Regex.Match(input, pattern);
        while (match.Success)
        {
            Console.Write("The {0} played in the {1} in",
                         match.Groups[1].Value, match.Groups[4].Value);
            foreach (Capture capture in match.Groups[5].Captures)
                Console.Write(capture.Value);

            Console.WriteLine(".");
            match = match.NextMatch();
        }
        Console.WriteLine();

        match = Regex.Match(input, pattern, RegexOptions.Multiline);
        while (match.Success)
        {
            Console.Write("The {0} played in the {1} in",
                         match.Groups[1].Value, match.Groups[4].Value);
            foreach (Capture capture in match.Groups[5].Captures)
                Console.Write(capture.Value);

            Console.WriteLine(".");
            match = match.NextMatch();
        }
        Console.WriteLine();
    }
}

// The example displays the following output:
//      The Brooklyn Dodgers played in the National League in 1911, 1912, 1932-1957.
//
//      The Brooklyn Dodgers played in the National League in 1911, 1912, 1932-1957.
//      The Chicago Cubs played in the National League in 1903-present.
//      The Detroit Tigers played in the American League in 1901-present.
//      The New York Giants played in the National League in 1885-1957.
//      The Washington Senators played in the American League in 1901-1960.

```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim input As String = "Brooklyn Dodgers, National League, 1911, 1912, 1932-1957" + vbCrLf +
                            "Chicago Cubs, National League, 1903-present" + vbCrLf +
                            "Detroit Tigers, American League, 1901-present" + vbCrLf +
                            "New York Giants, National League, 1885-1957" + vbCrLf +
                            "Washington Senators, American League, 1901-1960" + vbCrLf

        Dim pattern As String = "^((\w+(\s?)){2,}),\s(\w+\s\w+),(\s\d{4}(-(\d{4}|present))?,?)+" 
        Dim match As Match

        match = Regex.Match(input, pattern)
        Do While match.Success
            Console.WriteLine("The {0} played in the {1} in",
                               match.Groups(1).Value, match.Groups(4).Value)
            For Each capture As Capture In match.Groups(5).Captures
                Console.WriteLine(capture.Value)
            Next
            Console.WriteLine(".")
            match = match.NextMatch()
        Loop
        Console.WriteLine()

        match = Regex.Match(input, pattern, RegexOptions.Multiline)
        Do While match.Success
            Console.WriteLine("The {0} played in the {1} in",
                               match.Groups(1).Value, match.Groups(4).Value)
            For Each capture As Capture In match.Groups(5).Captures
                Console.WriteLine(capture.Value)
            Next
            Console.WriteLine(".")
            match = match.NextMatch()
        Loop
        Console.WriteLine()
    End Sub
End Module
' The example displays the following output:
'   The Brooklyn Dodgers played in the National League in 1911, 1912, 1932-1957.
'
'   The Brooklyn Dodgers played in the National League in 1911, 1912, 1932-1957.
'   The Chicago Cubs played in the National League in 1903-present.
'   The Detroit Tigers played in the American League in 1901-present.
'   The New York Giants played in the National League in 1885-1957.
'   The Washington Senators played in the American League in 1901-1960.

```

The regular expression pattern `^((\w+(\s?)){2,}),\s(\w+\s\w+),(\s\d{4}(-(\d{4}|present))?,?)+` is defined as shown in the following table.

PATTERN	DESCRIPTION
<code>^</code>	Begin the match at the beginning of the input string (or the beginning of the line if the method is called with the RegexOptions.Multiline option).
<code>((\w+(\s?)){2,})</code>	Match one or more word characters followed either by zero or by one space at least two times. This is the first capturing group. This expression also defines a second and third capturing group: The second consists of the captured word, and the third consists of the captured white space.
<code>,\s</code>	Match a comma followed by a white-space character.

PATTERN	DESCRIPTION
(\w+\s\w+)	Match one or more word characters followed by a space, followed by one or more word characters. This is the fourth capturing group.
,	Match a comma.
\s\d{4}	Match a space followed by four decimal digits.
(-(\d{4} present))?	Match zero or one occurrence of a hyphen followed by four decimal digits or the string "present". This is the sixth capturing group. It also includes a seventh capturing group.
,?	Match zero or one occurrence of a comma.
(\s\d{4}(-(\d{4} present))?,?)+	Match one or more occurrences of the following: a space, four decimal digits, zero or one occurrence of a hyphen followed by four decimal digits or the string "present", and zero or one comma. This is the fifth capturing group.

End of String or Line: \$

The `$` anchor specifies that the preceding pattern must occur at the end of the input string, or before `\n` at the end of the input string.

If you use `$` with the [RegexOptions.Multiline](#) option, the match can also occur at the end of a line. Note that `$` matches `\n` but does not match `\r\n` (the combination of carriage return and newline characters, or CR/LF). To match the CR/LF character combination, include `\r?\$` in the regular expression pattern.

The following example adds the `$` anchor to the regular expression pattern used in the example in the [Start of String or Line](#) section. When used with the original input string, which includes five lines of text, the `Regex.Matches(String, String)` method is unable to find a match, because the end of the first line does not match the `$` pattern. When the original input string is split into a string array, the `Regex.Matches(String, String)` method succeeds in matching each of the five lines. When the `Regex.Matches(String, String, RegexOptions)` method is called with the `options` parameter set to `RegexOptions.Multiline`, no matches are found because the regular expression pattern does not account for the carriage return element (`\u000D`). However, when the regular expression pattern is modified by replacing `$` with `\r?\$`, calling the `Regex.Matches(String, String, RegexOptions)` method with the `options` parameter set to `RegexOptions.Multiline` again finds five matches.

```
using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string cr = Environment.NewLine;
        string input = "Brooklyn Dodgers, National League, 1911, 1912, 1932-1957" + cr +
                      "Chicago Cubs, National League, 1903-present" + cr +
                      "Detroit Tigers, American League, 1901-present" + cr +
                      "New York Giants, National League, 1885-1957" + cr +
                      "Washington Senators, American League, 1901-1960" + cr;
        Match match;

        string basePattern = @"^((\w+(\s?)){2,}),\s(\w+\s\w+),(\s\d{4}(-(\d{4}|present))?,?)+" ;
        string pattern = basePattern + "$";
        Console.WriteLine("Attempting to match the entire input string.");
    }
}
```

```

    Console.WriteLine("Attempting to match the entire input string.");
    match = Regex.Match(input, pattern);
    while (match.Success)
    {
        Console.Write("The {0} played in the {1} in",
                     match.Groups[1].Value, match.Groups[4].Value);
        foreach (Capture capture in match.Groups[5].Captures)
            Console.Write(capture.Value);

        Console.WriteLine(".");
        match = match.NextMatch();
    }
    Console.WriteLine();

    string[] teams = input.Split(new String[] { cr }, StringSplitOptions.RemoveEmptyEntries);
    Console.WriteLine("Attempting to match each element in a string array:");
    foreach (string team in teams)
    {
        match = Regex.Match(team, pattern);
        if (match.Success)
        {
            Console.Write("The {0} played in the {1} in",
                         match.Groups[1].Value, match.Groups[4].Value);
            foreach (Capture capture in match.Groups[5].Captures)
                Console.Write(capture.Value);
            Console.WriteLine(".");
        }
    }
    Console.WriteLine();

    Console.WriteLine("Attempting to match each line of an input string with '$':");
    match = Regex.Match(input, pattern, RegexOptions.Multiline);
    while (match.Success)
    {
        Console.Write("The {0} played in the {1} in",
                     match.Groups[1].Value, match.Groups[4].Value);
        foreach (Capture capture in match.Groups[5].Captures)
            Console.Write(capture.Value);

        Console.WriteLine(".");
        match = match.NextMatch();
    }
    Console.WriteLine();

    pattern = basePattern + "\r?$";
    Console.WriteLine(@"Attempting to match each line of an input string with '\r?$':");
    match = Regex.Match(input, pattern, RegexOptions.Multiline);
    while (match.Success)
    {
        Console.Write("The {0} played in the {1} in",
                     match.Groups[1].Value, match.Groups[4].Value);
        foreach (Capture capture in match.Groups[5].Captures)
            Console.Write(capture.Value);

        Console.WriteLine(".");
        match = match.NextMatch();
    }
    Console.WriteLine();
}

// The example displays the following output:
//   Attempting to match the entire input string:
//
//   Attempting to match each element in a string array:
//   The Brooklyn Dodgers played in the National League in 1911, 1912, 1932-1957.
//   The Chicago Cubs played in the National League in 1903-present.
//   The Detroit Tigers played in the American League in 1901-present.
//   The New York Giants played in the National League in 1885-1957.
//   The Washington Senators played in the American League in 1901-1960.

```

```

// Attempting to match each line of an input string with '$':
//
// Attempting to match each line of an input string with '\r?$':
// The Brooklyn Dodgers played in the National League in 1911, 1912, 1932-1957.
// The Chicago Cubs played in the National League in 1903-present.
// The Detroit Tigers played in the American League in 1901-present.
// The New York Giants played in the National League in 1885-1957.
// The Washington Senators played in the American League in 1901-1960.

```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim input As String = "Brooklyn Dodgers, National League, 1911, 1912, 1932-1957" + vbCrLf +
                            "Chicago Cubs, National League, 1903-present" + vbCrLf +
                            "Detroit Tigers, American League, 1901-present" + vbCrLf +
                            "New York Giants, National League, 1885-1957" + vbCrLf +
                            "Washington Senators, American League, 1901-1960" + vbCrLf

        Dim basePattern As String = "^((\w+(\s?)){2,}),\s(\w+\s\w+),(\s\d{4}(-(\d{4}|present))?,?)"
        Dim match As Match

        Dim pattern As String = basePattern + "$"
        Console.WriteLine("Attempting to match the entire input string:")
        match = Regex.Match(input, pattern)
        Do While match.Success
            Console.Write("The {0} played in the {1} in",
                         match.Groups(1).Value, match.Groups(4).Value)
            For Each capture As Capture In match.Groups(5).Captures
                Console.Write(capture.Value)
            Next
            Console.WriteLine(".")
            match = match.NextMatch()
        Loop
        Console.WriteLine()

        Dim teams() As String = input.Split(New String() {vbCrLf}, StringSplitOptions.RemoveEmptyEntries)
        Console.WriteLine("Attempting to match each element in a string array:")
        For Each team As String In teams
            match = Regex.Match(team, pattern)
            If match.Success Then
                Console.Write("The {0} played in the {1} in",
                             match.Groups(1).Value, match.Groups(4).Value)
                For Each capture As Capture In match.Groups(5).Captures
                    Console.Write(capture.Value)
                Next
                Console.WriteLine(".")
            End If
        Next
        Console.WriteLine()

        Console.WriteLine("Attempting to match each line of an input string with '$':")
        match = Regex.Match(input, pattern, RegexOptions.Multiline)
        Do While match.Success
            Console.Write("The {0} played in the {1} in",
                         match.Groups(1).Value, match.Groups(4).Value)
            For Each capture As Capture In match.Groups(5).Captures
                Console.Write(capture.Value)
            Next
            Console.WriteLine(".")
            match = match.NextMatch()
        Loop
        Console.WriteLine()

        pattern = basePattern + "\r?$"
        Console.WriteLine("Attempting to match each line of an input string with '\r?$':")
        match = Regex.Match(input, pattern, RegexOptions.Multiline)
    End Sub
End Module

```

```

match = Regex.Match(input, pattern, RegexOptions.Multiline)
Do While match.Success
    Console.WriteLine("The {0} played in the {1} in",
                      match.Groups(1).Value, match.Groups(4).Value)
    For Each capture As Capture In match.Groups(5).Captures
        Console.WriteLine(capture.Value)
    Next
    Console.WriteLine(".")

    match = match.NextMatch()
Loop
Console.WriteLine()
End Sub
End Module
' The example displays the following output:
'   Attempting to match the entire input string:
'
'   Attempting to match each element in a string array:
'   The Brooklyn Dodgers played in the National League in 1911, 1912, 1932-1957.
'   The Chicago Cubs played in the National League in 1903-present.
'   The Detroit Tigers played in the American League in 1901-present.
'   The New York Giants played in the National League in 1885-1957.
'   The Washington Senators played in the American League in 1901-1960.
'
'   Attempting to match each line of an input string with '$':
'
'   Attempting to match each line of an input string with '\r?$':
'   The Brooklyn Dodgers played in the National League in 1911, 1912, 1932-1957.
'   The Chicago Cubs played in the National League in 1903-present.
'   The Detroit Tigers played in the American League in 1901-present.
'   The New York Giants played in the National League in 1885-1957.
'   The Washington Senators played in the American League in 1901-1960.

```

Start of String Only: \A

The `\A` anchor specifies that a match must occur at the beginning of the input string. It is identical to the `^` anchor, except that `\A` ignores the `RegexOptions.Multiline` option. Therefore, it can only match the start of the first line in a multiline input string.

The following example is similar to the examples for the `^` and `$` anchors. It uses the `\A` anchor in a regular expression that extracts information about the years during which some professional baseball teams existed. The input string includes five lines. The call to the `Regex.Matches(String, String, RegexOptions)` method finds only the first substring in the input string that matches the regular expression pattern. As the example shows, the `Multiline` option has no effect.

```

using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string input = "Brooklyn Dodgers, National League, 1911, 1912, 1932-1957\n" +
                      "Chicago Cubs, National League, 1903-present\n" +
                      "Detroit Tigers, American League, 1901-present\n" +
                      "New York Giants, National League, 1885-1957\n" +
                      "Washington Senators, American League, 1901-1960\n";

        string pattern = @"\A((\w+(\s?)){2,}),\s(\w+\s\w+),(\s\d{4}(-(\d{4}|present))?,?)+";

        Match match = Regex.Match(input, pattern, RegexOptions.Multiline);
        while (match.Success)
        {
            Console.Write("The {0} played in the {1} in",
                         match.Groups[1].Value, match.Groups[4].Value);
            foreach (Capture capture in match.Groups[5].Captures)
                Console.Write(capture.Value);

            Console.WriteLine(".");
            match = match.NextMatch();
        }
        Console.WriteLine();
    }
}

// The example displays the following output:
//     The Brooklyn Dodgers played in the National League in 1911, 1912, 1932-1957.

```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim input As String = "Brooklyn Dodgers, National League, 1911, 1912, 1932-1957" + vbCrLf +
                             "Chicago Cubs, National League, 1903-present" + vbCrLf +
                             "Detroit Tigers, American League, 1901-present" + vbCrLf +
                             "New York Giants, National League, 1885-1957" + vbCrLf +
                             "Washington Senators, American League, 1901-1960" + vbCrLf

        Dim pattern As String = "\A((\w+(\s?)){2,}),\s(\w+\s\w+),(\s\d{4}(-(\d{4}|present))?,?)+""

        Dim match As Match = Regex.Match(input, pattern, RegexOptions.Multiline)
        Do While match.Success
            Console.Write("The {0} played in the {1} in",
                         match.Groups(1).Value, match.Groups(4).Value)
            For Each capture As Capture In match.Groups(5).Captures
                Console.Write(capture.Value)
            Next
            Console.WriteLine(".")
            match = match.NextMatch()
        Loop
        Console.WriteLine()
    End Sub
End Module
' The example displays the following output:
'     The Brooklyn Dodgers played in the National League in 1911, 1912, 1932-1957.

```

End of String or Before Ending Newline: \Z

The `\Z` anchor specifies that a match must occur at the end of the input string, or before `\n` at the end of the

input string. It is identical to the `\$` anchor, except that `\z` ignores the `RegexOptions.Multiline` option. Therefore, in a multiline string, it can only match the end of the last line, or the last line before `\n`.

Note that `\z` matches `\n` but does not match `\r\n` (the CR/LF character combination). To match CR/LF, include `\r?\z` in the regular expression pattern.

The following example uses the `\z` anchor in a regular expression that is similar to the example in the [Start of String or Line](#) section, which extracts information about the years during which some professional baseball teams existed. The subexpression `\r?\z` in the regular expression

`^((\w+(\s?)){2,}),\s(\w+\s\w+),(\s\d{4}(-(\d{4}|present))?,?)+\r?\z` matches the end of a string, and also matches a string that ends with `\n` or `\r\n`. As a result, each element in the array matches the regular expression pattern.

```
using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string[] inputs = { "Brooklyn Dodgers, National League, 1911, 1912, 1932-1957",
                            "Chicago Cubs, National League, 1903-present" + Environment.NewLine,
                            "Detroit Tigers, American League, 1901-present" + Regex.Unescape(@"\n"),
                            "New York Giants, National League, 1885-1957",
                            "Washington Senators, American League, 1901-1960" + Environment.NewLine};
        string pattern = @"^((\w+(\s?)){2,}),\s(\w+\s\w+),(\s\d{4}(-(\d{4}|present))?,?)+\r?\z";

        foreach (string input in inputs)
        {
            Console.WriteLine(Regex.Escape(input));
            Match match = Regex.Match(input, pattern);
            if (match.Success)
                Console.WriteLine("  Match succeeded.");
            else
                Console.WriteLine("  Match failed.");
        }
    }
    // The example displays the following output:
    // Brooklyn\ Dodgers,\ National\ League,\ 1911,\ 1912,\ 1932-1957
    //     Match succeeded.
    // Chicago\ Cubs,\ National\ League,\ 1903-present\r\n
    //     Match succeeded.
    // Detroit\ Tigers,\ American\ League,\ 1901-present\n
    //     Match succeeded.
    // New\ York\ Giants,\ National\ League,\ 1885-1957
    //     Match succeeded.
    // Washington\ Senators,\ American\ League,\ 1901-1960\r\n
    //     Match succeeded.
```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim inputs() As String = {"Brooklyn Dodgers, National League, 1911, 1912, 1932-1957",
                                  "Chicago Cubs, National League, 1903-present" + vbCrLf,
                                  "Detroit Tigers, American League, 1901-present" + vbLf,
                                  "New York Giants, National League, 1885-1957",
                                  "Washington Senators, American League, 1901-1960" + vbCrLf}
        Dim pattern As String = "^((\w+(\s?)){2,}),\s(\w+\s\w+),(\s\d{4}(-(\d{4}|present))?,?)+'\r?\z"

        For Each input As String In inputs
            Console.WriteLine(Regex.Escape(input))
            Dim match As Match = Regex.Match(input, pattern)
            If match.Success Then
                Console.WriteLine("    Match succeeded.")
            Else
                Console.WriteLine("    Match failed.")
            End If
        Next
    End Sub
End Module
' The example displays the following output:
' Brooklyn\ Dodgers,\ National\ League,\ 1911,\ 1912,\ 1932-1957
'     Match succeeded.
' Chicago\ Cubs,\ National\ League,\ 1903-present\r\n
'     Match succeeded.
' Detroit\ Tigers,\ American\ League,\ 1901-present\n
'     Match succeeded.
' New\ York\ Giants,\ National\ League,\ 1885-1957
'     Match succeeded.
' Washington\ Senators,\ American\ League,\ 1901-1960\r\n
'     Match succeeded.

```

End of String Only: \z

The `\z` anchor specifies that a match must occur at the end of the input string. Like the `$` language element, `\z` ignores the [RegexOptions.Multiline](#) option. Unlike the `\z` language element, `\z` does not match a `\n` character at the end of a string. Therefore, it can only match the last line of the input string.

The following example uses the `\z` anchor in a regular expression that is otherwise identical to the example in the previous section, which extracts information about the years during which some professional baseball teams existed. The example tries to match each of five elements in a string array with the regular expression pattern `^((\w+(\s?)){2,}),\s(\w+\s\w+),(\s\d{4}(-(\d{4}|present))?,?)+'\r?\z`. Two of the strings end with carriage return and line feed characters, one ends with a line feed character, and two end with neither a carriage return nor a line feed character. As the output shows, only the strings without a carriage return or line feed character match the pattern.

```
using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string[] inputs = { "Brooklyn Dodgers, National League, 1911, 1912, 1932-1957",
                            "Chicago Cubs, National League, 1903-present" + Environment.NewLine,
                            "Detroit Tigers, American League, 1901-present\n",
                            "New York Giants, National League, 1885-1957",
                            "Washington Senators, American League, 1901-1960" + Environment.NewLine };
        string pattern = @"^((\w+(\s?)){2,}),\s(\w+\s\w+),(\s\d{4}(-(\d{4}|present))?,?)+\r?\z";

        foreach (string input in inputs)
        {
            Console.WriteLine(Regex.Escape(input));
            Match match = Regex.Match(input, pattern);
            if (match.Success)
                Console.WriteLine("    Match succeeded.");
            else
                Console.WriteLine("    Match failed.");
        }
    }
}

// The example displays the following output:
// Brooklyn\ Dodgers,\ National\ League,\ 1911,\ 1912,\ 1932-1957
//     Match succeeded.
// Chicago\ Cubs,\ National\ League,\ 1903-present\r\n
//     Match failed.
// Detroit\ Tigers,\ American\ League,\ 1901-present\n
//     Match failed.
// New\ York\ Giants,\ National\ League,\ 1885-1957
//     Match succeeded.
// Washington\ Senators,\ American\ League,\ 1901-1960\r\n
//     Match failed.
```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim inputs() As String = {"Brooklyn Dodgers, National League, 1911, 1912, 1932-1957",
                                "Chicago Cubs, National League, 1903-present" + vbCrLf,
                                "Detroit Tigers, American League, 1901-present" + vbLf,
                                "New York Giants, National League, 1885-1957",
                                "Washington Senators, American League, 1901-1960" + vbCrLf}
        Dim pattern As String = "^((\w+(\s?)){2,}),\s(\w+\s\w+),(\s\d{4}(-(\d{4}|present))?,?)+" + vbCrLf

        For Each input As String In inputs
            Console.WriteLine(Regex.Escape(input))
            Dim match As Match = Regex.Match(input, pattern)
            If match.Success Then
                Console.WriteLine("    Match succeeded.")
            Else
                Console.WriteLine("    Match failed.")
            End If
        Next
    End Sub
End Module
' The example displays the following output:
' Brooklyn\ Dodgers,\ National\ League,\ 1911,\ 1912,\ 1932-1957
'     Match succeeded.
' Chicago\ Cubs,\ National\ League,\ 1903-present\r\n
'     Match failed.
' Detroit\ Tigers,\ American\ League,\ 1901-present\n
'     Match failed.
' New\ York\ Giants,\ National\ League,\ 1885-1957
'     Match succeeded.
' Washington\ Senators,\ American\ League,\ 1901-1960\r\n
'     Match failed.

```

Contiguous Matches: \G

The `\G` anchor specifies that a match must occur at the point where the previous match ended, or if there was no previous match, at the position in the string where matching started. When you use this anchor with the [Regex.Matches](#) or [Match.NextMatch](#) method, it ensures that all matches are contiguous.

TIP

Typically, you place a `\G` anchor at the left end of your pattern. In the uncommon case you're performing a right-to-left search, place the `\G` anchor at the right end of your pattern.

The following example uses a regular expression to extract the names of rodent species from a comma-delimited string.

```

using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string input = "capybara,squirrel,chipmunk,porcupine,gopher," +
                      "beaver,groundhog,hamster,guinea pig,gerbil," +
                      "chinchilla,prairie dog,mouse,rat";
        string pattern = @"\G(\w+\s?\w*),?";
        Match match = Regex.Match(input, pattern);
        while (match.Success)
        {
            Console.WriteLine(match.Groups[1].Value);
            match = match.NextMatch();
        }
    }
}

// The example displays the following output:
//      capybara
//      squirrel
//      chipmunk
//      porcupine
//      gopher
//      beaver
//      groundhog
//      hamster
//      guinea pig
//      gerbil
//      chinchilla
//      prairie dog
//      mouse
//      rat

```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim input As String = "capybara,squirrel,chipmunk,porcupine,gopher," +
                             "beaver,groundhog,hamster,guinea pig,gerbil," +
                             "chinchilla,prairie dog,mouse,rat"
        Dim pattern As String = "\G(\w+\s?\w*),?"
        Dim match As Match = Regex.Match(input, pattern)
        Do While match.Success
            Console.WriteLine(match.Groups(1).Value)
            match = match.NextMatch()
        Loop
    End Sub
End Module

' The example displays the following output:
'      capybara
'      squirrel
'      chipmunk
'      porcupine
'      gopher
'      beaver
'      groundhog
'      hamster
'      guinea pig
'      gerbil
'      chinchilla
'      prairie dog
'      mouse
'      rat

```

The regular expression `\G(\w+\s?\w*),?` is interpreted as shown in the following table.

PATTERN	DESCRIPTION
<code>\G</code>	Begin where the last match ended.
<code>\w+</code>	Match one or more word characters.
<code>\s?</code>	Match zero or one space.
<code>\w*</code>	Match zero or more word characters.
<code>(\w+\s?\w*)</code>	Match one or more word characters followed by zero or one space, followed by zero or more word characters. This is the first capturing group.
<code>,?</code>	Match zero or one occurrence of a literal comma character.

Word Boundary: \b

The `\b` anchor specifies that the match must occur on a boundary between a word character (the `\w` language element) and a non-word character (the `\W` language element). Word characters consist of alphanumeric characters and underscores; a non-word character is any character that is not alphanumeric or an underscore. (For more information, see [Character Classes](#).) The match may also occur on a word boundary at the beginning or end of the string.

The `\b` anchor is frequently used to ensure that a subexpression matches an entire word instead of just the beginning or end of a word. The regular expression `\bare\w*\b` in the following example illustrates this usage. It matches any word that begins with the substring "are". The output from the example also illustrates that `\b` matches both the beginning and the end of the input string.

```
using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string input = "area bare arena mare";
        string pattern = @"\bare\w*\b";
        Console.WriteLine("Words that begin with 'are':");
        foreach (Match match in Regex.Matches(input, pattern))
            Console.WriteLine("{0} found at position {1}",
                             match.Value, match.Index);
    }
}
// The example displays the following output:
//      Words that begin with 'are':
//      'area' found at position 0
//      'arena' found at position 10
```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim input As String = "area bare arena mare"
        Dim pattern As String = "\bare\w*\b"
        Console.WriteLine("Words that begin with 'are':")
        For Each match As Match In Regex.Matches(input, pattern)
            Console.WriteLine("{0} found at position {1}",
                match.Value, match.Index)
        Next
    End Sub
End Module
' The example displays the following output:
'     Words that begin with 'are':
'     'area' found at position 0
'     'arena' found at position 10

```

The regular expression pattern is interpreted as shown in the following table.

PATTERN	DESCRIPTION
\b	Begin the match at a word boundary.
are	Match the substring "are".
\w*	Match zero or more word characters.
\b	End the match at a word boundary.

Non-Word Boundary: \B

The \B anchor specifies that the match must not occur on a word boundary. It is the opposite of the \b anchor.

The following example uses the \B anchor to locate occurrences of the substring "qu" in a word. The regular expression pattern \Bqu\w+ matches a substring that begins with a "qu" that does not start a word and that continues to the end of the word.

```

using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string input = "equity queen equip acquaint quiet";
        string pattern = @"\Bqu\w+";
        foreach (Match match in Regex.Matches(input, pattern))
            Console.WriteLine("{0} found at position {1}",
                match.Value, match.Index);
    }
}
// The example displays the following output:
//     'quity' found at position 1
//     'quip' found at position 14
//     'quaint' found at position 21

```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim input As String = "equity queen equip acquaint quiet"
        Dim pattern As String = "\Bqu\w+"
        For Each match As Match In Regex.Matches(input, pattern)
            Console.WriteLine("{0} found at position {1}",
                match.Value, match.Index)
        Next
    End Sub
End Module
' The example displays the following output:
'     'quity' found at position 1
'     'quip' found at position 14
'     'quaint' found at position 21

```

The regular expression pattern is interpreted as shown in the following table.

PATTERN	DESCRIPTION
\B	Do not begin the match at a word boundary.
qu	Match the substring "qu".
\w+	Match one or more word characters.

See also

- [Regular Expression Language - Quick Reference](#)
- [Regular Expression Options](#)

Grouping Constructs in Regular Expressions

9/20/2022 • 37 minutes to read • [Edit Online](#)

Grouping constructs delineate the subexpressions of a regular expression and capture the substrings of an input string. You can use grouping constructs to do the following:

- Match a subexpression that is repeated in the input string.
- Apply a quantifier to a subexpression that has multiple regular expression language elements. For more information about quantifiers, see [Quantifiers](#).
- Include a subexpression in the string that is returned by the [Regex.Replace](#) and [Match.Result](#) methods.
- Retrieve individual subexpressions from the [Match.Groups](#) property and process them separately from the matched text as a whole.

The following table lists the grouping constructs supported by the .NET regular expression engine and indicates whether they are capturing or non-capturing.

GROUPING CONSTRUCT	CAPTURING OR NONCAPTURING
Matched subexpressions	Capturing
Named matched subexpressions	Capturing
Balancing group definitions	Capturing
Noncapturing groups	Noncapturing
Group options	Noncapturing
Zero-width positive lookahead assertions	Noncapturing
Zero-width negative lookahead assertions	Noncapturing
Zero-width positive lookbehind assertions	Noncapturing
Zero-width negative lookbehind assertions	Noncapturing
Atomic groups	Noncapturing

For information on groups and the regular expression object model, see [Grouping constructs and regular expression objects](#).

Matched Subexpressions

The following grouping construct captures a matched subexpression:

(*subexpression*)

where *subexpression* is any valid regular expression pattern. Captures that use parentheses are numbered automatically from left to right based on the order of the opening parentheses in the regular expression,

starting from one. The capture that is numbered zero is the text matched by the entire regular expression pattern.

NOTE

By default, the `(subexpression)` language element captures the matched subexpression. But if the `RegexOptions` parameter of a regular expression pattern matching method includes the `RegexOptions.ExplicitCapture` flag, or if the `n` option is applied to this subexpression (see [Group options](#) later in this topic), the matched subexpression is not captured.

You can access captured groups in four ways:

- By using the backreference construct within the regular expression. The matched subexpression is referenced in the same regular expression by using the syntax `\ number`, where `number` is the ordinal number of the captured subexpression.
- By using the named backreference construct within the regular expression. The matched subexpression is referenced in the same regular expression by using the syntax `\k< name >`, where `name` is the name of a capturing group, or `\k< number >`, where `number` is the ordinal number of a capturing group. A capturing group has a default name that is identical to its ordinal number. For more information, see [Named matched subexpressions](#) later in this topic.
- By using the `$ number` replacement sequence in a `Regex.Replace` or `Match.Result` method call, where `number` is the ordinal number of the captured subexpression.
- Programmatically, by using the `GroupCollection` object returned by the `Match.Groups` property. The member at position zero in the collection represents the entire regular expression match. Each subsequent member represents a matched subexpression. For more information, see the [Grouping Constructs and Regular Expression Objects](#) section.

The following example illustrates a regular expression that identifies duplicated words in text. The regular expression pattern's two capturing groups represent the two instances of the duplicated word. The second instance is captured to report its starting position in the input string.

```
using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string pattern = @"(\w+)\s(\1)\W";
        string input = "He said that that was the the correct answer.";
        foreach (Match match in Regex.Matches(input, pattern, RegexOptions.IgnoreCase))
            Console.WriteLine("Duplicate '{0}' found at positions {1} and {2}.",
                match.Groups[1].Value, match.Groups[1].Index, match.Groups[2].Index);
    }
}
// The example displays the following output:
//      Duplicate 'that' found at positions 8 and 13.
//      Duplicate 'the' found at positions 22 and 26.
```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim pattern As String = "(\w+)\s(\1)\W"
        Dim input As String = "He said that that was the the correct answer."
        For Each match As Match In Regex.Matches(input, pattern, RegexOptions.IgnoreCase)
            Console.WriteLine("Duplicate '{0}' found at positions {1} and {2}.", _
                match.Groups(1).Value, match.Groups(1).Index, match.Groups(2).Index)
        Next
    End Sub
End Module
' The example displays the following output:
'     Duplicate 'that' found at positions 8 and 13.
'     Duplicate 'the' found at positions 22 and 26.

```

The regular expression pattern is the following:

(\w+)\s(\1)\W

The following table shows how the regular expression pattern is interpreted.

PATTERN	DESCRIPTION
(\w+)	Match one or more word characters. This is the first capturing group.
\s	Match a white-space character.
(\1)	Match the string in the first captured group. This is the second capturing group. The example assigns it to a captured group so that the starting position of the duplicate word can be retrieved from the <code>Match.Index</code> property.
\W	Match a non-word character, including white space and punctuation. This prevents the regular expression pattern from matching a word that starts with the word from the first captured group.

Named Matched Subexpressions

The following grouping construct captures a matched subexpression and lets you access it by name or by number:

(?<name>subexpression)

or:

(?'name' subexpression)

where *name* is a valid group name, and *subexpression* is any valid regular expression pattern. *name* must not contain any punctuation characters and cannot begin with a number.

NOTE

If the [RegexOptions](#) parameter of a regular expression pattern matching method includes the [RegexOptions.ExplicitCapture](#) flag, or if the `n` option is applied to this subexpression (see [Group options](#) later in this topic), the only way to capture a subexpression is to explicitly name capturing groups.

You can access named captured groups in the following ways:

- By using the named backreference construct within the regular expression. The matched subexpression is referenced in the same regular expression by using the syntax `\k<name>`, where *name* is the name of the captured subexpression.
- By using the backreference construct within the regular expression. The matched subexpression is referenced in the same regular expression by using the syntax `\number`, where *number* is the ordinal number of the captured subexpression. Named matched subexpressions are numbered consecutively from left to right after matched subexpressions.
- By using the `${name}` replacement sequence in a [Regex.Replace](#) or [Match.Result](#) method call, where *name* is the name of the captured subexpression.
- By using the `$number` replacement sequence in a [Regex.Replace](#) or [Match.Result](#) method call, where *number* is the ordinal number of the captured subexpression.
- Programmatically, by using the [GroupCollection](#) object returned by the [Match.Groups](#) property. The member at position zero in the collection represents the entire regular expression match. Each subsequent member represents a matched subexpression. Named captured groups are stored in the collection after numbered captured groups.
- Programmatically, by providing the subexpression name to the [GroupCollection](#) object's indexer (in C#) or to its [Item\[\]](#) property (in Visual Basic).

A simple regular expression pattern illustrates how numbered (unnamed) and named groups can be referenced either programmatically or by using regular expression language syntax. The regular expression

`((?<One>abc)\d+)?(?<Two>xyz)(.*)` produces the following capturing groups by number and by name. The first capturing group (number 0) always refers to the entire pattern.

NUMBER	NAME	PATTERN
0	0 (default name)	<code>((?<One>abc)\d+)?(?<Two>xyz)(.*)</code>
1	1 (default name)	<code>((?<One>abc)\d+)</code>
2	2 (default name)	<code>(.*)</code>
3	One	<code>(?<One>abc)</code>
4	Two	<code>(?<Two>xyz)</code>

The following example illustrates a regular expression that identifies duplicated words and the word that immediately follows each duplicated word. The regular expression pattern defines two named subexpressions: `duplicateWord`, which represents the duplicated word; and `nextWord`, which represents the word that follows the duplicated word.

```

using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string pattern = @"(?<duplicateWord>\w+)\s\k<duplicateWord>\W(?<nextWord>\w+)";
        string input = "He said that that was the the correct answer.";
        foreach (Match match in Regex.Matches(input, pattern, RegexOptions.IgnoreCase))
            Console.WriteLine("A duplicate '{0}' at position {1} is followed by '{2}'.",
                match.Groups["duplicateWord"].Value, match.Groups["duplicateWord"].Index,
                match.Groups["nextWord"].Value);
    }
}
// The example displays the following output:
//      A duplicate 'that' at position 8 is followed by 'was'.
//      A duplicate 'the' at position 22 is followed by 'correct'.

```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim pattern As String = "(?<duplicateWord>\w+)\s\k<duplicateWord>\W(?<nextWord>\w+)"
        Dim input As String = "He said that that was the the correct answer."
        Console.WriteLine(Regex.Matches(input, pattern, RegexOptions.IgnoreCase).Count)
        For Each match As Match In Regex.Matches(input, pattern, RegexOptions.IgnoreCase)
            Console.WriteLine("A duplicate '{0}' at position {1} is followed by '{2}'.",
                match.Groups("duplicateWord").Value, match.Groups("duplicateWord").Index,
                match.Groups("nextWord").Value)
        Next
    End Sub
End Module
' The example displays the following output:
'      A duplicate 'that' at position 8 is followed by 'was'.
'      A duplicate 'the' at position 22 is followed by 'correct'.

```

The regular expression pattern is as follows:

```
(?<duplicateWord>\w+)\s\k<duplicateWord>\W(?<nextWord>\w+)
```

The following table shows how the regular expression is interpreted.

PATTERN	DESCRIPTION
(?<duplicateWord>\w+)	Match one or more word characters. Name this capturing group <code>duplicateWord</code> .
\s	Match a white-space character.
\k<duplicateWord>	Match the string from the captured group that is named <code>duplicateWord</code> .
\W	Match a non-word character, including white space and punctuation. This prevents the regular expression pattern from matching a word that starts with the word from the first captured group.
(?<nextWord>\w+)	Match one or more word characters. Name this capturing group <code>nextWord</code> .

Note that a group name can be repeated in a regular expression. For example, it is possible for more than one group to be named `digit`, as the following example illustrates. In the case of duplicate names, the value of the `Group` object is determined by the last successful capture in the input string. In addition, the `CaptureCollection` is populated with information about each capture just as it would be if the group name was not duplicated.

In the following example, the regular expression `\D+(?<digit>\d+)\D+(?<digit>\d+)?` includes two occurrences of a group named `digit`. The first `digit` named group captures one or more digit characters. The second `digit` named group captures either zero or one occurrence of one or more digit characters. As the output from the example shows, if the second capturing group successfully matches text, the value of that text defines the value of the `Group` object. If the second capturing group cannot does not match the input string, the value of the last successful match defines the value of the `Group` object.

```
using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        String pattern = @"\D+(?<digit>\d+)\D+(?<digit>\d+)?";
        String[] inputs = { "abc123def456", "abc123def" };
        foreach (var input in inputs) {
            Match m = Regex.Match(input, pattern);
            if (m.Success) {
                Console.WriteLine("Match: {0}", m.Value);
                for (int grpCtr = 1; grpCtr < m.Groups.Count; grpCtr++) {
                    Group grp = m.Groups[grpCtr];
                    Console.WriteLine("Group {0}: {1}", grpCtr, grp.Value);
                    for (int capCtr = 0; capCtr < grp.Captures.Count; capCtr++)
                        Console.WriteLine("    Capture {0}: {1}", capCtr,
                                         grp.Captures[capCtr].Value);
                }
            }
            else {
                Console.WriteLine("The match failed.");
            }
            Console.WriteLine();
        }
    }
}

// The example displays the following output:
//      Match: abc123def456
//      Group 1: 456
//          Capture 0: 123
//          Capture 1: 456
//
//      Match: abc123def
//      Group 1: 123
//          Capture 0: 123
```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim pattern As String = "\D+(?<digit>\d+)\D+(?<digit>\d+)?"
        Dim inputs() As String = {"abc123def456", "abc123def"}
        For Each input As String In inputs
            Dim m As Match = Regex.Match(input, pattern)
            If m.Success Then
                Console.WriteLine("Match: {0}", m.Value)
                For grpCtr As Integer = 1 To m.Groups.Count - 1
                    Dim grp As Group = m.Groups(grpCtr)
                    Console.WriteLine("Group {0}: {1}", grpCtr, grp.Value)
                    For capCtr As Integer = 0 To grp.Captures.Count - 1
                        Console.WriteLine("    Capture {0}: {1}", capCtr,
                                         grp.Captures(capCtr).Value)
                    Next
                Next
            Else
                Console.WriteLine("The match failed.")
            End If
            Console.WriteLine()
        Next
    End Sub
End Module
' The example displays the following output:
'   Match: abc123def456
'   Group 1: 456
'       Capture 0: 123
'       Capture 1: 456
'
'   Match: abc123def
'   Group 1: 123
'       Capture 0: 123

```

The following table shows how the regular expression is interpreted.

PATTERN	DESCRIPTION
\D+	Match one or more non-decimal digit characters.
(?<digit>\d+)	Match one or more decimal digit characters. Assign the match to the <code>digit</code> named group.
\D+	Match one or more non-decimal digit characters.
(?<digit>\d+)?	Match zero or one occurrence of one or more decimal digit characters. Assign the match to the <code>digit</code> named group.

Balancing Group Definitions

A balancing group definition deletes the definition of a previously defined group and stores, in the current group, the interval between the previously defined group and the current group. This grouping construct has the following format:

`(?<name1-name2>subexpression)`

or:

`(?'name1-name2' subexpression)`

where *name1* is the current group (optional), *name2* is a previously defined group, and *subexpression* is any valid regular expression pattern. The balancing group definition deletes the definition of *name2* and stores the interval between *name2* and *name1* in *name1*. If no *name2* group is defined, the match backtracks. Because deleting the last definition of *name2* reveals the previous definition of *name2*, this construct lets you use the stack of captures for group *name2* as a counter for keeping track of nested constructs such as parentheses or opening and closing brackets.

The balancing group definition uses *name2* as a stack. The beginning character of each nested construct is placed in the group and in its [Group.Captures](#) collection. When the closing character is matched, its corresponding opening character is removed from the group, and the [Captures](#) collection is decreased by one. After the opening and closing characters of all nested constructs have been matched, *name2* is empty.

NOTE

After you modify the regular expression in the following example to use the appropriate opening and closing character of a nested construct, you can use it to handle most nested constructs, such as mathematical expressions or lines of program code that include multiple nested method calls.

The following example uses a balancing group definition to match left and right angle brackets (<>) in an input string. The example defines two named groups, `open` and `close`, that are used like a stack to track matching pairs of angle brackets. Each captured left angle bracket is pushed into the capture collection of the `open` group, and each captured right angle bracket is pushed into the capture collection of the `close` group. The balancing group definition ensures that there is a matching right angle bracket for each left angle bracket. If there is not, the final subpattern, `(?:Open)(?!)`, is evaluated only if the `open` group is not empty (and, therefore, if all nested constructs have not been closed). If the final subpattern is evaluated, the match fails, because the `(?!)` subpattern is a zero-width negative lookahead assertion that always fails.

```

using System;
using System.Text.RegularExpressions;

class Example
{
    public static void Main()
    {
        string pattern = "^[^>]*" +
            "(" +
            "((?'Open'<)[^>]*)+" +
            "((?'Close-Open'>)[^>]*)+" +
            ")"* +
            "(?(Open)(?!))$";
        string input = "<abc><mno<xyz>>";

        Match m = Regex.Match(input, pattern);
        if (m.Success == true)
        {
            Console.WriteLine("Input: \"{0}\" \nMatch: \"{1}\"", input, m);
            int grpCtr = 0;
            foreach (Group grp in m.Groups)
            {
                Console.WriteLine("    Group {0}: {1}", grpCtr, grp.Value);
                grpCtr++;
                int capCtr = 0;
                foreach (Capture cap in grp.Captures)
                {
                    Console.WriteLine("        Capture {0}: {1}", capCtr, cap.Value);
                    capCtr++;
                }
            }
        }
        else
        {
            Console.WriteLine("Match failed.");
        }
    }
}

// The example displays the following output:
//   Input: "<abc><mno<xyz>>"
//   Match: "<abc><mno<xyz>>"
//       Group 0: <abc><mno<xyz>>
//           Capture 0: <abc><mno<xyz>>
//       Group 1: <mno<xyz>>
//           Capture 0: <abc>
//           Capture 1: <mno<xyz>>
//       Group 2: <xyz
//           Capture 0: <abc
//           Capture 1: <mno
//           Capture 2: <xyz
//       Group 3: >
//           Capture 0: >
//           Capture 1: >
//           Capture 2: >
//       Group 4:
//       Group 5: mno<xyz>
//           Capture 0: abc
//           Capture 1: xyz
//           Capture 2: mno<xyz>

```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim pattern As String = "^[^<>]*" & _
            "(" + "(?'Open'<)[^<>]*)+" & _
            "((?'Close-Open'>)[^<>]*)+" + ")" * & _
            "(?(Open)(?!))$"
        Dim input As String = "<abc><mno<xyz>>"
        Dim rgx AS New Regex(pattern) '
        Dim m As Match = Regex.Match(input, pattern)
        If m.Success Then
            Console.WriteLine("Input: ""{0}"" " & vbCrLf & "Match: ""{1}"""", _
                input, m)
            Dim grpCtr As Integer = 0
            For Each grp As Group In m.Groups
                Console.WriteLine("    Group {0}: {1}", grpCtr, grp.Value)
                grpCtr += 1
                Dim capCtr As Integer = 0
                For Each cap As Capture In grp.Captures
                    Console.WriteLine("        Capture {0}: {1}", capCtr, cap.Value)
                    capCtr += 1
                Next
            Next
        Else
            Console.WriteLine("Match failed.")
        End If
    End Sub
End Module
' The example displays the following output:
'     Input: "<abc><mno<xyz>>"
'     Match: "<abc><mno<xyz>>"
'         Group 0: <abc><mno<xyz>>
'             Capture 0: <abc><mno<xyz>>
'         Group 1: <mno<xyz>>
'             Capture 0: <abc>
'             Capture 1: <mno
'             Capture 2: <xyz
'         Group 2: <xyz
'             Capture 0: <abc
'             Capture 1: <mno
'             Capture 2: <xyz
'         Group 3: >
'             Capture 0: >
'             Capture 1: >
'             Capture 2: >
'         Group 4:
'         Group 5: mno<xyz>
'             Capture 0: abc
'             Capture 1: xyz
'             Capture 2: mno<xyz>

```

The regular expression pattern is:

```
^[^<>]*(((?'Open'<)[^<>]*+)((?'Close-Open'>)[^<>]*+))*?(?(Open)(?!))$
```

The regular expression is interpreted as follows:

PATTERN	DESCRIPTION
^	Begin at the start of the string.
[^<>]*	Match zero or more characters that are not left or right angle brackets.

PATTERN	DESCRIPTION
(?'Open'<)	Match a left angle bracket and assign it to a group named <code>Open</code> .
[^<>]*	Match zero or more characters that are not left or right angle brackets.
((?'Open'<)[^<>]*)+	Match one or more occurrences of a left angle bracket followed by zero or more characters that are not left or right angle brackets. This is the second capturing group.
(?'Close-Open'>)	Match a right angle bracket, assign the substring between the <code>Open</code> group and the current group to the <code>Close</code> group, and delete the definition of the <code>Open</code> group.
[^<>]*	Match zero or more occurrences of any character that is neither a left nor a right angle bracket.
((?'Close-Open'>)[^<>]*)+	Match one or more occurrences of a right angle bracket, followed by zero or more occurrences of any character that is neither a left nor a right angle bracket. When matching the right angle bracket, assign the substring between the <code>Open</code> group and the current group to the <code>Close</code> group, and delete the definition of the <code>Open</code> group. This is the third capturing group.
((('Open'<)[^<>]*)+((?'Close-Open'>)[^<>]*)+)*	Match zero or more occurrences of the following pattern: one or more occurrences of a left angle bracket, followed by zero or more non-angle bracket characters, followed by one or more occurrences of a right angle bracket, followed by zero or more occurrences of non-angle brackets. When matching the right angle bracket, delete the definition of the <code>Open</code> group, and assign the substring between the <code>Open</code> group and the current group to the <code>Close</code> group. This is the first capturing group.
(?(Open)(?!))	If the <code>open</code> group exists, abandon the match if an empty string can be matched, but do not advance the position of the regular expression engine in the string. This is a zero-width negative lookahead assertion. Because an empty string is always implicitly present in an input string, this match always fails. Failure of this match indicates that the angle brackets are not balanced.
\$	Match the end of the input string.

The final subexpression, `(?(Open)(?!))`, indicates whether the nesting constructs in the input string are properly balanced (for example, whether each left angle bracket is matched by a right angle bracket). It uses conditional matching based on a valid captured group; for more information, see [Alternation Constructs](#). If the `open` group is defined, the regular expression engine attempts to match the subexpression `(?!)` in the input string. The `Open` group should be defined only if nesting constructs are unbalanced. Therefore, the pattern to be matched in the input string should be one that always causes the match to fail. In this case, `(?!)` is a zero-width negative lookahead assertion that always fails, because an empty string is always implicitly present at the next position in the input string.

In the example, the regular expression engine evaluates the input string "`<abc><mno<xyz>>`" as shown in the

following table.

STEP	PATTERN	RESULT
1	<code>^</code>	Starts the match at the beginning of the input string
2	<code>[^<>]*</code>	Looks for non-angle bracket characters before the left angle bracket; finds no matches.
3	<code>(((? 'Open' <)</code>	Matches the left angle bracket in "<abc>" and assigns it to the <code>open</code> group.
4	<code>[^<>]*</code>	Matches "abc".
5	<code>) +</code>	"<abc>" is the value of the second captured group. The next character in the input string is not a left angle bracket, so the regular expression engine does not loop back to the <code>(? 'Open' <) [^<>]*</code> subpattern.
6	<code>((? 'Close-Open' >)</code>	Matches the right angle bracket in "<abc>", assigns "abc", which is the substring between the <code>open</code> group and the right angle bracket, to the <code>close</code> group, and deletes the current value ("<") of the <code>open</code> group, leaving it empty.
7	<code>[^<>]*</code>	Looks for non-angle bracket characters after the right angle bracket; finds no matches.
8	<code>) +</code>	The value of the third captured group is ">". The next character in the input string is not a right angle bracket, so the regular expression engine does not loop back to the <code>((? 'Close-Open' >) [^<>]*</code> subpattern.
9	<code>) *</code>	The value of the first captured group is "<abc>". The next character in the input string is a left angle bracket, so the regular expression engine loops back to the <code>(((? 'Open' <)</code> subpattern.

STEP	PATTERN	RESULT
10	(((? 'Open' <)	Matches the left angle bracket in "<mno" and assigns it to the <code>open</code> group. Its <code>Group.Captures</code> collection now has a single value, "<".
11	[^<>]*	Matches "mno".
12)+	"<mno" is the value of the second captured group. The next character in the input string is an left angle bracket, so the regular expression engine loops back to the <code>(?'Open' <) [^<>]*</code> subpattern.
13	(((? 'Open' <)	Matches the left angle bracket in "<xyz>" and assigns it to the <code>open</code> group. The <code>Group.Captures</code> collection of the <code>open</code> group now includes two captures: the left angle bracket from "<mno", and the left angle bracket from "<xyz>".
14	[^<>]*	Matches "xyz".
15)+	"<xyz" is the value of the second captured group. The next character in the input string is not a left angle bracket, so the regular expression engine does not loop back to the <code>(?'Open' <) [^<>]*</code> subpattern.
16	((?'Close-Open' >)	Matches the right angle bracket in "<xyz>". "xyz", assigns the substring between the <code>open</code> group and the right angle bracket to the <code>close</code> group, and deletes the current value of the <code>open</code> group. The value of the previous capture (the left angle bracket in "<mno") becomes the current value of the <code>open</code> group. The <code>Captures</code> collection of the <code>open</code> group now includes a single capture, the left angle bracket from "<xyz>".
17	[^<>]*	Looks for non-angle bracket characters; finds no matches.

STEP	PATTERN	RESULT
18)+	The value of the third captured group is ">". The next character in the input string is a right angle bracket, so the regular expression engine loops back to the ((?'Close-Open'>)[^<>]*) subpattern.
19	((?'Close-Open'>)	Matches the final right angle bracket in "xyz>", assigns "mno<xyz>" (the substring between the Open group and the right angle bracket) to the Close group, and deletes the current value of the Open group. The Open group is now empty.
20	[^<>]*	Looks for non-angle bracket characters; finds no matches.
21)+	The value of the third captured group is ">". The next character in the input string is not a right angle bracket, so the regular expression engine does not loop back to the ((?'Close-Open'>)[^<>]*) subpattern.
22)*	The value of the first captured group is "<mno<xyz>>". The next character in the input string is not a left angle bracket, so the regular expression engine does not loop back to the (((?'Open'<) subpattern.
23	(?(Open)(?!))	The Open group is not defined, so no match is attempted.
24	\$	Matches the end of the input string.

Noncapturing Groups

The following grouping construct does not capture the substring that is matched by a subexpression:

(?:subexpression)

where *subexpression* is any valid regular expression pattern. The noncapturing group construct is typically used when a quantifier is applied to a group, but the substrings captured by the group are of no interest.

NOTE

If a regular expression includes nested grouping constructs, an outer noncapturing group construct does not apply to the inner nested group constructs.

The following example illustrates a regular expression that includes noncapturing groups. Note that the output does not include any captured groups.

```
using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string pattern = @"(?:\b(?:\w+)\W*)+\. ";
        string input = "This is a short sentence.";
        Match match = Regex.Match(input, pattern);
        Console.WriteLine("Match: {0}", match.Value);
        for (int ctr = 1; ctr < match.Groups.Count; ctr++)
            Console.WriteLine("    Group {0}: {1}", ctr, match.Groups[ctr].Value);
    }
}
// The example displays the following output:
//      Match: This is a short sentence.
```

```
Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim pattern As String = "(?:\b(?:\w+)\W*)+\. "
        Dim input As String = "This is a short sentence."
        Dim match As Match = Regex.Match(input, pattern)
        Console.WriteLine("Match: {0}", match.Value)
        For ctr As Integer = 1 To match.Groups.Count - 1
            Console.WriteLine("    Group {0}: {1}", ctr, match.Groups(ctr).Value)
        Next
    End Sub
End Module
' The example displays the following output:
'      Match: This is a short sentence.
```

The regular expression `(?:\b(?:\w+)\W*)+\.` matches a sentence that is terminated by a period. Because the regular expression focuses on sentences and not on individual words, grouping constructs are used exclusively as quantifiers. The regular expression pattern is interpreted as shown in the following table.

PATTERN	DESCRIPTION
<code>\b</code>	Begin the match at a word boundary.
<code>(?:\w+)</code>	Match one or more word characters. Do not assign the matched text to a captured group.
<code>\W*</code>	Match zero or more non-word characters.

PATTERN	DESCRIPTION
(?:\b(?:\w+)\W*)+	Match the pattern of one or more word characters starting at a word boundary, followed by zero or more non-word characters, one or more times. Do not assign the matched text to a captured group.
\.	Match a period.

Group Options

The following grouping construct applies or disables the specified options within a subexpression:

```
(?imnsx-imnsx: subexpression)
```

where *subexpression* is any valid regular expression pattern. For example, (?i-s:) turns on case insensitivity and disables single-line mode. For more information about the inline options you can specify, see [Regular Expression Options](#).

NOTE

You can specify options that apply to an entire regular expression rather than a subexpression by using a [System.Text.RegularExpressions.Regex](#) class constructor or a static method. You can also specify inline options that apply after a specific point in a regular expression by using the (?imnsx-imnsx) language construct.

The group options construct is not a capturing group. That is, although any portion of a string that is captured by *subexpression* is included in the match, it is not included in a captured group nor used to populate the [GroupCollection](#) object.

For example, the regular expression \b(?ix: d \w+)\s in the following example uses inline options in a grouping construct to enable case-insensitive matching and ignore pattern white space in identifying all words that begin with the letter "d". The regular expression is defined as shown in the following table.

PATTERN	DESCRIPTION
\b	Begin the match at a word boundary.
(?ix: d \w+)	Using case-insensitive matching and ignoring white space in this pattern, match a "d" followed by one or more word characters.
\s	Match a white-space character.

```
string pattern = @"\b(?ix: d \w+)\s";
string input = "Dogs are decidedly good pets.";

foreach (Match match in Regex.Matches(input, pattern))
    Console.WriteLine("{0}// found at index {1}.", match.Value, match.Index);
// The example displays the following output:
//      'Dogs // found at index 0.
//      'decidedly // found at index 9.
```

```

Dim pattern As String = "\b(?ix: d \w+)\s"
Dim input As String = "Dogs are decidedly good pets."

For Each match As Match In Regex.Matches(input, pattern)
    Console.WriteLine("{0}' found at index {1}.", match.Value, match.Index)
Next
' The example displays the following output:
'   'Dogs ' found at index 0.
'   'decidedly ' found at index 9.

```

Zero-Width Positive Lookahead Assertions

The following grouping construct defines a zero-width positive lookahead assertion:

`(?= subexpression)`

where *subexpression* is any regular expression pattern. For a match to be successful, the input string must match the regular expression pattern in *subexpression*, although the matched substring is not included in the match result. A zero-width positive lookahead assertion does not backtrack.

Typically, a zero-width positive lookahead assertion is found at the end of a regular expression pattern. It defines a substring that must be found at the end of a string for a match to occur but that should not be included in the match. It is also useful for preventing excessive backtracking. You can use a zero-width positive lookahead assertion to ensure that a particular captured group begins with text that matches a subset of the pattern defined for that captured group. For example, if a capturing group matches consecutive word characters, you can use a zero-width positive lookahead assertion to require that the first character be an alphabetical uppercase character.

The following example uses a zero-width positive lookahead assertion to match the word that precedes the verb "is" in the input string.

```

using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string pattern = @"\b\w+(?=\\sis\\b)";
        string[] inputs = { "The dog is a Malamute.",
                            "The island has beautiful birds.",
                            "The pitch missed home plate.",
                            "Sunday is a weekend day." };

        foreach (string input in inputs)
        {
            Match match = Regex.Match(input, pattern);
            if (match.Success)
                Console.WriteLine("{0}' precedes 'is'.", match.Value);
            else
                Console.WriteLine("{0}' does not match the pattern.", input);
        }
    }
}

// The example displays the following output:
//   'dog' precedes 'is'.
//   'The island has beautiful birds.' does not match the pattern.
//   'The pitch missed home plate.' does not match the pattern.
//   'Sunday' precedes 'is'.

```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim pattern As String = "\b\w+(?=\\sis\\b)"
        Dim inputs() As String = {"The dog is a Malamute.", _
                                "The island has beautiful birds.", _
                                "The pitch missed home plate.", _
                                "Sunday is a weekend day."}

        For Each input As String In inputs
            Dim match As Match = Regex.Match(input, pattern)
            If match.Success Then
                Console.WriteLine("'{0}' precedes 'is'.", match.Value)
            Else
                Console.WriteLine("'{0}' does not match the pattern.", input)
            End If
        Next
    End Sub
End Module
' The example displays the following output:
'     'dog' precedes 'is'.
'     'The island has beautiful birds.' does not match the pattern.
'     'The pitch missed home plate.' does not match the pattern.
'     'Sunday' precedes 'is'.

```

The regular expression `\b\w+(?=\\sis\\b)` is interpreted as shown in the following table.

PATTERN	DESCRIPTION
<code>\b</code>	Begin the match at a word boundary.
<code>\w+</code>	Match one or more word characters.
<code>(?=\\sis\\b)</code>	Determine whether the word characters are followed by a white-space character and the string "is", which ends on a word boundary. If so, the match is successful.

Zero-Width Negative Lookahead Assertions

The following grouping construct defines a zero-width negative lookahead assertion:

`(?! subexpression)`

where *subexpression* is any regular expression pattern. For the match to be successful, the input string must not match the regular expression pattern in *subexpression*, although the matched string is not included in the match result.

A zero-width negative lookahead assertion is typically used either at the beginning or at the end of a regular expression. At the beginning of a regular expression, it can define a specific pattern that should not be matched when the beginning of the regular expression defines a similar but more general pattern to be matched. In this case, it is often used to limit backtracking. At the end of a regular expression, it can define a subexpression that cannot occur at the end of a match.

The following example defines a regular expression that uses a zero-width lookahead assertion at the beginning of the regular expression to match words that do not begin with "un".

```

using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string pattern = @"\b(?!un)\w+\b";
        string input = "unite one unethical ethics use untie ultimate";
        foreach (Match match in Regex.Matches(input, pattern, RegexOptions.IgnoreCase))
            Console.WriteLine(match.Value);
    }
}
// The example displays the following output:
//      one
//      ethics
//      use
//      ultimate

```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim pattern As String = "\b(?!un)\w+\b"
        Dim input As String = "unite one unethical ethics use untie ultimate"
        For Each match As Match In Regex.Matches(input, pattern, RegexOptions.IgnoreCase)
            Console.WriteLine(match.Value)
        Next
    End Sub
End Module
' The example displays the following output:
'      one
'      ethics
'      use
'      ultimate

```

The regular expression `\b(?!un)\w+\b` is interpreted as shown in the following table.

PATTERN	DESCRIPTION
<code>\b</code>	Begin the match at a word boundary.
<code>(?!un)</code>	Determine whether the next two characters are "un". If they are not, a match is possible.
<code>\w+</code>	Match one or more word characters.
<code>\b</code>	End the match at a word boundary.

The following example defines a regular expression that uses a zero-width lookahead assertion at the end of the regular expression to match words that do not end with a punctuation character.

```

using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string pattern = @"\b\w+\b(?!p{P})";
        string input = "Disconnected, disjointed thoughts in a sentence fragment.";
        foreach (Match match in Regex.Matches(input, pattern))
            Console.WriteLine(match.Value);
    }
}
// The example displays the following output:
//      disjointed
//      thoughts
//      in
//      a
//      sentence

```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim pattern As String = "\b\w+\b(?!p{P})"
        Dim input As String = "Disconnected, disjointed thoughts in a sentence fragment."
        For Each match As Match In Regex.Matches(input, pattern)
            Console.WriteLine(match.Value)
        Next
    End Sub
End Module
' The example displays the following output:
'      disjointed
'      thoughts
'      in
'      a
'      sentence

```

The regular expression `\b\w+\b(?!p{P})` is interpreted as shown in the following table.

PATTERN	DESCRIPTION
<code>\b</code>	Begin the match at a word boundary.
<code>\w+</code>	Match one or more word characters.
<code>\b</code>	End the match at a word boundary.
<code>\p{P})</code>	If the next character is not a punctuation symbol (such as a period or a comma), the match succeeds.

Zero-Width Positive Lookbehind Assertions

The following grouping construct defines a zero-width positive lookbehind assertion:

`(?=< subexpression)`

where *subexpression* is any regular expression pattern. For a match to be successful, *subexpression* must occur at the input string to the left of the current position, although *subexpression* is not included in the match result.

A zero-width positive lookbehind assertion does not backtrack.

Zero-width positive lookbehind assertions are typically used at the beginning of regular expressions. The pattern that they define is a precondition for a match, although it is not a part of the match result.

For example, the following example matches the last two digits of the year for the twenty first century (that is, it requires that the digits "20" precede the matched string).

```
using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string input = "2010 1999 1861 2140 2009";
        string pattern = @"(?<=\b20)\d{2}\b";

        foreach (Match match in Regex.Matches(input, pattern))
            Console.WriteLine(match.Value);
    }
}
// The example displays the following output:
//      10
//      09
```

```
Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim input As String = "2010 1999 1861 2140 2009"
        Dim pattern As String = "(?<=\b20)\d{2}\b"

        For Each match As Match In Regex.Matches(input, pattern)
            Console.WriteLine(match.Value)
        Next
    End Sub
End Module
' The example displays the following output:
'      10
'      09
```

The regular expression pattern `(?<=\b20)\d{2}\b` is interpreted as shown in the following table.

PATTERN	DESCRIPTION
<code>\d{2}</code>	Match two decimal digits.
<code>(?<=\b20)</code>	Continue the match if the two decimal digits are preceded by the decimal digits "20" on a word boundary.
<code>\b</code>	End the match at a word boundary.

Zero-width positive lookbehind assertions are also used to limit backtracking when the last character or characters in a captured group must be a subset of the characters that match that group's regular expression pattern. For example, if a group captures all consecutive word characters, you can use a zero-width positive lookbehind assertion to require that the last character be alphabetical.

Zero-Width Negative Lookbehind Assertions

The following grouping construct defines a zero-width negative lookbehind assertion:

```
(?<! subexpression )
```

where *subexpression* is any regular expression pattern. For a match to be successful, *subexpression* must not occur at the input string to the left of the current position. However, any substring that does not match *subexpression* is not included in the match result.

Zero-width negative lookbehind assertions are typically used at the beginning of regular expressions. The pattern that they define precludes a match in the string that follows. They are also used to limit backtracking when the last character or characters in a captured group must not be one or more of the characters that match that group's regular expression pattern. For example, if a group captures all consecutive word characters, you can use a zero-width positive lookbehind assertion to require that the last character not be an underscore (_).

The following example matches the date for any day of the week that is not a weekend (that is, that is neither Saturday nor Sunday).

```
using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string[] dates = { "Monday February 1, 2010",
                           "Wednesday February 3, 2010",
                           "Saturday February 6, 2010",
                           "Sunday February 7, 2010",
                           "Monday, February 8, 2010" };
        string pattern = @"(?<!(Saturday|Sunday) )\b\w+ \d{1,2}, \d{4}\b";

        foreach (string dateValue in dates)
        {
            Match match = Regex.Match(dateValue, pattern);
            if (match.Success)
                Console.WriteLine(match.Value);
        }
    }
}

// The example displays the following output:
//      February 1, 2010
//      February 3, 2010
//      February 8, 2010
```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim dates() As String = {"Monday February 1, 2010", _
                               "Wednesday February 3, 2010", _
                               "Saturday February 6, 2010", _
                               "Sunday February 7, 2010", _
                               "Monday, February 8, 2010"}
        Dim pattern As String = "(?<!(Saturday|Sunday) )\b\w+ \d{1,2}, \d{4}\b"

        For Each dateValue As String In dates
            Dim match As Match = Regex.Match(dateValue, pattern)
            If match.Success Then
                Console.WriteLine(match.Value)
            End If
        Next
    End Sub
End Module
' The example displays the following output:
'     February 1, 2010
'     February 3, 2010
'     February 8, 2010

```

The regular expression pattern `(?<!(Saturday|Sunday))\b\w+ \d{1,2}, \d{4}\b` is interpreted as shown in the following table.

PATTERN	DESCRIPTION
<code>\b</code>	Begin the match at a word boundary.
<code>\w+</code>	Match one or more word characters followed by a white-space character.
<code>\d{1,2},</code>	Match either one or two decimal digits followed by a white-space character and a comma.
<code>\d{4}\b</code>	Match four decimal digits, and end the match at a word boundary.
<code>(?<!(Saturday Sunday))</code>	If the match is preceded by something other than the strings "Saturday" or "Sunday" followed by a space, the match is successful.

Atomic groups

The following grouping construct represents an atomic group (known in some other regular expression engines as a nonbacktracking subexpression, an atomic subexpression, or a once-only subexpression):

`(?> subexpression)`

where *subexpression* is any regular expression pattern.

Ordinarily, if a regular expression includes an optional or alternative matching pattern and a match does not succeed, the regular expression engine can branch in multiple directions to match an input string with a pattern. If a match is not found when it takes the first branch, the regular expression engine can back up or backtrack to the point where it took the first match and attempt the match using the second branch. This process can continue until all branches have been tried.

The `(?> subexpression)` language construct disables backtracking. The regular expression engine will match as many characters in the input string as it can. When no further match is possible, it will not backtrack to attempt alternate pattern matches. (That is, the subexpression matches only strings that would be matched by the subexpression alone; it does not attempt to match a string based on the subexpression and any subexpressions that follow it.)

This option is recommended if you know that backtracking will not succeed. Preventing the regular expression engine from performing unnecessary searching improves performance.

The following example illustrates how an atomic group modifies the results of a pattern match. The backtracking regular expression successfully matches a series of repeated characters followed by one more occurrence of the same character on a word boundary, but the nonbacktracking regular expression does not.

```
using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string[] inputs = { "cccd.", "aaad", "aaaa" };
        string back = @"(\w)\1+\.\b";
        string noback = @"(?>(\w)\1+)\.\b";

        foreach (string input in inputs)
        {
            Match match1 = Regex.Match(input, back);
            Match match2 = Regex.Match(input, noback);
            Console.WriteLine("{0}: ", input);

            Console.Write(" Backtracking : ");
            if (match1.Success)
                Console.WriteLine(match1.Value);
            else
                Console.WriteLine("No match");

            Console.Write(" Nonbacktracking: ");
            if (match2.Success)
                Console.WriteLine(match2.Value);
            else
                Console.WriteLine("No match");
        }
    }
}

// The example displays the following output:
//    cccd.:
//        Backtracking : cccd
//        Nonbacktracking: cccd
//    aaad:
//        Backtracking : aaad
//        Nonbacktracking: aaad
//    aaaa:
//        Backtracking : aaaa
//        Nonbacktracking: No match
```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim inputs() As String = {"cccd.", "aaad", "aaaa"}
        Dim back As String = "(\w)\1+\.\b"
        Dim noback As String = "(?>(\w)\1+)\.\b"

        For Each input As String In inputs
            Dim match1 As Match = Regex.Match(input, back)
            Dim match2 As Match = Regex.Match(input, noback)
            Console.WriteLine("{0}: ", input)

            Console.Write("    Backtracking : ")
            If match1.Success Then
                Console.WriteLine(match1.Value)
            Else
                Console.WriteLine("No match")
            End If

            Console.Write("    Nonbacktracking: ")
            If match2.Success Then
                Console.WriteLine(match2.Value)
            Else
                Console.WriteLine("No match")
            End If
        Next
    End Sub
End Module
' The example displays the following output:
' cccd.:
'     Backtracking : cccd
'     Nonbacktracking: cccd
' aaad:
'     Backtracking : aaad
'     Nonbacktracking: aaad
' aaaa:
'     Backtracking : aaaa
'     Nonbacktracking: No match

```

The nonbacktracking regular expression `(?>(\w)\1+)\.\b` is defined as shown in the following table.

PATTERN	DESCRIPTION
<code>(\w)</code>	Match a single word character and assign it to the first capturing group.
<code>\1+</code>	Match the value of the first captured substring one or more times.
<code>.</code>	Match any character.
<code>\b</code>	End the match on a word boundary.
<code>(?>(\w)\1+)\.\b</code>	Match one or more occurrences of a duplicated word character, but do not backtrack to match the last character on a word boundary.

Grouping Constructs and Regular Expression Objects

Substrings that are matched by a regular expression capturing group are represented by [System.Text.RegularExpressions.Group](#) objects, which can be retrieved from the [System.Text.RegularExpressions.GroupCollection](#) object that is returned by the [Match.Groups](#) property. The [GroupCollection](#) object is populated as follows:

- The first [Group](#) object in the collection (the object at index zero) represents the entire match.
- The next set of [Group](#) objects represent unnamed (numbered) capturing groups. They appear in the order in which they are defined in the regular expression, from left to right. The index values of these groups range from 1 to the number of unnamed capturing groups in the collection. (The index of a particular group is equivalent to its numbered backreference. For more information about backreferences, see [Backreference Constructs](#).)
- The final set of [Group](#) objects represent named capturing groups. They appear in the order in which they are defined in the regular expression, from left to right. The index value of the first named capturing group is one greater than the index of the last unnamed capturing group. If there are no unnamed capturing groups in the regular expression, the index value of the first named capturing group is one.

If you apply a quantifier to a capturing group, the corresponding [Group](#) object's [Capture.Value](#), [Capture.Index](#), and [Capture.Length](#) properties reflect the last substring that is captured by a capturing group. You can retrieve a complete set of substrings that are captured by groups that have quantifiers from the [CaptureCollection](#) object that is returned by the [Group.Captures](#) property.

The following example clarifies the relationship between the [Group](#) and [Capture](#) objects.

```
using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string pattern = @"(\b(\w+)\w+)+";
        string input = "This is a short sentence.";
        Match match = Regex.Match(input, pattern);
        Console.WriteLine("Match: '{0}'", match.Value);
        for (int ctr = 1; ctr < match.Groups.Count; ctr++)
        {
            Console.WriteLine("    Group {0}: '{1}'", ctr, match.Groups[ctr].Value);
            int capCtr = 0;
            foreach (Capture capture in match.Groups[ctr].Captures)
            {
                Console.WriteLine("        Capture {0}: '{1}'", capCtr, capture.Value);
                capCtr++;
            }
        }
    }
}

// The example displays the following output:
//      Match: 'This is a short sentence.'
//          Group 1: 'sentence.'
//              Capture 0: 'This '
//              Capture 1: 'is '
//              Capture 2: 'a '
//              Capture 3: 'short '
//              Capture 4: 'sentence.'
//          Group 2: 'sentence'
//              Capture 0: 'This'
//              Capture 1: 'is'
//              Capture 2: 'a'
//              Capture 3: 'short'
//              Capture 4: 'sentence'
```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim pattern As String = "(\b(\w+)\W+)+"
        Dim input As String = "This is a short sentence."
        Dim match As Match = Regex.Match(input, pattern)
        Console.WriteLine("Match: '{0}'", match.Value)
        For ctr As Integer = 1 To match.Groups.Count - 1
            Console.WriteLine("    Group {0}: '{1}'", ctr, match.Groups(ctr).Value)
            Dim capCtr As Integer = 0
            For Each capture As Capture In match.Groups(ctr).Captures
                Console.WriteLine("        Capture {0}: '{1}'", capCtr, capture.Value)
                capCtr += 1
            Next
        Next
    End Sub
End Module
' The example displays the following output:
'     Match: 'This is a short sentence.'
'         Group 1: 'sentence.'
'             Capture 0: 'This '
'             Capture 1: 'is '
'             Capture 2: 'a '
'             Capture 3: 'short '
'             Capture 4: 'sentence.'
'         Group 2: 'sentence'
'             Capture 0: 'This'
'             Capture 1: 'is'
'             Capture 2: 'a'
'             Capture 3: 'short'
'             Capture 4: 'sentence'

```

The regular expression pattern `(\b(\w+)\W+)+` extracts individual words from a string. It is defined as shown in the following table.

PATTERN	DESCRIPTION
<code>\b</code>	Begin the match at a word boundary.
<code>(\w+)</code>	Match one or more word characters. Together, these characters form a word. This is the second capturing group.
<code>\W+</code>	Match one or more non-word characters.
<code>(\b(\w+)\W+)</code>	Match the pattern of one or more word characters followed by one or more non-word characters one or more times. This is the first capturing group.

The second capturing group matches each word of the sentence. The first capturing group matches each word along with the punctuation and white space that follow the word. The [Group](#) object whose index is 2 provides information about the text matched by the second capturing group. The complete set of words captured by the capturing group are available from the [CaptureCollection](#) object returned by the [Group.Captures](#) property.

See also

- [Regular Expression Language - Quick Reference](#)
- [Backtracking](#)

Quantifiers in Regular Expressions

9/20/2022 • 23 minutes to read • [Edit Online](#)

Quantifiers specify how many instances of a character, group, or character class must be present in the input for a match to be found. The following table lists the quantifiers supported by .NET:

GREEDY QUANTIFIER	LAZY QUANTIFIER	DESCRIPTION
*	*?	Matches zero or more times.
+	+?	Matches one or more times.
?	??	Matches zero or one time.
{ n }	{ n }?	Matches exactly <i>n</i> times.
{ n , }	{ n , }?	Matches at least <i>n</i> times.
{ n , m }	{ n , m }?	Matches from <i>n</i> to <i>m</i> times.

The quantities `n` and `m` are integer constants. Ordinarily, quantifiers are greedy. They cause the regular expression engine to match as many occurrences of particular patterns as possible. Appending the `?` character to a quantifier makes it lazy. It causes the regular expression engine to match as few occurrences as possible. For a complete description of the difference between greedy and lazy quantifiers, see the section [Greedy and Lazy Quantifiers](#) later in this article.

IMPORTANT

Nesting quantifiers, such as the regular expression pattern `(a*)*`, can increase the number of comparisons that the regular expression engine must perform. The number of comparisons can increase as an exponential function of the number of characters in the input string. For more information about this behavior and its workarounds, see [Backtracking](#).

Regular Expression Quantifiers

The following sections list the quantifiers supported by .NET regular expressions:

NOTE

If the `*`, `+`, `?`, `{`, and `}` characters are encountered in a regular expression pattern, the regular expression engine interprets them as quantifiers or part of quantifier constructs unless they are included in a [character class](#). To interpret these as literal characters outside a character class, you must escape them by preceding them with a backslash. For example, the string `*` in a regular expression pattern is interpreted as a literal asterisk ("`*`") character.

Match Zero or More Times: *

The `*` quantifier matches the preceding element zero or more times. It's equivalent to the `{0,}` quantifier. `*` is a greedy quantifier whose lazy equivalent is `*?`.

The following example illustrates this regular expression. Five of the nine digit-groups in the input string match

the pattern and four (`95`, `929`, `9219`, and `9919`) don't.

```
string pattern = @"\b91*9*\b";
string input = "99 95 919 929 9119 9219 999 9919 91119";
foreach (Match match in Regex.Matches(input, pattern))
    Console.WriteLine("{0} found at position {1}.", match.Value, match.Index);

// The example displays the following output:
//      '99' found at position 0.
//      '919' found at position 6.
//      '9119' found at position 14.
//      '999' found at position 24.
//      '91119' found at position 33.
```

```
Dim pattern As String = "\b91*9*\b"
Dim input As String = "99 95 919 929 9119 9219 999 9919 91119"
For Each match As Match In Regex.Matches(input, pattern)
    Console.WriteLine("{0} found at position {1}.", match.Value, match.Index)
Next
' The example displays the following output:
'      '99' found at position 0.
'      '919' found at position 6.
'      '9119' found at position 14.
'      '999' found at position 24.
'      '91119' found at position 33.
```

The regular expression pattern is defined as shown in the following table:

PATTERN	DESCRIPTION
<code>\b</code>	Specifies that the match must start at a word boundary.
<code>91*</code>	Matches a <code>9</code> followed by zero or more <code>1</code> characters.
<code>9*</code>	Matches zero or more <code>9</code> characters.
<code>\b</code>	Specifies that the match must end at a word boundary.

Match One or More Times: +

The `+` quantifier matches the preceding element one or more times. It's equivalent to `{1,}`. `+` is a greedy quantifier whose lazy equivalent is `+?`.

For example, the regular expression `\ban+\w*\b` tries to match entire words that begin with the letter `a` followed by one or more instances of the letter `n`. The following example illustrates this regular expression. The regular expression matches the words `an`, `annual`, `announcement`, and `antique`, and correctly fails to match `autumn` and `all`.

```

string pattern = @"\ban+\w*\b";

string input = "Autumn is a great time for an annual announcement to all antique collectors.";
foreach (Match match in Regex.Matches(input, pattern, RegexOptions.IgnoreCase))
    Console.WriteLine("{0} found at position {1}.", match.Value, match.Index);

// The example displays the following output:
//      'an' found at position 27.
//      'annual' found at position 30.
//      'announcement' found at position 37.
//      'antique' found at position 57.

```

```

Dim pattern As String = "\ban+\w*\b"

Dim input As String = "Autumn is a great time for an annual announcement to all antique collectors."
For Each match As Match In Regex.Matches(input, pattern, RegexOptions.IgnoreCase)
    Console.WriteLine("{0} found at position {1}.", match.Value, match.Index)
Next
'The example displays the following output:
'      'an' found at position 27.
'      'annual' found at position 30.
'      'announcement' found at position 37.
'      'antique' found at position 57.

```

The regular expression pattern is defined as shown in the following table:

PATTERN	DESCRIPTION
\b	Start at a word boundary.
an+	Matches an a followed by one or more n characters.
\w*?	Matches a word character zero or more times but as few times as possible.
\b	End at a word boundary.

Match Zero or One Time: ?

The **?** quantifier matches the preceding element zero or one time. It's equivalent to **{0,1}**. **?** is a greedy quantifier whose lazy equivalent is **??**.

For example, the regular expression **\ban?\b** tries to match entire words that begin with the letter **a** followed by zero or one instance of the letter **n**. In other words, it tries to match the words **a** and **an**. The following example illustrates this regular expression:

```

string pattern = @"\ban?\b";
string input = "An amiable animal with a large snout and an animated nose.";
foreach (Match match in Regex.Matches(input, pattern, RegexOptions.IgnoreCase))
    Console.WriteLine("{0} found at position {1}.", match.Value, match.Index);

// The example displays the following output:
//      'An' found at position 0.
//      'a' found at position 23.
//      'an' found at position 42.

```

```

Dim pattern As String = "\ban?\b"
Dim input As String = "An amiable animal with a large snout and an animated nose."
For Each match As Match In Regex.Matches(input, pattern, RegexOptions.IgnoreCase)
    Console.WriteLine("{0}' found at position {1}.", match.Value, match.Index)
Next
'The example displays the following output:
'      'An' found at position 0.
'      'a' found at position 23.
'      'an' found at position 42.

```

The regular expression pattern is defined as shown in the following table:

PATTERN	DESCRIPTION
\b	Start at a word boundary.
an?	Matches an a followed by zero or one n character.
\b	End at a word boundary.

Match Exactly n Times: {n}

The {n} quantifier matches the preceding element exactly *n* times, where *n* is any integer. {n} is a greedy quantifier whose lazy equivalent is {n}?.

For example, the regular expression \b\d+\,\d{3}\b tries to match a word boundary followed by one or more decimal digits followed by three decimal digits followed by a word boundary. The following example illustrates this regular expression:

```

string pattern = @"\b\d+\,\d{3}\b";
string input = "Sales totaled 103,524 million in January, " +
              "106,971 million in February, but only " +
              "943 million in March.";
foreach (Match match in Regex.Matches(input, pattern))
    Console.WriteLine("{0}' found at position {1}.", match.Value, match.Index);

// The example displays the following output:
//      '103,524' found at position 14.
//      '106,971' found at position 45.

```

```

Dim pattern As String = "\b\d+\,\d{3}\b"
Dim input As String = "Sales totaled 103,524 million in January, " + _
                      "106,971 million in February, but only " + _
                      "943 million in March."
For Each match As Match In Regex.Matches(input, pattern)
    Console.WriteLine("{0}' found at position {1}.", match.Value, match.Index)
Next
'The example displays the following output:
'      '103,524' found at position 14.
'      '106,971' found at position 45.

```

The regular expression pattern is defined as shown in the following table:

PATTERN	DESCRIPTION
\b	Start at a word boundary.

PATTERN	DESCRIPTION
\d+	Matches one or more decimal digits.
\,	Matches a comma character.
\d{3}	Matches three decimal digits.
\b	End at a word boundary.

Match at Least n Times: {n,}

The `{n,}` quantifier matches the preceding element at least n times, where n is any integer. `{n,}` is a greedy quantifier whose lazy equivalent is `{n,}?`.

For example, the regular expression `\b\d{2,}\b\D+` tries to match a word boundary followed by at least two digits followed by a word boundary and a non-digit character. The following example illustrates this regular expression. The regular expression fails to match the phrase "7 days" because it contains just one decimal digit, but it successfully matches the phrases "10 weeks" and "300 years".

```
string pattern = @"\b\d{2,}\b\D+";
string input = "7 days, 10 weeks, 300 years";
foreach (Match match in Regex.Matches(input, pattern))
    Console.WriteLine("{0}' found at position {1}.", match.Value, match.Index);

// The example displays the following output:
//      '10 weeks, ' found at position 8.
//      '300 years' found at position 18.
```

```
Dim pattern As String = "\b\d{2,}\b\D+"
Dim input As String = "7 days, 10 weeks, 300 years"
For Each match As Match In Regex.Matches(input, pattern)
    Console.WriteLine("{0}' found at position {1}.", match.Value, match.Index)
Next
' The example displays the following output:
'      '10 weeks, ' found at position 8.
'      '300 years' found at position 18.
```

The regular expression pattern is defined as shown in the following table:

PATTERN	DESCRIPTION
\b	Start at a word boundary.
\d{2,}	Matches at least two decimal digits.
\b	Matches a word boundary.
\D+	Matches at least one non-decimal digit.

Match Between n and m Times: {n,m}

The `{n,m}` quantifier matches the preceding element at least n times, but no more than m times, where n and m are integers. `{n,m}` is a greedy quantifier whose lazy equivalent is `{n,m}?`.

In the following example, the regular expression `(00\s){2,4}` tries to match between two and four occurrences

of two zero digits followed by a space. The final portion of the input string includes this pattern five times rather than the maximum of four. However, only the initial portion of this substring (up to the space and the fifth pair of zeros) matches the regular expression pattern.

```
string pattern = @"(00\s){2,4}";
string input = "0x00 FF 00 00 18 17 FF 00 00 00 21 00 00 00 00 00";
foreach (Match match in Regex.Matches(input, pattern))
    Console.WriteLine("{0}' found at position {1}.", match.Value, match.Index);

// The example displays the following output:
//      '00 00 ' found at position 8.
//      '00 00 00 ' found at position 23.
//      '00 00 00 00 ' found at position 35.
```

```
Dim pattern As String = "(00\s){2,4}"
Dim input As String = "0x00 FF 00 00 18 17 FF 00 00 00 21 00 00 00 00 00"
For Each match As Match In Regex.Matches(input, pattern)
    Console.WriteLine("{0}' found at position {1}.", match.Value, match.Index)
Next
'The example displays the following output:
'      '00 00 ' found at position 8.
'      '00 00 00 ' found at position 23.
'      '00 00 00 00 ' found at position 35.
```

Match Zero or More Times (Lazy Match): *?

The `*?` quantifier matches the preceding element zero or more times but as few times as possible. It's the lazy counterpart of the greedy quantifier `*`.

In the following example, the regular expression `\b\w*?oo\w*?\b` matches all words that contain the string `oo`.

```
string pattern = @"\b\w*?oo\w*?\b";
string input = "woof root root rob oof woo woe";
foreach (Match match in Regex.Matches(input, pattern, RegexOptions.IgnoreCase))
    Console.WriteLine("{0}' found at position {1}.", match.Value, match.Index);

// The example displays the following output:
//      'woof' found at position 0.
//      'root' found at position 5.
//      'root' found at position 10.
//      'oof' found at position 19.
//      'woo' found at position 23.
```

```
Dim pattern As String = "\b\w*?oo\w*?\b"
Dim input As String = "woof root root rob oof woo woe"
For Each match As Match In Regex.Matches(input, pattern, RegexOptions.IgnoreCase)
    Console.WriteLine("{0}' found at position {1}.", match.Value, match.Index)
Next
'The example displays the following output:
'      'woof' found at position 0.
'      'root' found at position 5.
'      'root' found at position 10.
'      'oof' found at position 19.
'      'woo' found at position 23.
```

The regular expression pattern is defined as shown in the following table:

PATTERN	DESCRIPTION
\b	Start at a word boundary.
\w*?	Matches zero or more word characters but as few characters as possible.
oo	Matches the string oo.
\w*?	Matches zero or more word characters but as few characters as possible.
\b	End on a word boundary.

Match One or More Times (Lazy Match): +?

The +? quantifier matches the preceding element one or more times but as few times as possible. It's the lazy counterpart of the greedy quantifier +.

For example, the regular expression \b\w+?\b matches one or more characters separated by word boundaries. The following example illustrates this regular expression:

```
string pattern = @"\b\w+?\b";
string input = "Aa Bb Cc Dd Ee Ff";
foreach (Match match in Regex.Matches(input, pattern))
    Console.WriteLine("{0} found at position {1}.", match.Value, match.Index);

// The example displays the following output:
//      'Aa' found at position 0.
//      'Bb' found at position 3.
//      'Cc' found at position 6.
//      'Dd' found at position 9.
//      'Ee' found at position 12.
//      'Ff' found at position 15.
```

```
Dim pattern As String = "\b\w+?\b"
Dim input As String = "Aa Bb Cc Dd Ee Ff"
For Each match As Match In Regex.Matches(input, pattern)
    Console.WriteLine("{0} found at position {1}.", match.Value, match.Index)
Next
' The example displays the following output:
'      'Aa' found at position 0.
'      'Bb' found at position 3.
'      'Cc' found at position 6.
'      'Dd' found at position 9.
'      'Ee' found at position 12.
'      'Ff' found at position 15.
```

Match Zero or One Time (Lazy Match): ??

The ?? quantifier matches the preceding element zero or one time but as few times as possible. It's the lazy counterpart of the greedy quantifier ?.

For example, the regular expression ^\s*(System.)??Console.WriteLine()??\(? attempts to match the strings Console.WriteLine or Console.WriteLine. The string can also include System. before Console, and it can be followed by an opening parenthesis. The string must be at the beginning of a line, although it can be preceded by white space. The following example illustrates this regular expression:

```

string pattern = @"^s*(System.)??Console.WriteLine()?(?";
string input = "System.Console.WriteLine(\"Hello!\")\n" +
    "Console.Write(\"Hello!\")\n" +
    "Console.WriteLine(\"Hello!\")\n" +
    "Console.ReadLine()\n" +
    "    Console.WriteLine";
foreach (Match match in Regex.Matches(input, pattern,
    RegexOptions.IgnorePatternWhitespace |
    RegexOptions.IgnoreCase |
    RegexOptions.Multiline))
    Console.WriteLine("{0}' found at position {1}.", match.Value, match.Index);

// The example displays the following output:
//     'System.Console.WriteLine' found at position 0.
//     'Console.Write' found at position 36.
//     'Console.WriteLine' found at position 61.
//         'Console.ReadLine' found at position 110.

```

```

Dim pattern As String = "^\s*(System.)??Console.WriteLine()?(?"
Dim input As String = "System.Console.WriteLine(""Hello!""") + vbCrLf + _
    "Console.WriteLine("")" + vbCrLf + _
    "Console.WriteLine("")" + vbCrLf + _
    "Console.ReadLine()" + vbCrLf + _
    "    Console.WriteLine"
For Each match As Match In Regex.Matches(input, pattern, _
    RegexOptions.IgnorePatternWhitespace Or RegexOptions.IgnoreCase Or
RegexOptions.MultiLine)
    Console.WriteLine("{0}' found at position {1}.", match.Value, match.Index)
Next
' The example displays the following output:
'     'System.Console.WriteLine' found at position 0.
'     'Console.WriteLine' found at position 36.
'     'Console.ReadLine' found at position 61.
'         'Console.ReadLine' found at position 110.

```

The regular expression pattern is defined as shown in the following table:

PATTERN	DESCRIPTION
^	Matches the start of the input stream.
\s*	Matches zero or more white-space characters.
(System.)??	Matches zero or one occurrence of the string System..
Console.WriteLine	Matches the string Console.WriteLine.
(Line)??	Matches zero or one occurrence of the string Line..
\(??	Matches zero or one occurrence of the opening parenthesis.

Match Exactly n Times (Lazy Match): {n}?

The {n}? quantifier matches the preceding element exactly n times, where n is any integer. It's the lazy counterpart of the greedy quantifier {n}.

In the following example, the regular expression \b(\w{3,}?\.\){2}?\w{3,}?\b is used to identify a website address. The expression matches www.microsoft.com and msdn.microsoft.com but doesn't match mywebsite or mycompany.com.

```

string pattern = @"\b(\w{3,}?\.){2}\w{3,}\b";
string input = "www.microsoft.com msdn.microsoft.com mywebsite mycompany.com";
foreach (Match match in Regex.Matches(input, pattern))
    Console.WriteLine("{0} found at position {1}.", match.Value, match.Index);

// The example displays the following output:
//      'www.microsoft.com' found at position 0.
//      'msdn.microsoft.com' found at position 18.

```

```

Dim pattern As String = "\b(\w{3,}?\.){2}\w{3,}\b"
Dim input As String = "www.microsoft.com msdn.microsoft.com mywebsite mycompany.com"
For Each match As Match In Regex.Matches(input, pattern)
    Console.WriteLine("{0} found at position {1}.", match.Value, match.Index)
Next
' The example displays the following output:
'      'www.microsoft.com' found at position 0.
'      'msdn.microsoft.com' found at position 18.

```

The regular expression pattern is defined as shown in the following table:

PATTERN	DESCRIPTION
\b	Start at a word boundary.
(\w{3,}?\.)	Matches at least three word-characters but as few characters as possible, followed by a dot or period character. This pattern is the first capturing group.
(\w{3,}?\.){2}?	Matches the pattern in the first group two times but as few times as possible.
\b	End the match on a word boundary.

Match at Least n Times (Lazy Match): {n,}?

The `{ n , }?` quantifier matches the preceding element at least `n` times, where `n` is any integer but as few times as possible. It's the lazy counterpart of the greedy quantifier `{ n , }`.

See the example for the `{ n }?` quantifier in the previous section for an illustration. The regular expression in that example uses the `{ n , }` quantifier to match a string that has at least three characters followed by a period.

Match Between n and m Times (Lazy Match): {n,m}?

The `{ n , m }?` quantifier matches the preceding element between `n` and `m` times, where `n` and `m` are integers but as few times as possible. It's the lazy counterpart of the greedy quantifier `{ n , m }`.

In the following example, the regular expression `\b[A-Z](\w*?\s*){1,10}[.!?]` matches sentences that contain between 1 and 10 words. It matches all the sentences in the input string except for one sentence that contains 18 words.

```

string pattern = @"\b[A-Z](\w*\s*){1,10}[.!?]";
string input = "Hi. I am writing a short note. Its purpose is " +
    "to test a regular expression that attempts to find " +
    "sentences with ten or fewer words. Most sentences " +
    "in this note are short.";
foreach (Match match in Regex.Matches(input, pattern))
    Console.WriteLine("{0} found at position {1}.", match.Value, match.Index);

// The example displays the following output:
//      'Hi.' found at position 0.
//      'I am writing a short note.' found at position 4.
//      'Most sentences in this note are short.' found at position 132.

```

```

Dim pattern As String = "\b[A-Z](\w*\s*){1,10}[.!?]"
Dim input As String = "Hi. I am writing a short note. Its purpose is " + _
    "to test a regular expression that attempts to find " + _
    "sentences with ten or fewer words. Most sentences " + _
    "in this note are short."
For Each match As Match In Regex.Matches(input, pattern)
    Console.WriteLine("{0} found at position {1}.", match.Value, match.Index)
Next
' The example displays the following output:
'      'Hi.' found at position 0.
'      'I am writing a short note.' found at position 4.
'      'Most sentences in this note are short.' found at position 132.

```

The regular expression pattern is defined as shown in the following table:

PATTERN	DESCRIPTION
\b	Start at a word boundary.
[A-Z]	Matches an uppercase character from A to Z.
(\w*?\s*)	Matches zero or more word characters, followed by one or more white-space characters but as few times as possible. This pattern is the first capturing group.
{1,10}	Matches the previous pattern between 1 and 10 times.
[.!?]	Matches any one of the punctuation characters ., !, or ?.

Greedy and Lazy Quantifiers

Some quantifiers have two versions:

- A greedy version.

A greedy quantifier tries to match an element as many times as possible.

- A non-greedy (or lazy) version.

A non-greedy quantifier tries to match an element as few times as possible. You can turn a greedy quantifier into a lazy quantifier by adding a ?.

Consider a regular expression that's intended to extract the last four digits from a string of numbers, such as a

credit card number. The version of the regular expression that uses the `*` greedy quantifier is `\b.*([0-9]{4})\b`. However, if a string contains two numbers, this regular expression matches the last four digits of the second number only, as the following example shows:

```
string greedyPattern = @"\b.*([0-9]{4})\b";
string input1 = "1112223333 3992991999";
foreach (Match match in Regex.Matches(input1, greedyPattern))
    Console.WriteLine("Account ending in *****{0}.", match.Groups[1].Value);

// The example displays the following output:
//      Account ending in *****1999.
```

```
Dim greedyPattern As String = "\b.*([0-9]{4})\b"
Dim input1 As String = "1112223333 3992991999"
For Each match As Match In Regex.Matches(input1, greedypattern)
    Console.WriteLine("Account ending in *****{0}.", match.Groups(1).Value)
Next
' The example displays the following output:
'      Account ending in *****1999.
```

The regular expression fails to match the first number because the `*` quantifier tries to match the previous element as many times as possible in the entire string, and so it finds its match at the end of the string.

This behavior isn't the desired one. Instead, you can use the `*?` lazy quantifier to extract digits from both numbers, as the following example shows:

```
string lazyPattern = @"\b.*?([0-9]{4})\b";
string input2 = "1112223333 3992991999";
foreach (Match match in Regex.Matches(input2, lazyPattern))
    Console.WriteLine("Account ending in *****{0}.", match.Groups[1].Value);

// The example displays the following output:
//      Account ending in *****3333.
//      Account ending in *****1999.
```

```
Dim lazyPattern As String = "\b.*?([0-9]{4})\b"
Dim input2 As String = "1112223333 3992991999"
For Each match As Match In Regex.Matches(input2, lazypattern)
    Console.WriteLine("Account ending in *****{0}.", match.Groups(1).Value)
Next
' The example displays the following output:
'      Account ending in *****3333.
'      Account ending in *****1999.
```

In most cases, regular expressions with greedy and lazy quantifiers return the same matches. They most commonly return different results when they're used with the wildcard (`.`) metacharacter, which matches any character.

Quantifiers and Empty Matches

The quantifiers `*`, `+`, and `{n, m}` and their lazy counterparts never repeat after an empty match when the minimum number of captures has been found. This rule prevents quantifiers from entering infinite loops on empty subexpression matches when the maximum number of possible group captures is infinite or near infinite.

For example, the following code shows the result of a call to the `Regex.Match` method with the regular expression pattern `(a?)*`, which matches zero or one `a` character zero or more times. The single capturing

group captures each `a` and `String.Empty`, but there's no second empty match because the first empty match causes the quantifier to stop repeating.

```
using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string pattern = "(a?)*";
        string input = "aaabbb";
        Match match = Regex.Match(input, pattern);
        Console.WriteLine("Match: '{0}' at index {1}",
                          match.Value, match.Index);
        if (match.Groups.Count > 1) {
            GroupCollection groups = match.Groups;
            for (int grpCtr = 1; grpCtr <= groups.Count - 1; grpCtr++) {
                Console.WriteLine("    Group {0}: '{1}' at index {2}",
                                  grpCtr,
                                  groups[grpCtr].Value,
                                  groups[grpCtr].Index);
                int captureCtr = 0;
                foreach (Capture capture in groups[grpCtr].Captures) {
                    captureCtr++;
                    Console.WriteLine("        Capture {0}: '{1}' at index {2}",
                                      captureCtr, capture.Value, capture.Index);
                }
            }
        }
    }
}
// The example displays the following output:
//      Match: 'aaa' at index 0
//      Group 1: '' at index 3
//          Capture 1: 'a' at index 0
//          Capture 2: 'a' at index 1
//          Capture 3: 'a' at index 2
//          Capture 4: '' at index 3
```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim pattern As String = "(a?)"
        Dim input As String = "aaabbb"
        Dim match As Match = Regex.Match(input, pattern)
        Console.WriteLine("Match: '{0}' at index {1}",
                           match.Value, match.Index)
        If match.Groups.Count > 1 Then
            Dim groups As GroupCollection = match.Groups
            For grpCtr As Integer = 1 To groups.Count - 1
                Console.WriteLine("  Group {0}: '{1}' at index {2}",
                                  grpCtr,
                                  groups(grpCtr).Value,
                                  groups(grpCtr).Index)
            Dim captureCtr As Integer = 0
            For Each capture As Capture In groups(grpCtr).Captures
                captureCtr += 1
                Console.WriteLine("    Capture {0}: '{1}' at index {2}",
                                 captureCtr, capture.Value, capture.Index)
            Next
        Next
    End If
End Sub
End Module
' The example displays the following output:
'   Match: 'aaa' at index 0
'     Group 1: '' at index 3
'       Capture 1: 'a' at index 0
'       Capture 2: 'a' at index 1
'       Capture 3: 'a' at index 2
'       Capture 4: '' at index 3

```

To see the practical difference between a capturing group that defines a minimum and a maximum number of captures and one that defines a fixed number of captures, consider the regular expression patterns

`(a\1|(?(1)\1)){0,2}` and `(a\1|(?(1)\1)){2}`. Both regular expressions consist of a single capturing group, which is defined in the following table:

PATTERN	DESCRIPTION
<code>(a\1 (?(1)\1)){0,2}</code>	Either matches <code>a</code> along with the value of the first captured group ...
<code> (?(1)\1){2}</code>	... or tests whether the first captured group has been defined. The <code>(?(1)</code> construct doesn't define a capturing group.
<code>\1))</code>	If the first captured group exists, match its value. If the group doesn't exist, the group will match <code>String.Empty</code> .

The first regular expression tries to match this pattern between zero and two times; the second, exactly two times. Because the first pattern reaches its minimum number of captures with its first capture of `String.Empty`, it never repeats to try to match `a\1`. The `{0,2}` quantifier allows only empty matches in the last iteration. In contrast, the second regular expression does match `a` because it evaluates `a\1` a second time. The minimum number of iterations, 2, forces the engine to repeat after an empty match.

```

using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string pattern, input;

        pattern = @"(a\1|(?(1)\1)){0,2}";
        input = "aaabbb";

        Console.WriteLine("Regex pattern: {0}", pattern);
        Match match = Regex.Match(input, pattern);
        Console.WriteLine("Match: '{0}' at position {1}.",
                          match.Value, match.Index);
        if (match.Groups.Count > 1) {
            for (int groupCtr = 1; groupCtr <= match.Groups.Count - 1; groupCtr++)
            {
                Group group = match.Groups[groupCtr];
                Console.WriteLine("    Group: {0}: '{1}' at position {2}.",
                                  groupCtr, group.Value, group.Index);
                int captureCtr = 0;
                foreach (Capture capture in group.Captures) {
                    captureCtr++;
                    Console.WriteLine("        Capture: {0}: '{1}' at position {2}.",
                                      captureCtr, capture.Value, capture.Index);
                }
            }
        }
        Console.WriteLine();

        pattern = @"(a\1|(?(1)\1)){2}";
        Console.WriteLine("Regex pattern: {0}", pattern);
        match = Regex.Match(input, pattern);
        Console.WriteLine("Matched '{0}' at position {1}.",
                          match.Value, match.Index);
        if (match.Groups.Count > 1) {
            for (int groupCtr = 1; groupCtr <= match.Groups.Count - 1; groupCtr++)
            {
                Group group = match.Groups[groupCtr];
                Console.WriteLine("    Group: {0}: '{1}' at position {2}.",
                                  groupCtr, group.Value, group.Index);
                int captureCtr = 0;
                foreach (Capture capture in group.Captures) {
                    captureCtr++;
                    Console.WriteLine("        Capture: {0}: '{1}' at position {2}.",
                                      captureCtr, capture.Value, capture.Index);
                }
            }
        }
    }
}

// The example displays the following output:
//     Regex pattern: (a\1|(?(1)\1)){0,2}
//     Match: '' at position 0.
//         Group: 1: '' at position 0.
//             Capture: 1: '' at position 0.
//
//     Regex pattern: (a\1|(?(1)\1)){2}
//     Matched 'a' at position 0.
//         Group: 1: 'a' at position 0.
//             Capture: 1: '' at position 0.
//             Capture: 2: 'a' at position 0.

```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim pattern, input As String

        pattern = "(a\1|(?(1)\1)){0,2}"
        input = "aaabbb"

        Console.WriteLine("Regex pattern: {0}", pattern)
        Dim match As Match = Regex.Match(input, pattern)
        Console.WriteLine("Match: '{0}' at position {1}.",
                          match.Value, match.Index)
        If match.Groups.Count > 1 Then
            For groupCtr As Integer = 1 To match.Groups.Count - 1
                Dim group As Group = match.Groups(groupCtr)
                Console.WriteLine("    Group: {0}: '{1}' at position {2}.",
                                  groupCtr, group.Value, group.Index)
                Dim captureCtr As Integer = 0
                For Each capture As Capture In group.Captures
                    captureCtr += 1
                    Console.WriteLine("        Capture: {0}: '{1}' at position {2}.",
                                      captureCtr, capture.Value, capture.Index)
                Next
            Next
        End If
        Console.WriteLine()

        pattern = "(a\1|(?(1)\1)){2}"
        Console.WriteLine("Regex pattern: {0}", pattern)
        match = Regex.Match(input, pattern)
        Console.WriteLine("Matched '{0}' at position {1}.",
                          match.Value, match.Index)
        If match.Groups.Count > 1 Then
            For groupCtr As Integer = 1 To match.Groups.Count - 1
                Dim group As Group = match.Groups(groupCtr)
                Console.WriteLine("    Group: {0}: '{1}' at position {2}.",
                                  groupCtr, group.Value, group.Index)
                Dim captureCtr As Integer = 0
                For Each capture As Capture In group.Captures
                    captureCtr += 1
                    Console.WriteLine("        Capture: {0}: '{1}' at position {2}.",
                                      captureCtr, capture.Value, capture.Index)
                Next
            Next
        End If
    End Sub
End Module
' The example displays the following output:
'   Regex pattern: (a\1|(?(1)\1)){0,2}
'   Match: '' at position 0.
'       Group: 1: '' at position 0.
'           Capture: 1: '' at position 0.

'   Regex pattern: (a\1|(?(1)\1)){2}
'   Matched 'a' at position 0.
'       Group: 1: 'a' at position 0.
'           Capture: 1: '' at position 0.
'           Capture: 2: 'a' at position 0.

```

See also

- [Regular Expression Language - Quick Reference](#)
- [Backtracking](#)

Backreference Constructs in Regular Expressions

9/20/2022 • 10 minutes to read • [Edit Online](#)

Backreferences provide a convenient way to identify a repeated character or substring within a string. For example, if the input string contains multiple occurrences of an arbitrary substring, you can match the first occurrence with a capturing group, and then use a backreference to match subsequent occurrences of the substring.

NOTE

A separate syntax is used to refer to named and numbered capturing groups in replacement strings. For more information, see [Substitutions](#).

.NET defines separate language elements to refer to numbered and named capturing groups. For more information about capturing groups, see [Grouping Constructs](#).

Numbered Backreferences

A numbered backreference uses the following syntax:

`\ number`

where *number* is the ordinal position of the capturing group in the regular expression. For example, `\4` matches the contents of the fourth capturing group. If *number* is not defined in the regular expression pattern, a parsing error occurs, and the regular expression engine throws an [ArgumentException](#). For example, the regular expression `\b(\w+)\s\1` is valid, because `(\w+)` is the first and only capturing group in the expression. On the other hand, `\b(\w+)\s\2` is invalid and throws an argument exception, because there is no capturing group numbered `\2`. In addition, if *number* identifies a capturing group in a particular ordinal position, but that capturing group has been assigned a numeric name different than its ordinal position, the regular expression parser also throws an [ArgumentException](#).

Note the ambiguity between octal escape codes (such as `\16`) and `\ number` backreferences that use the same notation. This ambiguity is resolved as follows:

- The expressions `\1` through `\9` are always interpreted as backreferences, and not as octal codes.
- If the first digit of a multidigit expression is 8 or 9 (such as `\80` or `\91`), the expression is interpreted as a literal.
- Expressions from `\10` and greater are considered backreferences if there is a backreference corresponding to that number; otherwise, they are interpreted as octal codes.
- If a regular expression contains a backreference to an undefined group number, a parsing error occurs, and the regular expression engine throws an [ArgumentException](#).

If the ambiguity is a problem, you can use the `\k< name >` notation, which is unambiguous and cannot be confused with octal character codes. Similarly, hexadecimal codes such as `\xdd` are unambiguous and cannot be confused with backreferences.

The following example finds doubled word characters in a string. It defines a regular expression, `(\w)\1`, which consists of the following elements.

ELEMENT	DESCRIPTION
(\w)	Match a word character and assign it to the first capturing group.
\1	Match the next character that is the same as the value of the first capturing group.

```

using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string pattern = @"(\w)\1";
        string input = "trellis llama webbing dresser swagger";
        foreach (Match match in Regex.Matches(input, pattern))
            Console.WriteLine("Found '{0}' at position {1}.",
                match.Value, match.Index);
    }
}
// The example displays the following output:
//      Found 'll' at position 3.
//      Found 'll' at position 8.
//      Found 'bb' at position 16.
//      Found 'ss' at position 25.
//      Found 'gg' at position 33.

```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim pattern As String = "(\w)\1"
        Dim input As String = "trellis llama webbing dresser swagger"
        For Each match As Match In Regex.Matches(input, pattern)
            Console.WriteLine("Found '{0}' at position {1}.",
                match.Value, match.Index)
        Next
    End Sub
End Module
' The example displays the following output:
'      Found 'll' at position 3.
'      Found 'll' at position 8.
'      Found 'bb' at position 16.
'      Found 'ss' at position 25.
'      Found 'gg' at position 33.

```

Named Backreferences

A named backreference is defined by using the following syntax:

\k< *name* >

or:

\k' *name* '

where *name* is the name of a capturing group defined in the regular expression pattern. If *name* is not defined in the regular expression pattern, a parsing error occurs, and the regular expression engine throws an

ArgumentException

The following example finds doubled word characters in a string. It defines a regular expression,

(?<char>\w)\k<char>, which consists of the following elements.

ELEMENT	DESCRIPTION
(?<char>\w)	Match a word character and assign it to a capturing group named <code>char</code> .
\k<char>	Match the next character that is the same as the value of the <code>char</code> capturing group.

```
using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string pattern = @"(?<char>\w)\k<char>";
        string input = "trellis llama webbing dresser swagger";
        foreach (Match match in Regex.Matches(input, pattern))
            Console.WriteLine("Found '{0}' at position {1}.",
                match.Value, match.Index);
    }
}
// The example displays the following output:
//      Found 'll' at position 3.
//      Found 'll' at position 8.
//      Found 'bb' at position 16.
//      Found 'ss' at position 25.
//      Found 'gg' at position 33.
```

```
Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim pattern As String = "(?<char>\w)\k<char>"
        Dim input As String = "trellis llama webbing dresser swagger"
        For Each match As Match In Regex.Matches(input, pattern)
            Console.WriteLine("Found '{0}' at position {1}.",
                match.Value, match.Index)
        Next
    End Sub
End Module
' The example displays the following output:
'      Found 'll' at position 3.
'      Found 'll' at position 8.
'      Found 'bb' at position 16.
'      Found 'ss' at position 25.
'      Found 'gg' at position 33.
```

Named numeric backreferences

In a named backreference with `\k`, `name` can also be the string representation of a number. For example, the following example uses the regular expression `(?<2>\w)\k<2>` to find doubled word characters in a string. In this case, the example defines a capturing group that is explicitly named "2", and the backreference is correspondingly named "2".

```

using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string pattern = @"(?<2>\w)\k<2>";
        string input = "trellis llama webbing dresser swagger";
        foreach (Match match in Regex.Matches(input, pattern))
            Console.WriteLine("Found '{0}' at position {1}.",
                match.Value, match.Index);
    }
}
// The example displays the following output:
//     Found 'll' at position 3.
//     Found 'll' at position 8.
//     Found 'bb' at position 16.
//     Found 'ss' at position 25.
//     Found 'gg' at position 33.

```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim pattern As String = "(?<2>\w)\k<2>"
        Dim input As String = "trellis llama webbing dresser swagger"
        For Each match As Match In Regex.Matches(input, pattern)
            Console.WriteLine("Found '{0}' at position {1}.",
                match.Value, match.Index)
        Next
    End Sub
End Module
' The example displays the following output:
'     Found 'll' at position 3.
'     Found 'll' at position 8.
'     Found 'bb' at position 16.
'     Found 'ss' at position 25.
'     Found 'gg' at position 33.

```

If *name* is the string representation of a number, and no capturing group has that name, `\k< name >` is the same as the backreference `\ number`, where *number* is the ordinal position of the capture. In the following example, there is a single capturing group named `char`. The backreference construct refers to it as `\k<1>`. As the output from the example shows, the call to the `Regex.IsMatch` succeeds because `char` is the first capturing group.

```

using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        Console.WriteLine(Regex.IsMatch("aa", @"(?<char>\w)\k<1>"));
        // Displays "True".
    }
}

```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Console.WriteLine(Regex.IsMatch("aa", "(?<char>\w)\k<1>"))
        ' Displays "True".
    End Sub
End Module

```

However, if *name* is the string representation of a number and the capturing group in that position has been explicitly assigned a numeric name, the regular expression parser cannot identify the capturing group by its ordinal position. Instead, it throws an [ArgumentException](#). The only capturing group in the following example is named "2". Because the `\k` construct is used to define a backreference named "1", the regular expression parser is unable to identify the first capturing group and throws an exception.

```

using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        Console.WriteLine(Regex.IsMatch("aa", @"(?<2>\w)\k<1>"));
        // Throws an ArgumentException.
    }
}

```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Console.WriteLine(Regex.IsMatch("aa", "(?<2>\w)\k<1>"))
        ' Throws an ArgumentException.
    End Sub
End Module

```

What Backreferences Match

A backreference refers to the most recent definition of a group (the definition most immediately to the left, when matching left to right). When a group makes multiple captures, a backreference refers to the most recent capture.

The following example includes a regular expression pattern, `(?<1>a)(?<1>\1b)*`, which redefines the \1 named group. The following table describes each pattern in the regular expression.

PATTERN	DESCRIPTION
<code>(?<1>a)</code>	Match the character "a" and assign the result to the capturing group named <code>1</code> .
<code>(?<1>\1b)*</code>	Match zero or more occurrences of the group named <code>1</code> along with a "b", and assign the result to the capturing group named <code>1</code> .

```

using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string pattern = @"(?<1>a)(?<1>\1b)*";
        string input = "aababb";
        foreach (Match match in Regex.Matches(input, pattern))
        {
            Console.WriteLine("Match: " + match.Value);
            foreach (Group group in match.Groups)
                Console.WriteLine("    Group: " + group.Value);
        }
    }
}

// The example displays the following output:
//      Group: aababb
//      Group: abb

```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim pattern As String = "(?<1>a)(?<1>\1b)*"
        Dim input As String = "aababb"
        For Each match As Match In Regex.Matches(input, pattern)
            Console.WriteLine("Match: " + match.Value)
            For Each group As Group In match.Groups
                Console.WriteLine("    Group: " + group.Value)
            Next
        Next
        End Sub
    End Module
' The example display the following output:
'      Group: aababb
'      Group: abb

```

In comparing the regular expression with the input string ("aababb"), the regular expression engine performs the following operations:

1. It starts at the beginning of the string, and successfully matches "a" with the expression `(?<1>a)`. The value of the `\1` group is now "a".
2. It advances to the second character, and successfully matches the string "ab" with the expression `\1b`, or "ab". It then assigns the result, "ab" to `\1`.
3. It advances to the fourth character. The expression `(?<1>\1b)*` is to be matched zero or more times, so it successfully matches the string "abb" with the expression `\1b`. It assigns the result, "abb", back to `\1`.

In this example, `*` is a looping quantifier -- it is evaluated repeatedly until the regular expression engine cannot match the pattern it defines. Looping quantifiers do not clear group definitions.

If a group has not captured any substrings, a backreference to that group is undefined and never matches. This is illustrated by the regular expression pattern `\b(\p{Lu}{2})(\d{2})?(\p{Lu}{2})\b`, which is defined as follows:

PATTERN	DESCRIPTION
<code>\b</code>	Begin the match on a word boundary.

PATTERN	DESCRIPTION
(\p{Lu}{2})	Match two uppercase letters. This is the first capturing group.
(\d{2})?	Match zero or one occurrence of two decimal digits. This is the second capturing group.
(\p{Lu}{2})	Match two uppercase letters. This is the third capturing group.
\b	End the match on a word boundary.

An input string can match this regular expression even if the two decimal digits that are defined by the second capturing group are not present. The following example shows that even though the match is successful, an empty capturing group is found between two successful capturing groups.

```

using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string pattern = @"\b(\p{Lu}{2})(\d{2})?( \p{Lu}{2})\b";
        string[] inputs = { "AA22ZZ", "AABB" };
        foreach (string input in inputs)
        {
            Match match = Regex.Match(input, pattern);
            if (match.Success)
            {
                Console.WriteLine("Match in {0}: {1}", input, match.Value);
                if (match.Groups.Count > 1)
                {
                    for (int ctr = 1; ctr <= match.Groups.Count - 1; ctr++)
                    {
                        if (match.Groups[ctr].Success)
                            Console.WriteLine("Group {0}: {1}",
                                ctr, match.Groups[ctr].Value);
                        else
                            Console.WriteLine("Group {0}: <no match>", ctr);
                    }
                }
                Console.WriteLine();
            }
        }
    }
}

// The example displays the following output:
//      Match in AA22ZZ: AA22ZZ
//      Group 1: AA
//      Group 2: 22
//      Group 3: ZZ
//
//      Match in AABB: AABB
//      Group 1: AA
//      Group 2: <no match>
//      Group 3: BB

```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim pattern As String = "\b(\p{Lu}{2})(\d{2})?(\p{Lu}{2})\b"
        Dim inputs() As String = {"AA22ZZ", "AABB"}
        For Each input As String In inputs
            Dim match As Match = Regex.Match(input, pattern)
            If match.Success Then
                Console.WriteLine("Match in {0}: {1}", input, match.Value)
                If match.Groups.Count > 1 Then
                    For ctr As Integer = 1 To match.Groups.Count - 1
                        If match.Groups(ctr).Success Then
                            Console.WriteLine("Group {0}: {1}",
                                ctr, match.Groups(ctr).Value)
                        Else
                            Console.WriteLine("Group {0}: <no match>", ctr)
                        End If
                    Next
                End If
            End If
            Console.WriteLine()
        Next
    End Sub
End Module
' The example displays the following output:
'   Match in AA22ZZ: AA22ZZ
'   Group 1: AA
'   Group 2: 22
'   Group 3: ZZ
'
'   Match in AABB: AABB
'   Group 1: AA
'   Group 2: <no match>
'   Group 3: BB

```

See also

- [Regular Expression Language - Quick Reference](#)

Alternation Constructs in Regular Expressions

9/20/2022 • 8 minutes to read • [Edit Online](#)

Alternation constructs modify a regular expression to enable either/or or conditional matching. .NET supports three alternation constructs:

- [Pattern matching with |](#)
- [Conditional matching with \(?\(expression\)yes|no\)](#)
- [Conditional matching based on a valid captured group](#)

Pattern Matching with |

You can use the vertical bar (`|`) character to match any one of a series of patterns, where the `|` character separates each pattern.

Like the positive character class, the `|` character can be used to match any one of a number of single characters. The following example uses both a positive character class and either/or pattern matching with the `|` character to locate occurrences of the words "gray" or "grey" in a string. In this case, the `|` character produces a regular expression that is more verbose.

```
using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        // Regular expression using character class.
        string pattern1 = @"\bgr[ae]y\b";
        // Regular expression using either/or.
        string pattern2 = @"\bgr(a|e)y\b";

        string input = "The gray wolf blended in among the grey rocks.";
        foreach (Match match in Regex.Matches(input, pattern1))
            Console.WriteLine("'{}' found at position {}", match.Value, match.Index);
        Console.WriteLine();
        foreach (Match match in Regex.Matches(input, pattern2))
            Console.WriteLine("'{}' found at position {}", match.Value, match.Index);
    }
}
// The example displays the following output:
//      'gray' found at position 4
//      'grey' found at position 35
//      'gray' found at position 4
//      'grey' found at position 35
```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        ' Regular expression using character class.
        Dim pattern1 As String = "\bgr[ae]y\b"
        ' Regular expression using either/or.
        Dim pattern2 As String = "\bgr(a|e)y\b"

        Dim input As String = "The gray wolf blended in among the grey rocks."
        For Each match As Match In Regex.Matches(input, pattern1)
            Console.WriteLine("{0} found at position {1}", _
                match.Value, match.Index)
        Next
        Console.WriteLine()
        For Each match As Match In Regex.Matches(input, pattern2)
            Console.WriteLine("{0} found at position {1}", _
                match.Value, match.Index)
        Next
    End Sub
End Module
' The example displays the following output:
'     'gray' found at position 4
'     'grey' found at position 35
'
'     'gray' found at position 4
'     'grey' found at position 35

```

The regular expression that uses the `|` character, `\bgr(a|e)y\b`, is interpreted as shown in the following table:

PATTERN	DESCRIPTION
<code>\b</code>	Start at a word boundary.
<code>gr</code>	Match the characters "gr".
<code>(a e)</code>	Match either an "a" or an "e".
<code>y\b</code>	Match a "y" on a word boundary.

The `|` character can also be used to perform an either/or match with multiple characters or subexpressions, which can include any combination of character literals and regular expression language elements. (The character class does not provide this functionality.) The following example uses the `|` character to extract either a U.S. Social Security Number (SSN), which is a 9-digit number with the format `ddd-dd-dddd`, or a U.S. Employer Identification Number (EIN), which is a 9-digit number with the format `dd-ddddddd`.

```

using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string pattern = @"\b(\d{2}-\d{7}|\d{3}-\d{2}-\d{4})\b";
        string input = "01-9999999 020-333333 777-88-9999";
        Console.WriteLine("Matches for {0}:", pattern);
        foreach (Match match in Regex.Matches(input, pattern))
            Console.WriteLine("    {0} at position {1}", match.Value, match.Index);
    }
}
// The example displays the following output:
//      Matches for \b(\d{2}-\d{7}|\d{3}-\d{2}-\d{4})\b:
//          01-9999999 at position 0
//          777-88-9999 at position 22

```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim pattern As String = "\b(\d{2}-\d{7}|\d{3}-\d{2}-\d{4})\b"
        Dim input As String = "01-9999999 020-333333 777-88-9999"
        Console.WriteLine("Matches for {0}:", pattern)
        For Each match As Match In Regex.Matches(input, pattern)
            Console.WriteLine("    {0} at position {1}", match.Value, match.Index)
        Next
    End Sub
End Module
' The example displays the following output:
'      Matches for \b(\d{2}-\d{7}|\d{3}-\d{2}-\d{4})\b:
'          01-9999999 at position 0
'          777-88-9999 at position 22

```

The regular expression `\b(\d{2}-\d{7}|\d{3}-\d{2}-\d{4})\b` is interpreted as shown in the following table:

PATTERN	DESCRIPTION
<code>\b</code>	Start at a word boundary.
<code>(\d{2}-\d{7} \d{3}-\d{2}-\d{4})</code>	Match either of the following: two decimal digits followed by a hyphen followed by seven decimal digits; or three decimal digits, a hyphen, two decimal digits, another hyphen, and four decimal digits.
<code>\b</code>	End the match at a word boundary.

Conditional matching with an expression

This language element attempts to match one of two patterns depending on whether it can match an initial pattern. Its syntax is:

`(?(<expression>) yes | no)`

or

`(?(<expression>) yes | no)`

where *expression* is the initial pattern to match, *yes* is the pattern to match if *expression* is matched, and *no* is the optional pattern to match if *expression* is not matched (if a *no* pattern is not provided, it's equivalent to an empty *no*). The regular expression engine treats *expression* as a zero-width assertion; that is, the regular expression engine does not advance in the input stream after it evaluates *expression*. Therefore, this construct is equivalent to the following:

```
(?(?= expression ) yes | no )
```

where `(?= expression)` is a zero-width assertion construct. (For more information, see [Grouping Constructs](#).)

Because the regular expression engine interprets *expression* as an anchor (a zero-width assertion), *expression* must either be a zero-width assertion (for more information, see [Anchors](#)) or a subexpression that is also contained in *yes*. Otherwise, the *yes* pattern cannot be matched.

NOTE

If *expression* is a named or numbered capturing group, the alternation construct is interpreted as a capture test; for more information, see the next section, [Conditional Matching Based on a Valid Capture Group](#). In other words, the regular expression engine does not attempt to match the captured substring, but instead tests for the presence or absence of the group.

The following example is a variation of the example that appears in the [Either/Or Pattern Matching with |](#) section. It uses conditional matching to determine whether the first three characters after a word boundary are two digits followed by a hyphen. If they are, it attempts to match a U.S. Employer Identification Number (EIN). If not, it attempts to match a U.S. Social Security Number (SSN).

```
using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string pattern = @"\b(?(\d{2}-)\d{2}-\d{7}|\d{3}-\d{2}-\d{4})\b";
        string input = "01-9999999 020-333333 777-88-9999";
        Console.WriteLine("Matches for {0}:", pattern);
        foreach (Match match in Regex.Matches(input, pattern))
            Console.WriteLine("    {0} at position {1}", match.Value, match.Index);
    }
}
// The example displays the following output:
//      Matches for \b(\d{2}-\d{7}|\d{3}-\d{2}-\d{4})\b:
//          01-9999999 at position 0
//          777-88-9999 at position 22
```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim pattern As String = "\b(?:(\d{2}-)\d{2}-\d{7}|\d{3}-\d{2}-\d{4})\b"
        Dim input As String = "01-9999999 020-333333 777-88-9999"
        Console.WriteLine("Matches for {0}:", pattern)
        For Each match As Match In Regex.Matches(input, pattern)
            Console.WriteLine("  {0} at position {1}", match.Value, match.Index)
        Next
    End Sub
End Module
' The example displays the following output:
'   Matches for \b(?:(\d{2}-)\d{2}-\d{7}|\d{3}-\d{2}-\d{4})\b:
'   01-9999999 at position 0
'   777-88-9999 at position 22

```

The regular expression pattern `\b(?:(\d{2}-)\d{2}-\d{7}|\d{3}-\d{2}-\d{4})\b` is interpreted as shown in the following table:

PATTERN	DESCRIPTION
<code>\b</code>	Start at a word boundary.
<code>(?:(\d{2}-)</code>	Determine whether the next three characters consist of two digits followed by a hyphen.
<code>\d{2}-\d{7}</code>	If the previous pattern matches, match two digits followed by a hyphen followed by seven digits.
<code>\d{3}-\d{2}-\d{4}</code>	If the previous pattern does not match, match three decimal digits, a hyphen, two decimal digits, another hyphen, and four decimal digits.
<code>\b</code>	Match a word boundary.

Conditional matching based on a valid captured group

This language element attempts to match one of two patterns depending on whether it has matched a specified capturing group. Its syntax is:

`(?(<name>) yes | no)`

or

`(?(<name>) yes | no)`

or

`(?(<number>) yes | no)`

or

`(?(<number>) yes | no)`

where `name` is the name and `number` is the number of a capturing group, `yes` is the expression to match if `name` or `number` has a match, and `no` is the optional expression to match if it does not (if a `no` pattern is not provided, it's equivalent to an empty `no`).

If *name* does not correspond to the name of a capturing group that is used in the regular expression pattern, the alternation construct is interpreted as an expression test, as explained in the previous section. Typically, this means that *expression* evaluates to `false`. If *number* does not correspond to a numbered capturing group that is used in the regular expression pattern, the regular expression engine throws an [ArgumentException](#).

The following example is a variation of the example that appears in the [Either/Or Pattern Matching with |](#) section. It uses a capturing group named `n2` that consists of two digits followed by a hyphen. The alternation construct tests whether this capturing group has been matched in the input string. If it has, the alternation construct attempts to match the last seven digits of a nine-digit U.S. Employer Identification Number (EIN). If it has not, it attempts to match a nine-digit U.S. Social Security Number (SSN).

```
using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string pattern = @"\b(?:<n2>\d{2}-)?(?:n2)\d{7}|\d{3}-\d{2}-\d{4})\b";
        string input = "01-9999999 020-333333 777-88-9999";
        Console.WriteLine("Matches for {0}:", pattern);
        foreach (Match match in Regex.Matches(input, pattern))
            Console.WriteLine("    {0} at position {1}", match.Value, match.Index);
    }
}
// The example displays the following output:
//      Matches for \b(?:<n2>\d{2}-)?(?:n2)\d{7}|\d{3}-\d{2}-\d{4})\b:
//          01-9999999 at position 0
//          777-88-9999 at position 22
```

```
Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim pattern As String = "\b(?:<n2>\d{2}-)?(?:n2)\d{7}|\d{3}-\d{2}-\d{4})\b"
        Dim input As String = "01-9999999 020-333333 777-88-9999"
        Console.WriteLine("Matches for {0}:", pattern)
        For Each match As Match In Regex.Matches(input, pattern)
            Console.WriteLine("    {0} at position {1}", match.Value, match.Index)
        Next
    End Sub
End Module
```

The regular expression pattern `\b(?:<n2>\d{2}-)?(?:n2)\d{7}|\d{3}-\d{2}-\d{4})\b` is interpreted as shown in the following table:

PATTERN	DESCRIPTION
<code>\b</code>	Start at a word boundary.
<code>(?:<n2>\d{2}-)?</code>	Match zero or one occurrence of two digits followed by a hyphen. Name this capturing group <code>n2</code> .
<code>(?:n2)</code>	Test whether <code>n2</code> was matched in the input string.
<code>\d{7}</code>	If <code>n2</code> was matched, match seven decimal digits.

PATTERN	DESCRIPTION
<code> \d{3}-\d{2}-\d{4}</code>	If <code>n2</code> was not matched, match three decimal digits, a hyphen, two decimal digits, another hyphen, and four decimal digits.
<code>\b</code>	Match a word boundary.

A variation of this example that uses a numbered group instead of a named group is shown in the following example. Its regular expression pattern is `\b(\d{2}-)?(?:1)\d{7}|\d{3}-\d{2}-\d{4})\b`.

```
using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string pattern = @"\b(\d{2}-)?(?:1)\d{7}|\d{3}-\d{2}-\d{4})\b";
        string input = "01-9999999 020-333333 777-88-9999";
        Console.WriteLine("Matches for {0}:", pattern);
        foreach (Match match in Regex.Matches(input, pattern))
            Console.WriteLine("    {0} at position {1}", match.Value, match.Index);
    }
}
// The example display the following output:
//      Matches for \b(\d{2}-)?(?:1)\d{7}|\d{3}-\d{2}-\d{4})\b:
//      01-9999999 at position 0
//      777-88-9999 at position 22
```

```
Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim pattern As String = "\b(\d{2}-)?(?:1)\d{7}|\d{3}-\d{2}-\d{4})\b"
        Dim input As String = "01-9999999 020-333333 777-88-9999"
        Console.WriteLine("Matches for {0}:", pattern)
        For Each match As Match In Regex.Matches(input, pattern)
            Console.WriteLine("    {0} at position {1}", match.Value, match.Index)
        Next
    End Sub
End Module
' The example displays the following output:
'      Matches for \b(\d{2}-)?(?:1)\d{7}|\d{3}-\d{2}-\d{4})\b:
'      01-9999999 at position 0
'      777-88-9999 at position 22
```

See also

- [Regular Expression Language - Quick Reference](#)

Substitutions in Regular Expressions

9/20/2022 • 16 minutes to read • [Edit Online](#)

Substitutions are language elements that are recognized only within replacement patterns. They use a regular expression pattern to define all or part of the text that is to replace matched text in the input string. The replacement pattern can consist of one or more substitutions along with literal characters. Replacement patterns are provided to overloads of the [Regex.Replace](#) method that have a `replacement` parameter and to the [Match.Result](#) method. The methods replace the matched pattern with the pattern that is defined by the `replacement` parameter.

.NET defines the substitution elements listed in the following table.

SUBSTITUTION	DESCRIPTION
<code>\$ number</code>	Includes the last substring matched by the capturing group that is identified by <i>number</i> , where <i>number</i> is a decimal value, in the replacement string. For more information, see Substituting a Numbered Group .
<code> \${ name }</code>	Includes the last substring matched by the named group that is designated by <code>(?< name >)</code> in the replacement string. For more information, see Substituting a Named Group .
<code> \$\$</code>	Includes a single "\$" literal in the replacement string. For more information, see Substituting a "\$" Symbol .
<code> \$&</code>	Includes a copy of the entire match in the replacement string. For more information, see Substituting the Entire Match .
<code> \$`</code>	Includes all the text of the input string before the match in the replacement string. For more information, see Substituting the Text before the Match .
<code> \$'</code>	Includes all the text of the input string after the match in the replacement string. For more information, see Substituting the Text after the Match .
<code> \$+</code>	Includes the last group captured in the replacement string. For more information, see Substituting the Last Captured Group .
<code> \$_</code>	Includes the entire input string in the replacement string. For more information, see Substituting the Entire Input String .

Substitution Elements and Replacement Patterns

Substitutions are the only special constructs recognized in a replacement pattern. None of the other regular expression language elements, including character escapes and the period (`.`), which matches any character, are supported. Similarly, substitution language elements are recognized only in replacement patterns and are never valid in regular expression patterns.

The only character that can appear either in a regular expression pattern or in a substitution is the `$` character, although it has a different meaning in each context. In a regular expression pattern, `$` is an anchor that matches the end of the string. In a replacement pattern, `$` indicates the beginning of a substitution.

NOTE

For functionality similar to a replacement pattern within a regular expression, use a backreference. For more information about backreferences, see [Backreference Constructs](#).

Substituting a Numbered Group

The `$number` language element includes the last substring matched by the *number* capturing group in the replacement string, where *number* is the index of the capturing group. For example, the replacement pattern `$1` indicates that the matched substring is to be replaced by the first captured group. For more information about numbered capturing groups, see [Grouping Constructs](#).

All digits that follow `$` are interpreted as belonging to the *number* group. If this is not your intent, you can substitute a named group instead. For example, you can use the replacement string `${1}1` instead of `$11` to define the replacement string as the value of the first captured group along with the number "1". For more information, see [Substituting a Named Group](#).

Capturing groups that are not explicitly assigned names using the `(?<name>)` syntax are numbered from left to right starting at one. Named groups are also numbered from left to right, starting at one greater than the index of the last unnamed group. For example, in the regular expression `(\w)(?<digit>\d)`, the index of the `digit` named group is 2.

If *number* does not specify a valid capturing group defined in the regular expression pattern, `$number` is interpreted as a literal character sequence that is used to replace each match.

The following example uses the `$number` substitution to strip the currency symbol from a decimal value. It removes currency symbols found at the beginning or end of a monetary value, and recognizes the two most common decimal separators ("." and ",").

```
using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string pattern = @"\p{Sc}*(\s?\d+[.,]?\d*)\p{Sc}*";
        string replacement = "$1";
        string input = "$16.32 12.19 £16.29 €18.29 €18,29";
        string result = Regex.Replace(input, pattern, replacement);
        Console.WriteLine(result);
    }
}
// The example displays the following output:
//      16.32 12.19 16.29 18.29  18,29
```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim pattern As String = "\p{Sc}*(\s?\d+[.,]?\d*)\p{Sc}*"
        Dim replacement As String = "$1"
        Dim input As String = "$16.32 12.19 €16.29 €18.29  €18,29"
        Dim result As String = Regex.Replace(input, pattern, replacement)
        Console.WriteLine(result)
    End Sub
End Module
' The example displays the following output:
'      16.32 12.19 16.29 18.29  18,29

```

The regular expression pattern `\p{Sc}*(\s?\d+[.,]?\d*)\p{Sc}*` is defined as shown in the following table.

PATTERN	DESCRIPTION
<code>\p{Sc}*</code>	Match zero or more currency symbol characters.
<code>\s?</code>	Match zero or one white-space characters.
<code>\d+</code>	Match one or more decimal digits.
<code>[.,]?</code>	Match zero or one period or comma.
<code>\d*</code>	Match zero or more decimal digits.
<code>(\s?\d+[.,]?\d*)</code>	Match a white space followed by one or more decimal digits, followed by zero or one period or comma, followed by zero or more decimal digits. This is the first capturing group. Because the replacement pattern is <code>\$1</code> , the call to the <code>Regex.Replace</code> method replaces the entire matched substring with this captured group.

Substituting a Named Group

The `${ name }` language element substitutes the last substring matched by the `name` capturing group, where `name` is the name of a capturing group defined by the `(?< name >)` language element. For more information about named capturing groups, see [Grouping Constructs](#).

If `name` doesn't specify a valid named capturing group defined in the regular expression pattern but consists of digits, `${ name }` is interpreted as a numbered group.

If `name` specifies neither a valid named capturing group nor a valid numbered capturing group defined in the regular expression pattern, `${ name }` is interpreted as a literal character sequence that is used to replace each match.

The following example uses the `${ name }` substitution to strip the currency symbol from a decimal value. It removes currency symbols found at the beginning or end of a monetary value, and recognizes the two most common decimal separators ("." and ",").

```

using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string pattern = @"\p{Sc}*(?<amount>\s?\d+[.,]?\d*)\p{Sc}*";
        string replacement = "${amount}";
        string input = "$16.32 12.19 £16.29 €18.29 €18,29";
        string result = Regex.Replace(input, pattern, replacement);
        Console.WriteLine(result);
    }
}
// The example displays the following output:
//      16.32 12.19 16.29 18.29  18,29

```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim pattern As String = "\p{Sc}*(?<amount>\s?\d+[.,]?\d*)\p{Sc}*"
        Dim replacement As String = "${amount}"
        Dim input As String = "$16.32 12.19 £16.29 €18.29 €18,29"
        Dim result As String = Regex.Replace(input, pattern, replacement)
        Console.WriteLine(result)
    End Sub
End Module
' The example displays the following output:
'      16.32 12.19 16.29 18.29  18,29

```

The regular expression pattern `\p{Sc}*(?<amount>\s?\d[.,]?\d*)\p{Sc}* is defined as shown in the following table.`

PATTERN	DESCRIPTION
<code>\p{Sc}*</code>	Match zero or more currency symbol characters.
<code>\s?</code>	Match zero or one white-space characters.
<code>\d+</code>	Match one or more decimal digits.
<code>[.,]?</code>	Match zero or one period or comma.
<code>\d*</code>	Match zero or more decimal digits.
<code>(?<amount>\s?\d[.,]?\d*)</code>	Match a white space, followed by one or more decimal digits, followed by zero or one period or comma, followed by zero or more decimal digits. This is the capturing group named <code>amount</code> . Because the replacement pattern is <code>\${amount}</code> , the call to the <code>Regex.Replace</code> method replaces the entire matched substring with this captured group.

Substituting a "\$" Character

The `$$` substitution inserts a literal "\$" character in the replaced string.

The following example uses the `NumberFormatInfo` object to determine the current culture's currency symbol

and its placement in a currency string. It then builds both a regular expression pattern and a replacement pattern dynamically. If the example is run on a computer whose current culture is en-US, it generates the regular expression pattern `\b(\d+)(\.(.\d+))?` and the replacement pattern `$$ $1$2`. The replacement pattern replaces the matched text with a currency symbol and a space followed by the first and second captured groups.

```
using System;
using System.Globalization;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        // Define array of decimal values.
        string[] values= { "16.35", "19.72", "1234", "0.99"};
        // Determine whether currency precedes (True) or follows (False) number.
        bool precedes = NumberFormatInfo.CurrentInfo.CurrencyPositivePattern % 2 == 0;
        // Get decimal separator.
        string cSeparator = NumberFormatInfo.CurrentInfo.CurrencyDecimalSeparator;
        // Get currency symbol.
        string symbol = NumberFormatInfo.CurrentInfo.CurrencySymbol;
        // If symbol is a "$", add an extra "$".
        if (symbol == "$") symbol = "$$";

        // Define regular expression pattern and replacement string.
        string pattern = @"\b(\d+)" + cSeparator + @"(\d+)?";
        string replacement = "$1$2";
        replacement = precedes ? symbol + " " + replacement : replacement + " " + symbol;
        foreach (string value in values)
            Console.WriteLine("{0} --> {1}", value, Regex.Replace(value, pattern, replacement));
    }
}

// The example displays the following output:
//      16.35 --> $ 16.35
//      19.72 --> $ 19.72
//      1234 --> $ 1234
//      0.99 --> $ 0.99
```

```

Imports System.Globalization
Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        ' Define array of decimal values.
        Dim values() As String = {"16.35", "19.72", "1234", "0.99"}
        ' Determine whether currency precedes (True) or follows (False) number.
        Dim precedes As Boolean = (NumberFormatInfo.CurrentInfo.CurrencyPositivePattern Mod 2 = 0)
        ' Get decimal separator.
        Dim cSeparator As String = NumberFormatInfo.CurrentInfo.CurrencyDecimalSeparator
        ' Get currency symbol.
        Dim symbol As String = NumberFormatInfo.CurrentInfo.CurrencySymbol
        ' If symbol is a "$", add an extra "$".
        If symbol = "$" Then symbol = "$$"

        ' Define regular expression pattern and replacement string.
        Dim pattern As String = "\b(\d+)(\.(.\d+))?"
        Dim replacement As String = "$1$2"
        replacement = If(precedes, symbol + " " + replacement, replacement + " " + symbol)
        For Each value In values
            Console.WriteLine("{0} --> {1}", value, Regex.Replace(value, pattern, replacement))
        Next
    End Sub
End Module
' The example displays the following output:
' 16.35 --> $ 16.35
' 19.72 --> $ 19.72
' 1234 --> $ 1234
' 0.99 --> $ 0.99

```

The regular expression pattern `\b(\d+)(\.(.\d+))?` is defined as shown in the following table.

PATTERN	DESCRIPTION
<code>\b</code>	Start the match at the beginning of a word boundary.
<code>(\d+)</code>	Match one or more decimal digits. This is the first capturing group.
<code>\.</code>	Match a period (the decimal separator).
<code>(\d+)</code>	Match one or more decimal digits. This is the third capturing group.
<code>(\.(.\d+))?</code>	Match zero or one occurrence of a period followed by one or more decimal digits. This is the second capturing group.

Substituting the Entire Match

The `$&` substitution includes the entire match in the replacement string. Often, it is used to add a substring to the beginning or end of the matched string. For example, the `($&)` replacement pattern adds parentheses to the beginning and end of each match. If there is no match, the `$&` substitution has no effect.

The following example uses the `$&` substitution to add quotation marks at the beginning and end of book titles stored in a string array.

```

using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string pattern = @"^(\w+\s?)+$";
        string[] titles = { "A Tale of Two Cities",
                            "The Hound of the Baskervilles",
                            "The Protestant Ethic and the Spirit of Capitalism",
                            "The Origin of Species" };
        string replacement = "\"$&\"";
        foreach (string title in titles)
            Console.WriteLine(Regex.Replace(title, pattern, replacement));
    }
}

// The example displays the following output:
//      "A Tale of Two Cities"
//      "The Hound of the Baskervilles"
//      "The Protestant Ethic and the Spirit of Capitalism"
//      "The Origin of Species"

```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim pattern As String = "^(\w+\s?)+$"
        Dim titles() As String = {"A Tale of Two Cities", _
                                "The Hound of the Baskervilles", _
                                "The Protestant Ethic and the Spirit of Capitalism", _
                                "The Origin of Species"}
        Dim replacement As String = """$&"""
        For Each title As String In titles
            Console.WriteLine(Regex.Replace(title, pattern, replacement))
        Next
    End Sub
End Module

' The example displays the following output:
'      "A Tale of Two Cities"
'      "The Hound of the Baskervilles"
'      "The Protestant Ethic and the Spirit of Capitalism"
'      "The Origin of Species"

```

The regular expression pattern `^(\w+\s?)+$` is defined as shown in the following table.

PATTERN	DESCRIPTION
<code>^</code>	Start the match at the beginning of the input string.
<code>(\w+\s?)+</code>	Match the pattern of one or more word characters followed by zero or one white-space characters one or more times.
<code>\$</code>	Match the end of the input string.

The `"$&"` replacement pattern adds a literal quotation mark to the beginning and end of each match.

Substituting the Text Before the Match

The `$`` substitution replaces the matched string with the entire input string before the match. That is, it

duplicates the input string up to the match while removing the matched text. Any text that follows the matched text is unchanged in the result string. If there are multiple matches in an input string, the replacement text is derived from the original input string, rather than from the string in which text has been replaced by earlier matches. (The example provides an illustration.) If there is no match, the `$`` substitution has no effect.

The following example uses the regular expression pattern `\d+` to match a sequence of one or more decimal digits in the input string. The replacement string `$`` replaces these digits with the text that precedes the match.

```
using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string input = "aa1bb2cc3dd4ee5";
        string pattern = @"\d+";
        string substitution = "$`";
        Console.WriteLine("Matches:");
        foreach (Match match in Regex.Matches(input, pattern))
            Console.WriteLine("  {0} at position {1}", match.Value, match.Index);

        Console.WriteLine("Input string:  {0}", input);
        Console.WriteLine("Output string: " +
            Regex.Replace(input, pattern, substitution));
    }
}

// The example displays the following output:
//   Matches:
//     1 at position 2
//     2 at position 5
//     3 at position 8
//     4 at position 11
//     5 at position 14
//   Input string: aa1bb2cc3dd4ee5
//   Output string: aaaabbaa1bbccaa1bb2ccddaa1bb2cc3ddeaa1bb2cc3dd4ee
```

```
Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim input As String = "aa1bb2cc3dd4ee5"
        Dim pattern As String = "\d+"
        Dim substitution As String = "$`"
        Console.WriteLine("Matches:")
        For Each match As Match In Regex.Matches(input, pattern)
            Console.WriteLine("  {0} at position {1}", match.Value, match.Index)
        Next
        Console.WriteLine("Input string:  {0}", input)
        Console.WriteLine("Output string: " +
            Regex.Replace(input, pattern, substitution))
    End Sub
End Module

' The example displays the following output:
'   Matches:
'     1 at position 2
'     2 at position 5
'     3 at position 8
'     4 at position 11
'     5 at position 14
'   Input string: aa1bb2cc3dd4ee5
'   Output string: aaaabbaa1bbccaa1bb2ccddaa1bb2cc3ddeaa1bb2cc3dd4ee
```

In this example, the input string "aa1bb2cc3dd4ee5" contains five matches. The following table illustrates how the \$ substitution causes the regular expression engine to replace each match in the input string. Inserted text is shown in bold in the results column.

MATCH	POSITION	STRING BEFORE MATCH	RESULT STRING
1	2	aa	aaa ab b2cc3dd4ee5
2	5	aa1bb	aaaabb aa1bb cc3dd4ee5
3	8	aa1bb2cc	aaaabb aa1bb cca a1bb2cc d4ee5
4	11	aa1bb2cc3dd	aaaabb aa1bb cca a1bb2cc dd4ee5
5	14	aa1bb2cc3dd4ee	aaaabb aa1bb cca a1bb2cc 3ddee aa1bb2cc 3dd4ee

Substituting the Text After the Match

The \$ substitution replaces the matched string with the entire input string after the match. That is, it duplicates the input string after the match while removing the matched text. Any text that precedes the matched text is unchanged in the result string. If there is no match, the \$ substitution has no effect.

The following example uses the regular expression pattern \d+ to match a sequence of one or more decimal digits in the input string. The replacement string \$ replaces these digits with the text that follows the match.

```
using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string input = "aa1bb2cc3dd4ee5";
        string pattern = @"\d+";
        string substitution = "$!";
        Console.WriteLine("Matches:");
        foreach (Match match in Regex.Matches(input, pattern))
            Console.WriteLine("  {0} at position {1}", match.Value, match.Index);
        Console.WriteLine("Input string: {0}", input);
        Console.WriteLine("Output string: " +
            Regex.Replace(input, pattern, substitution));
    }
}
// The example displays the following output:
//   Matches:
//     1 at position 2
//     2 at position 5
//     3 at position 8
//     4 at position 11
//     5 at position 14
//   Input string: aa1bb2cc3dd4ee5
//   Output string: aabb2cc3dd4ee5bbcc3dd4ee5ccdd4ee5ddee5ee
```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim input As String = "aa1bb2cc3dd4ee5"
        Dim pattern As String = "\d+"
        Dim substitution As String = "$"
        Console.WriteLine("Matches:")
        For Each match As Match In Regex.Matches(input, pattern)
            Console.WriteLine("    {0} at position {1}", match.Value, match.Index)
        Next
        Console.WriteLine("Input string: {0}", input)
        Console.WriteLine("Output string: " + _
                         Regex.Replace(input, pattern, substitution))
    End Sub
End Module
' The example displays the following output:
'   Matches:
'   1 at position 2
'   2 at position 5
'   3 at position 8
'   4 at position 11
'   5 at position 14
'   Input string: aa1bb2cc3dd4ee5
'   Output string: aabb2cc3dd4ee5bbcc3dd4ee5ccdd4ee5dde5ee

```

In this example, the input string "aa1bb2cc3dd4ee5" contains five matches. The following table illustrates how the \$' substitution causes the regular expression engine to replace each match in the input string. Inserted text is shown in bold in the results column.

MATCH	POSITION	STRING AFTER MATCH	RESULT STRING
1	2	bb2cc3dd4ee5	aabb2 cc3dd4ee5 bb2cc3d d4ee5
2	5	cc3dd4ee5	aabb2cc3dd4ee5bbcc3dd 4 ee5 cc3dd4ee5
3	8	dd4ee5	aabb2cc3dd4ee5bbcc3dd4e e5ccdd 4ee5 dd4ee5
4	11	ee5	aabb2cc3dd4ee5bbcc3dd4e e5ccdd4ee5dde e5 ee5
5	14	String.Empty	aabb2cc3dd4ee5bbcc3dd4e e5ccdd4ee5dde5ee

Substituting the Last Captured Group

The \$+ substitution replaces the matched string with the last captured group. If there are no captured groups or if the value of the last captured group is `String.Empty`, the \$+ substitution has no effect.

The following example identifies duplicate words in a string and uses the \$+ substitution to replace them with a single occurrence of the word. The `RegexOptions.IgnoreCase` option is used to ensure that words that differ in case but that are otherwise identical are considered duplicates.

```

using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string pattern = @"\b(\w+)\s\1\b";
        string substitution = "$+";
        string input = "The the dog jumped over the fence fence.";
        Console.WriteLine(Regex.Replace(input, pattern, substitution,
            RegexOptions.IgnoreCase));
    }
}

// The example displays the following output:
//      The dog jumped over the fence.

```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim pattern As String = "\b(\w+)\s\1\b"
        Dim substitution As String = "$+"
        Dim input As String = "The the dog jumped over the fence fence."
        Console.WriteLine(Regex.Replace(input, pattern, substitution, _
            RegexOptions.IgnoreCase))

    End Sub
End Module
' The example displays the following output:
'      The dog jumped over the fence.

```

The regular expression pattern `\b(\w+)\s\1\b` is defined as shown in the following table.

PATTERN	DESCRIPTION
<code>\b</code>	Begin the match at a word boundary.
<code>(\w+)</code>	Match one or more word characters. This is the first capturing group.
<code>\s</code>	Match a white-space character.
<code>\1</code>	Match the first captured group.
<code>\b</code>	End the match at a word boundary.

Substituting the Entire Input String

The `$_` substitution replaces the matched string with the entire input string. That is, it removes the matched text and replaces it with the entire string, including the matched text.

The following example matches one or more decimal digits in the input string. It uses the `$_` substitution to replace them with the entire input string.

```

using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string input = "ABC123DEF456";
        string pattern = @"\d+";
        string substitution = "$_";
        Console.WriteLine("Original string: {0}", input);
        Console.WriteLine("String with substitution: {0}",
            Regex.Replace(input, pattern, substitution));
    }
}
// The example displays the following output:
//      Original string: ABC123DEF456
//      String with substitution: ABCABC123DEF456DEFABC123DEF456

```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim input As String = "ABC123DEF456"
        Dim pattern As String = "\d+"
        Dim substitution As String = "$_"
        Console.WriteLine("Original string: {0}", input)
        Console.WriteLine("String with substitution: {0}",
            Regex.Replace(input, pattern, substitution))
    End Sub
End Module
' The example displays the following output:
'      Original string: ABC123DEF456
'      String with substitution: ABCABC123DEF456DEFABC123DEF456

```

In this example, the input string `"ABC123DEF456"` contains two matches. The following table illustrates how the `$_` substitution causes the regular expression engine to replace each match in the input string. Inserted text is shown in bold in the results column.

MATCH	POSITION	MATCH	RESULT STRING
1	3	123	ABC A BC123DEF F 456DEF456
2	5	456	ABCABC123DEF456DEF A B C 123DEF456

See also

- [Regular Expression Language - Quick Reference](#)

Regular expression options

9/20/2022 • 48 minutes to read • [Edit Online](#)

By default, the comparison of an input string with any literal characters in a regular expression pattern is case-sensitive, white space in a regular expression pattern is interpreted as literal white-space characters, and capturing groups in a regular expression are named implicitly as well as explicitly. You can modify these and several other aspects of default regular expression behavior by specifying regular expression options. Some of these options, which are listed in the following table, can be included inline as part of the regular expression pattern, or they can be supplied to a `System.Text.RegularExpressions.Regex` class constructor or static pattern matching method as a `System.Text.RegularExpressions.RegexOptions` enumeration value.

REGEXOPTIONS MEMBER	INLINE CHARACTER	EFFECT
None	Not available	Use default behavior. For more information, see Default Options .
IgnoreCase	i	Use case-insensitive matching. For more information, see Case-Insensitive Matching .
Multiline	m	Use multiline mode, where <code>^</code> and <code>\$</code> match the beginning and end of each line (instead of the beginning and end of the input string). For more information, see Multiline Mode .
Singleline	s	Use single-line mode, where the period (.) matches every character (instead of every character except <code>\n</code>). For more information, see Single-line Mode .
ExplicitCapture	n	Do not capture unnamed groups. The only valid captures are explicitly named or numbered groups of the form <code>(?< name > subexpression)</code> . For more information, see Explicit Captures Only .
Compiled	Not available	Compile the regular expression to an assembly. For more information, see Compiled Regular Expressions .
IgnorePatternWhitespace	x	Exclude unescaped white space from the pattern, and enable comments after a number sign (<code>#</code>). For more information, see Ignore White Space .
RightToLeft	Not available	Change the search direction. Search moves from right to left instead of from left to right. For more information, see Right-to-Left Mode .

RegExOptions Member	Inline Character	Effect
ECMAScript	Not available	Enable ECMAScript-compliant behavior for the expression. For more information, see ECMAScript Matching Behavior .
CultureInvariant	Not available	Ignore cultural differences in language. For more information, see Comparison Using the Invariant Culture .
NonBacktracking	Not available	Match using an approach that avoids backtracking and guarantees linear-time processing in the length of the input. (Available in .NET 7 and later versions.)

Specify options

You can specify options for regular expressions in one of three ways:

- In the `options` parameter of a `System.Text.RegularExpressions.Regex` class constructor or static (`Shared` in Visual Basic) pattern-matching method, such as `Regex(String, RegexOptions)` or `Regex.Match(String, String, RegexOptions)`. The `options` parameter is a bitwise OR combination of `System.Text.RegularExpressions.RegexOptions` enumerated values.

When options are supplied to a `Regex` instance by using the `options` parameter of a class constructor, the options are assigned to the `System.Text.RegularExpressions.RegexOptions` property. However, the `System.Text.RegularExpressions.RegexOptions` property does not reflect inline options in the regular expression pattern itself.

The following example provides an illustration. It uses the `options` parameter of the `Regex.Match(String, String, RegexOptions)` method to enable case-insensitive matching and to ignore pattern white space when identifying words that begin with the letter "d".

```
string pattern = @"\d \w+ \s";
string input = "Dogs are decidedly good pets.";
RegexOptions options = RegexOptions.IgnoreCase | RegexOptions.IgnorePatternWhitespace;

foreach (Match match in Regex.Matches(input, pattern, options))
    Console.WriteLine("{0}// found at index {1}.", match.Value, match.Index);
// The example displays the following output:
//      'Dogs' // found at index 0.
//      'decidedly' // found at index 9.
```

```
Dim pattern As String = "d \w+ \s"
Dim input As String = "Dogs are decidedly good pets."
Dim options As RegexOptions = RegexOptions.IgnoreCase Or RegexOptions.IgnorePatternWhitespace

For Each match As Match In Regex.Matches(input, pattern, options)
    Console.WriteLine("{0}' found at index {1}.", match.Value, match.Index)
Next
' The example displays the following output:
'      'Dogs' found at index 0.
'      'decidedly' found at index 9.
```

- By applying inline options in a regular expression pattern with the syntax `(?imnsx-imnsx)`. The option

applies to the pattern from the point that the option is defined to either the end of the pattern or to the point at which the option is undefined by another inline option. Note that the

[System.Text.RegularExpressions.RegexOptions](#) property of a [Regex](#) instance does not reflect these inline options. For more information, see the [Miscellaneous Constructs](#) topic.

The following example provides an illustration. It uses inline options to enable case-insensitive matching and to ignore pattern white space when identifying words that begin with the letter "d".

```
string pattern = @"(?ix) d \w+ \s";
string input = "Dogs are decidedly good pets.";

foreach (Match match in Regex.Matches(input, pattern))
    Console.WriteLine("{0}// found at index {1}.", match.Value, match.Index);
// The example displays the following output:
//      'Dogs // found at index 0.
//      'decidedly // found at index 9.
```

```
Dim pattern As String = "\b(?ix) d \w+ \s"
Dim input As String = "Dogs are decidedly good pets."

For Each match As Match In Regex.Matches(input, pattern)
    Console.WriteLine("{0}' found at index {1}.", match.Value, match.Index)
Next
' The example displays the following output:
'      'Dogs ' found at index 0.
'      'decidedly ' found at index 9.
```

- By applying inline options in a particular grouping construct in a regular expression pattern with the syntax `(?imnsx-imnsx: subexpression)`. No sign before a set of options turns the set on; a minus sign before a set of options turns the set off. (`[]` is a fixed part of the language construct's syntax that is required whether options are enabled or disabled.) The option applies only to that group. For more information, see [Grouping Constructs](#).

The following example provides an illustration. It uses inline options in a grouping construct to enable case-insensitive matching and to ignore pattern white space when identifying words that begin with the letter "d".

```
string pattern = @"\b(?ix: d \w+)\s";
string input = "Dogs are decidedly good pets.";

foreach (Match match in Regex.Matches(input, pattern))
    Console.WriteLine("{0}// found at index {1}.", match.Value, match.Index);
// The example displays the following output:
//      'Dogs // found at index 0.
//      'decidedly // found at index 9.
```

```
Dim pattern As String = "\b(?ix: d \w+)\s"
Dim input As String = "Dogs are decidedly good pets."

For Each match As Match In Regex.Matches(input, pattern)
    Console.WriteLine("{0}' found at index {1}.", match.Value, match.Index)
Next
' The example displays the following output:
'      'Dogs ' found at index 0.
'      'decidedly ' found at index 9.
```

If options are specified inline, a minus sign (`-`) before an option or set of options turns off those options. For

example, the inline construct `(?ix-ms)` turns on the `RegexOptions.IgnoreCase` and `RegexOptions.IgnorePatternWhitespace` options and turns off the `RegexOptions.Multiline` and `RegexOptions.Singleline` options. All regular expression options are turned off by default.

NOTE

If the regular expression options specified in the `options` parameter of a constructor or method call conflict with the options specified inline in a regular expression pattern, the inline options are used.

The following five regular expression options can be set both with the options parameter and inline:

- `RegexOptions.IgnoreCase`
- `RegexOptions.Multiline`
- `RegexOptions.Singleline`
- `RegexOptions.ExplicitCapture`
- `RegexOptions.IgnorePatternWhitespace`

The following five regular expression options can be set using the `options` parameter but cannot be set inline:

- `RegexOptions.None`
- `RegexOptions.Compiled`
- `RegexOptions.RightToLeft`
- `RegexOptions.CultureInvariant`
- `RegexOptions.ECMAScript`

Determine options

You can determine which options were provided to a `Regex` object when it was instantiated by retrieving the value of the read-only `Regex.Options` property. This property is particularly useful for determining the options that are defined for a compiled regular expression created by the `Regex.CompileToAssembly` method.

To test for the presence of any option except `RegexOptions.None`, perform an AND operation with the value of the `Regex.Options` property and the `RegexOptions` value in which you are interested. Then test whether the result equals that `RegexOptions` value. The following example tests whether the `RegexOptions.IgnoreCase` option has been set.

```
if ((rgx.Options & RegexOptions.IgnoreCase) == RegexOptions.IgnoreCase)
    Console.WriteLine("Case-insensitive pattern comparison.");
else
    Console.WriteLine("Case-sensitive pattern comparison.");
```

```
If (rgx.Options And RegexOptions.IgnoreCase) = RegexOptions.IgnoreCase Then
    Console.WriteLine("Case-insensitive pattern comparison.")
Else
    Console.WriteLine("Case-sensitive pattern comparison.")
End If
```

To test for `RegexOptions.None`, determine whether the value of the `Regex.Options` property is equal to `RegexOptions.None`, as the following example illustrates.

```
if (rgx.Options == RegexOptions.None)
    Console.WriteLine("No options have been set.");
```

```
If rgx.Options = RegexOptions.None Then
    Console.WriteLine("No options have been set.")
End If
```

The following sections list the options supported by regular expression in .NET.

Default options

The [RegexOptions.None](#) option indicates that no options have been specified, and the regular expression engine uses its default behavior. This includes the following:

- The pattern is interpreted as a canonical rather than an ECMAScript regular expression.
- The regular expression pattern is matched in the input string from left to right.
- Comparisons are case-sensitive.
- The `^` and `$` language elements match the beginning and end of the input string. The end of the input string can be a trailing newline `\n` character.
- The `.` language element matches every character except `\n`.
- Any white space in a regular expression pattern is interpreted as a literal space character.
- The conventions of the current culture are used when comparing the pattern to the input string.
- Capturing groups in the regular expression pattern are implicit as well as explicit.

NOTE

The [RegexOptions.None](#) option has no inline equivalent. When regular expression options are applied inline, the default behavior is restored on an option-by-option basis, by turning a particular option off. For example, `(?i)` turns on case-insensitive comparison, and `(?-i)` restores the default case-sensitive comparison.

Because the [RegexOptions.None](#) option represents the default behavior of the regular expression engine, it is rarely explicitly specified in a method call. A constructor or static pattern-matching method without an `options` parameter is called instead.

Case-insensitive matching

The [IgnoreCase](#) option, or the `i` inline option, provides case-insensitive matching. By default, the casing conventions of the current culture are used.

The following example defines a regular expression pattern, `\bthe\w*\b`, that matches all words starting with "the". Because the first call to the [Match](#) method uses the default case-sensitive comparison, the output indicates that the string "The" that begins the sentence is not matched. It is matched when the [Match](#) method is called with options set to [IgnoreCase](#).

```

using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string pattern = @"\bthe\w*\b";
        string input = "The man then told them about that event.";
        foreach (Match match in Regex.Matches(input, pattern))
            Console.WriteLine("Found {0} at index {1}.", match.Value, match.Index);

        Console.WriteLine();
        foreach (Match match in Regex.Matches(input, pattern,
                                              RegexOptions.IgnoreCase))
            Console.WriteLine("Found {0} at index {1}.", match.Value, match.Index);
    }
}

// The example displays the following output:
//      Found then at index 8.
//      Found them at index 18.
//
//      Found The at index 0.
//      Found then at index 8.
//      Found them at index 18.

```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim pattern As String = "\bthe\w*\b"
        Dim input As String = "The man then told them about that event."
        For Each match As Match In Regex.Matches(input, pattern)
            Console.WriteLine("Found {0} at index {1}.", match.Value, match.Index)
        Next
        Console.WriteLine()
        For Each match As Match In Regex.Matches(input, pattern, _
                                              RegexOptions.IgnoreCase)
            Console.WriteLine("Found {0} at index {1}.", match.Value, match.Index)
        Next
    End Sub
End Module
' The example displays the following output:
'      Found then at index 8.
'      Found them at index 18.
'
'      Found The at index 0.
'      Found then at index 8.
'      Found them at index 18.

```

The following example modifies the regular expression pattern from the previous example to use inline options instead of the `options` parameter to provide case-insensitive comparison. The first pattern defines the case-insensitive option in a grouping construct that applies only to the letter "t" in the string "the". Because the option construct occurs at the beginning of the pattern, the second pattern applies the case-insensitive option to the entire regular expression.

```

using System;
using System.Text.RegularExpressions;

public class CaseExample
{
    public static void Main()
    {
        string pattern = @"\b(?i:t)he\w*\b";
        string input = "The man then told them about that event.";
        foreach (Match match in Regex.Matches(input, pattern))
            Console.WriteLine("Found {0} at index {1}.", match.Value, match.Index);

        Console.WriteLine();
        pattern = "(?i)\bthe\w*\b";
        foreach (Match match in Regex.Matches(input, pattern,
                                              RegexOptions.IgnoreCase))
            Console.WriteLine("Found {0} at index {1}.", match.Value, match.Index);
    }
}

// The example displays the following output:
//      Found The at index 0.
//      Found then at index 8.
//      Found them at index 18.
//
//      Found The at index 0.
//      Found then at index 8.
//      Found them at index 18.

```

```

Imports System.Text.RegularExpressions

Module CaseExample
    Public Sub Main()
        Dim pattern As String = "\b(?i:t)he\w*\b"
        Dim input As String = "The man then told them about that event."
        For Each match As Match In Regex.Matches(input, pattern)
            Console.WriteLine("Found {0} at index {1}.", match.Value, match.Index)
        Next
        Console.WriteLine()
        pattern = "(?i)\bthe\w*\b"
        For Each match As Match In Regex.Matches(input, pattern)
            Console.WriteLine("Found {0} at index {1}.", match.Value, match.Index)
        Next
    End Sub
End Module

' The example displays the following output:
'      Found The at index 0.
'      Found then at index 8.
'      Found them at index 18.
'
'      Found The at index 0.
'      Found then at index 8.
'      Found them at index 18.

```

Multiline mode

The [RegexOptions.Multiline](#) option, or the `m` inline option, enables the regular expression engine to handle an input string that consists of multiple lines. It changes the interpretation of the `^` and `$` language elements so that they match the beginning and end of a line, instead of the beginning and end of the input string.

By default, `$` matches only the end of the input string. If you specify the [RegexOptions.Multiline](#) option, it matches either the newline character (`\n`) or the end of the input string. It does not, however, match the

carriage return/line feed character combination. To successfully match them, use the subexpression `\r?$` instead of just `$`.

The following example extracts bowlers' names and scores and adds them to a `SortedList< TKey, TValue >` collection that sorts them in descending order. The `Matches` method is called twice. In the first method call, the regular expression is `^(\\w+)\\s(\\d+)$` and no options are set. As the output shows, because the regular expression engine cannot match the input pattern along with the beginning and end of the input string, no matches are found. In the second method call, the regular expression is changed to `^(\\w+)\\s(\\d+)\\r?$` and the options are set to `RegexOptions.Multiline`. As the output shows, the names and scores are successfully matched, and the scores are displayed in descending order.

```

using System;
using System.Collections.Generic;
using System.Text.RegularExpressions;

public class Multiline1Example
{
    public static void Main()
    {
        SortedList<int, string> scores = new SortedList<int, string>(new DescendingComparer1<int>());

        string input = "Joe 164\n" +
                      "Sam 208\n" +
                      "Allison 211\n" +
                      "Gwen 171\n";
        string pattern = @"^(\w+)\s(\d+)$";
        bool matched = false;

        Console.WriteLine("Without Multiline option:");
        foreach (Match match in Regex.Matches(input, pattern))
        {
            scores.Add(Int32.Parse(match.Groups[2].Value), (string)match.Groups[1].Value);
            matched = true;
        }
        if (!matched)
            Console.WriteLine("    No matches.");
        Console.WriteLine();

        // Redefine pattern to handle multiple lines.
        pattern = @"^(\w+)\s(\d+)\r*$";
        Console.WriteLine("With multiline option:");
        foreach (Match match in Regex.Matches(input, pattern, RegexOptions.Multiline))
            scores.Add(Int32.Parse(match.Groups[2].Value), (string)match.Groups[1].Value);

        // List scores in descending order.
        foreach (KeyValuePair<int, string> score in scores)
            Console.WriteLine("{0}: {1}", score.Value, score.Key);
    }
}

public class DescendingComparer1<T> : IComparer<T>
{
    public int Compare(T x, T y)
    {
        return Comparer<T>.Default.Compare(x, y) * -1;
    }
}
// The example displays the following output:
// Without Multiline option:
//     No matches.
//
// With multiline option:
// Allison: 211
// Sam: 208
// Gwen: 171
// Joe: 164

```

```

Imports System.Collections.Generic
Imports System.Text.RegularExpressions

Module Multiline1Example
    Public Sub Main()
        Dim scores As New SortedList(Of Integer, String)(New DescendingComparer1(Of Integer)())

        Dim input As String = "Joe 164" + vbCrLF +
                            "Sam 208" + vbCrLF +
                            "Allison 211" + vbCrLF +
                            "Gwen 171" + vbCrLF
        Dim pattern As String = "^(\w+)\s(\d+)$"
        Dim matched As Boolean = False

        Console.WriteLine("Without Multiline option:")
        For Each match As Match In Regex.Matches(input, pattern)
            scores.Add(CInt(match.Groups(2).Value), match.Groups(1).Value)
            matched = True
        Next
        If Not matched Then Console.WriteLine("    No matches.")
        Console.WriteLine()

        ' Redefine pattern to handle multiple lines.
        pattern = "^\w+\s\d+\r*$"
        Console.WriteLine("With multiline option:")
        For Each match As Match In Regex.Matches(input, pattern, RegexOptions.Multiline)
            scores.Add(CInt(match.Groups(2).Value), match.Groups(1).Value)
        Next
        ' List scores in descending order.
        For Each score As KeyValuePair(Of Integer, String) In scores
            Console.WriteLine("{0}: {1}", score.Value, score.Key)
        Next
    End Sub
End Module

Public Class DescendingComparer1(Of T) : Implements IComparer(Of T)
    Public Function Compare(x As T, y As T) As Integer _
        Implements IComparer(Of T).Compare
        Return Comparer(Of T).Default.Compare(x, y) * -1
    End Function
End Class
' The example displays the following output:
' Without Multiline option:
'     No matches.
'
' With multiline option:
' Allison: 211
' Sam: 208
' Gwen: 171
' Joe: 164

```

The regular expression pattern `^\w+\s\d+\r*$` is defined as shown in the following table.

PATTERN	DESCRIPTION
<code>^</code>	Begin at the start of the line.
<code>(\w+)</code>	Match one or more word characters. This is the first capturing group.
<code>\s</code>	Match a white-space character.

PATTERN	DESCRIPTION
(\d+)	Match one or more decimal digits. This is the second capturing group.
\r?	Match zero or one carriage return character.
\$	End at the end of the line.

The following example is equivalent to the previous one, except that it uses the inline option `(?m)` to set the multiline option.

```
using System;
using System.Collections.Generic;
using System.Text.RegularExpressions;

public class Multiline2Example
{
    public static void Main()
    {
        SortedList<int, string> scores = new SortedList<int, string>(new DescendingComparer<int>());

        string input = "Joe 164\n" +
                      "Sam 208\n" +
                      "Allison 211\n" +
                      "Gwen 171\n";
        string pattern = @"(?m)^(\w+)\s(\d+)\r*$";

        foreach (Match match in Regex.Matches(input, pattern, RegexOptions.Multiline))
            scores.Add(Convert.ToInt32(match.Groups[2].Value), match.Groups[1].Value);

        // List scores in descending order.
        foreach (KeyValuePair<int, string> score in scores)
            Console.WriteLine("{0}: {1}", score.Value, score.Key);
    }
}

public class DescendingComparer<T> : IComparer<T>
{
    public int Compare(T x, T y)
    {
        return Comparer<T>.Default.Compare(x, y) * -1;
    }
}
// The example displays the following output:
// Allison: 211
// Sam: 208
// Gwen: 171
// Joe: 164
```

```

Imports System.Collections.Generic
Imports System.Text.RegularExpressions

Module Multiline2Example
    Public Sub Main()
        Dim scores As New SortedList(Of Integer, String)(New DescendingComparer(Of Integer)())

        Dim input As String = "Joe 164" + vbCrLF +
                            "Sam 208" + vbCrLF +
                            "Allison 211" + vbCrLF +
                            "Gwen 171" + vbCrLF
        Dim pattern As String = "(?m)^(\w+)\s(\d+)\r*$"

        For Each match As Match In Regex.Matches(input, pattern, RegexOptions.Multiline)
            scores.Add(CInt(match.Groups(2).Value), match.Groups(1).Value)
        Next
        ' List scores in descending order.
        For Each score As KeyValuePair(Of Integer, String) In scores
            Console.WriteLine("{0}: {1}", score.Value, score.Key)
        Next
    End Sub
End Module

Public Class DescendingComparer(Of T) : Implements IComparer(Of T)
    Public Function Compare(x As T, y As T) As Integer _
        Implements IComparer(Of T).Compare
        Return Comparer(Of T).Default.Compare(x, y) * -1
    End Function
End Class
' The example displays the following output:
' Allison: 211
' Sam: 208
' Gwen: 171
' Joe: 164

```

Single-line mode

The [RegexOptions.Singleline](#) option, or the `s` inline option, causes the regular expression engine to treat the input string as if it consists of a single line. It does this by changing the behavior of the period (`.`) language element so that it matches every character, instead of matching every character except for the newline character `\n` or `\u000A`.

The `$` language element will match the end of the string or a trailing newline character `\n`.

The following example illustrates how the behavior of the `.` language element changes when you use the [RegexOptions.Singleline](#) option. The regular expression `^.+` starts at the beginning of the string and matches every character. By default, the match ends at the end of the first line; the regular expression pattern matches the carriage return character, `\r` or `\u000D`, but it does not match `\n`. Because the [RegexOptions.Singleline](#) option interprets the entire input string as a single line, it matches every character in the input string, including `\n`.

```

using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string pattern = "^.+";
        string input = "This is one line and" + Environment.NewLine + "this is the second.";
        foreach (Match match in Regex.Matches(input, pattern))
            Console.WriteLine(Regex.Escape(match.Value));

        Console.WriteLine();
        foreach (Match match in Regex.Matches(input, pattern, RegexOptions.Singleline))
            Console.WriteLine(Regex.Escape(match.Value));
    }
}

// The example displays the following output:
//      This\ is\ one\ line\ and\r
//
//      This\ is\ one\ line\ and\r\nthis\ is\ the\ second\.

```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim pattern As String = "^.+"
        Dim input As String = "This is one line and" + vbCrLf + "this is the second."
        For Each match As Match In Regex.Matches(input, pattern)
            Console.WriteLine(Regex.Escape(match.Value))
        Next
        Console.WriteLine()
        For Each match As Match In Regex.Matches(input, pattern, RegexOptions.SingleLine)
            Console.WriteLine(Regex.Escape(match.Value))
        Next
    End Sub
End Module
' The example displays the following output:
'      This\ is\ one\ line\ and\r
'
'      This\ is\ one\ line\ and\r\nthis\ is\ the\ second\.

```

The following example is equivalent to the previous one, except that it uses the inline option `(?s)` to enable single-line mode.

```

using System;
using System.Text.RegularExpressions;

public class SingleLineExample
{
    public static void Main()
    {
        string pattern = "(?s)^.+";
        string input = "This is one line and" + Environment.NewLine + "this is the second.";

        foreach (Match match in Regex.Matches(input, pattern))
            Console.WriteLine(Regex.Escape(match.Value));
    }
}

// The example displays the following output:
//      This\ is\ one\ line\ and\r\nthis\ is\ the\ second\.

```

```

Imports System.Text.RegularExpressions

Module SingleLineExample
    Public Sub Main()
        Dim pattern As String = "(?s)^.+"
        Dim input As String = "This is one line and" + vbCrLf + "this is the second."

        For Each match As Match In Regex.Matches(input, pattern)
            Console.WriteLine(Regex.Escape(match.Value))
        Next
    End Sub
End Module
' The example displays the following output:
' This\ is\ one\ line\ and\r\nthis\ is\ the\ second\.

```

Explicit captures only

By default, capturing groups are defined by the use of parentheses in the regular expression pattern. Named groups are assigned a name or number by the `(?<name>subexpression)` language option, whereas unnamed groups are accessible by index. In the [GroupCollection](#) object, unnamed groups precede named groups.

Grouping constructs are often used only to apply quantifiers to multiple language elements, and the captured substrings are of no interest. For example, if the following regular expression:

```
\b\((?((\w+),?\s?)+[\.\!?])\)?
```

is intended only to extract sentences that end with a period, exclamation point, or question mark from a document, only the resulting sentence (which is represented by the [Match](#) object) is of interest. The individual words in the collection are not.

Capturing groups that are not subsequently used can be expensive, because the regular expression engine must populate both the [GroupCollection](#) and [CaptureCollection](#) collection objects. As an alternative, you can use either the [RegexOptions.ExplicitCapture](#) option or the `n` inline option to specify that the only valid captures are explicitly named or numbered groups that are designated by the `(?<name>subexpression)` construct.

The following example displays information about the matches returned by the `\b\((?((\w+),?\s?)+[\.\!?])\)?` regular expression pattern when the [Match](#) method is called with and without the [RegexOptions.ExplicitCapture](#) option. As the output from the first method call shows, the regular expression engine fully populates the [GroupCollection](#) and [CaptureCollection](#) collection objects with information about captured substrings. Because the second method is called with `options` set to [RegexOptions.ExplicitCapture](#), it does not capture information on groups.

```

using System;
using System.Text.RegularExpressions;

public class Explicit1Example
{
    public static void Main()
    {
        string input = "This is the first sentence. Is it the beginning " +
                      "of a literary masterpiece? I think not. Instead, " +
                      "it is a nonsensical paragraph.";
        string pattern = @"\b\((?(>\w+),?\s?)+[\.\!?]\)\?";
        Console.WriteLine("With implicit captures:");
        foreach (Match match in Regex.Matches(input, pattern))
        {
            Console.WriteLine("The match: {0}", match.Value);
            int groupCtr = 0;
            foreach (Group group in match.Groups)

```

```

    {
        Console.WriteLine("  Group {0}: {1}", groupCtr, group.Value);
        groupCtr++;
        int captureCtr = 0;
        foreach (Capture capture in group.Captures)
        {
            Console.WriteLine("    Capture {0}: {1}", captureCtr, capture.Value);
            captureCtr++;
        }
    }
}
Console.WriteLine();
Console.WriteLine("With explicit captures only:");
foreach (Match match in Regex.Matches(input, pattern, RegexOptions.ExplicitCapture))
{
    Console.WriteLine("The match: {0}", match.Value);
    int groupCtr = 0;
    foreach (Group group in match.Groups)
    {
        Console.WriteLine("  Group {0}: {1}", groupCtr, group.Value);
        groupCtr++;
        int captureCtr = 0;
        foreach (Capture capture in group.Captures)
        {
            Console.WriteLine("    Capture {0}: {1}", captureCtr, capture.Value);
            captureCtr++;
        }
    }
}
}

// The example displays the following output:
//   With implicit captures:
//     The match: This is the first sentence.
//       Group 0: This is the first sentence.
//         Capture 0: This is the first sentence.
//       Group 1: sentence
//         Capture 0: This
//         Capture 1: is
//         Capture 2: the
//         Capture 3: first
//         Capture 4: sentence
//       Group 2: sentence
//         Capture 0: This
//         Capture 1: is
//         Capture 2: the
//         Capture 3: first
//         Capture 4: sentence
//     The match: Is it the beginning of a literary masterpiece?
//       Group 0: Is it the beginning of a literary masterpiece?
//         Capture 0: Is it the beginning of a literary masterpiece?
//       Group 1: masterpiece
//         Capture 0: Is
//         Capture 1: it
//         Capture 2: the
//         Capture 3: beginning
//         Capture 4: of
//         Capture 5: a
//         Capture 6: literary
//         Capture 7: masterpiece
//       Group 2: masterpiece
//         Capture 0: Is
//         Capture 1: it
//         Capture 2: the
//         Capture 3: beginning
//         Capture 4: of
//         Capture 5: a
//         Capture 6: literary
//         Capture 7: masterpiece

```

```

// The match: I think not.
// Group 0: I think not.
//     Capture 0: I think not.
// Group 1: not
//     Capture 0: I
//     Capture 1: think
//     Capture 2: not
// Group 2: not
//     Capture 0: I
//     Capture 1: think
//     Capture 2: not
// The match: Instead, it is a nonsensical paragraph.
// Group 0: Instead, it is a nonsensical paragraph.
//     Capture 0: Instead, it is a nonsensical paragraph.
// Group 1: paragraph
//     Capture 0: Instead,
//     Capture 1: it
//     Capture 2: is
//     Capture 3: a
//     Capture 4: nonsensical
//     Capture 5: paragraph
// Group 2: paragraph
//     Capture 0: Instead
//     Capture 1: it
//     Capture 2: is
//     Capture 3: a
//     Capture 4: nonsensical
//     Capture 5: paragraph
//
// With explicit captures only:
// The match: This is the first sentence.
// Group 0: This is the first sentence.
//     Capture 0: This is the first sentence.
// The match: Is it the beginning of a literary masterpiece?
// Group 0: Is it the beginning of a literary masterpiece?
//     Capture 0: Is it the beginning of a literary masterpiece?
// The match: I think not.
// Group 0: I think not.
//     Capture 0: I think not.
// The match: Instead, it is a nonsensical paragraph.
// Group 0: Instead, it is a nonsensical paragraph.
//     Capture 0: Instead, it is a nonsensical paragraph.

```

```

Imports System.Text.RegularExpressions

Module Explicit1Example
    Public Sub Main()
        Dim input As String = "This is the first sentence. Is it the beginning " +
                            "of a literary masterpiece? I think not. Instead, " +
                            "it is a nonsensical paragraph."
        Dim pattern As String = "\b\((?:\w+),?\s?)+[\.\!?\]\)\?"
        Console.WriteLine("With implicit captures:")
        For Each match As Match In Regex.Matches(input, pattern)
            Console.WriteLine("The match: {0}", match.Value)
            Dim groupCtr As Integer = 0
            For Each group As Group In match.Groups
                Console.WriteLine("    Group {0}: {1}", groupCtr, group.Value)
                groupCtr += 1
                Dim captureCtr As Integer = 0
                For Each capture As Capture In group.Captures
                    Console.WriteLine("        Capture {0}: {1}", captureCtr, capture.Value)
                    captureCtr += 1
                Next
            Next
        Next
        Console.WriteLine()
        Console.WriteLine("With explicit captures only:")

```

```

For Each match As Match In Regex.Matches(input, pattern, RegexOptions.ExplicitCapture)
    Console.WriteLine("The match: {0}", match.Value)
    Dim groupCtr As Integer = 0
    For Each group As Group In match.Groups
        Console.WriteLine("    Group {0}: {1}", groupCtr, group.Value)
        groupCtr += 1
        Dim captureCtr As Integer = 0
        For Each capture As Capture In group.Captures
            Console.WriteLine("        Capture {0}: {1}", captureCtr, capture.Value)
            captureCtr += 1
        Next
    Next
End Sub
End Module

' The example displays the following output:
' With implicit captures:
' The match: This is the first sentence.
'     Group 0: This is the first sentence.
'         Capture 0: This is the first sentence.
'             Group 1: sentence
'                 Capture 0: This
'                 Capture 1: is
'                 Capture 2: the
'                 Capture 3: first
'                 Capture 4: sentence
'             Group 2: sentence
'                 Capture 0: This
'                 Capture 1: is
'                 Capture 2: the
'                 Capture 3: first
'                 Capture 4: sentence
' The match: Is it the beginning of a literary masterpiece?
'     Group 0: Is it the beginning of a literary masterpiece?
'         Capture 0: Is it the beginning of a literary masterpiece?
'             Group 1: masterpiece
'                 Capture 0: Is
'                 Capture 1: it
'                 Capture 2: the
'                 Capture 3: beginning
'                 Capture 4: of
'                 Capture 5: a
'                 Capture 6: literary
'                 Capture 7: masterpiece
'             Group 2: masterpiece
'                 Capture 0: Is
'                 Capture 1: it
'                 Capture 2: the
'                 Capture 3: beginning
'                 Capture 4: of
'                 Capture 5: a
'                 Capture 6: literary
'                 Capture 7: masterpiece
' The match: I think not.
'     Group 0: I think not.
'         Capture 0: I think not.
'             Group 1: not
'                 Capture 0: I
'                 Capture 1: think
'                 Capture 2: not
'             Group 2: not
'                 Capture 0: I
'                 Capture 1: think
'                 Capture 2: not
' The match: Instead, it is a nonsensical paragraph.
'     Group 0: Instead, it is a nonsensical paragraph.
'         Capture 0: Instead, it is a nonsensical paragraph.
'             Group 1: paragraph
'                 Capture 0: Instead.

```

```

Capture 1: it
Capture 2: is
Capture 3: a
Capture 4: nonsensical
Capture 5: paragraph
Group 2: paragraph
    Capture 0: Instead
    Capture 1: it
    Capture 2: is
    Capture 3: a
    Capture 4: nonsensical
    Capture 5: paragraph

With explicit captures only:
The match: This is the first sentence.
    Group 0: This is the first sentence.
        Capture 0: This is the first sentence.
The match: Is it the beginning of a literary masterpiece?
    Group 0: Is it the beginning of a literary masterpiece?
        Capture 0: Is it the beginning of a literary masterpiece?
The match: I think not.
    Group 0: I think not.
        Capture 0: I think not.
The match: Instead, it is a nonsensical paragraph.
    Group 0: Instead, it is a nonsensical paragraph.
        Capture 0: Instead, it is a nonsensical paragraph.

```

The regular expression pattern `\b\((?((?>\w+),?\s?)+[\.\!?\])\)?` is defined as shown in the following table.

PATTERN	DESCRIPTION
<code>\b</code>	Begin at a word boundary.
<code>\(?</code>	Match zero or one occurrences of the opening parenthesis ("(").
<code>(?>\w+),?</code>	Match one or more word characters, followed by zero or one commas. Do not backtrack when matching word characters.
<code>\s?</code>	Match zero or one white-space characters.
<code>((\w+),?\s?)+</code>	Match the combination of one or more word characters, zero or one commas, and zero or one white-space characters one or more times.
<code>[\.\!?\]\)?</code>	Match any of the three punctuation symbols, followed by zero or one closing parentheses (")").

You can also use the `(?n)` inline element to suppress automatic captures. The following example modifies the previous regular expression pattern to use the `(?n)` inline element instead of the `RegexOptions.ExplicitCapture` option.

```
using System;
using System.Text.RegularExpressions;

public class Explicit2Example
{
    public static void Main()
    {
        string input = "This is the first sentence. Is it the beginning " +
                      "of a literary masterpiece? I think not. Instead, " +
                      "it is a nonsensical paragraph.";
        string pattern = @"(?n)\b\((?(>\w+),?\s?)+[.\!?\]\)\)?";

        foreach (Match match in Regex.Matches(input, pattern))
        {
            Console.WriteLine("The match: {0}", match.Value);
            int groupCtr = 0;
            foreach (Group group in match.Groups)
            {
                Console.WriteLine("    Group {0}: {1}", groupCtr, group.Value);
                groupCtr++;
                int captureCtr = 0;
                foreach (Capture capture in group.Captures)
                {
                    Console.WriteLine("        Capture {0}: {1}", captureCtr, capture.Value);
                    captureCtr++;
                }
            }
        }
    }
}

// The example displays the following output:
//     The match: This is the first sentence.
//         Group 0: This is the first sentence.
//             Capture 0: This is the first sentence.
//     The match: Is it the beginning of a literary masterpiece?
//         Group 0: Is it the beginning of a literary masterpiece?
//             Capture 0: Is it the beginning of a literary masterpiece?
//     The match: I think not.
//         Group 0: I think not.
//             Capture 0: I think not.
//     The match: Instead, it is a nonsensical paragraph.
//         Group 0: Instead, it is a nonsensical paragraph.
//             Capture 0: Instead, it is a nonsensical paragraph.
```

```

Imports System.Text.RegularExpressions

Module Explicit2Example
    Public Sub Main()
        Dim input As String = "This is the first sentence. Is it the beginning " +
                            "of a literary masterpiece? I think not. Instead, " +
                            "it is a nonsensical paragraph."
        Dim pattern As String = "(?n)\b\((?:\w+,?\s?)+[\.\!?\]\)\?"

        For Each match As Match In Regex.Matches(input, pattern)
            Console.WriteLine("The match: {0}", match.Value)
            Dim groupCtr As Integer = 0
            For Each group As Group In match.Groups
                Console.WriteLine("    Group {0}: {1}", groupCtr, group.Value)
                groupCtr += 1
                Dim captureCtr As Integer = 0
                For Each capture As Capture In group.Captures
                    Console.WriteLine("        Capture {0}: {1}", captureCtr, capture.Value)
                    captureCtr += 1
                Next
            Next
        Next
    End Sub
End Module
' The example displays the following output:
'   The match: This is the first sentence.
'       Group 0: This is the first sentence.
'           Capture 0: This is the first sentence.
'   The match: Is it the beginning of a literary masterpiece?
'       Group 0: Is it the beginning of a literary masterpiece?
'           Capture 0: Is it the beginning of a literary masterpiece?
'   The match: I think not.
'       Group 0: I think not.
'           Capture 0: I think not.
'   The match: Instead, it is a nonsensical paragraph.
'       Group 0: Instead, it is a nonsensical paragraph.
'           Capture 0: Instead, it is a nonsensical paragraph.

```

Finally, you can use the inline group element `(?n:)` to suppress automatic captures on a group-by-group basis. The following example modifies the previous pattern to suppress unnamed captures in the outer group, `((?:\w+,?\s?))`. Note that this suppresses unnamed captures in the inner group as well.

```
using System;
using System.Text.RegularExpressions;

public class Explicit3Example
{
    public static void Main()
    {
        string input = "This is the first sentence. Is it the beginning " +
                      "of a literary masterpiece? I think not. Instead, " +
                      "it is a nonsensical paragraph.";
        string pattern = @"\b\(?(\?n:(?>\w+),?\s?)+[\.\!?\]\)\?"';

        foreach (Match match in Regex.Matches(input, pattern))
        {
            Console.WriteLine("The match: {0}", match.Value);
            int groupCtr = 0;
            foreach (Group group in match.Groups)
            {
                Console.WriteLine("    Group {0}: {1}", groupCtr, group.Value);
                groupCtr++;
                int captureCtr = 0;
                foreach (Capture capture in group.Captures)
                {
                    Console.WriteLine("        Capture {0}: {1}", captureCtr, capture.Value);
                    captureCtr++;
                }
            }
        }
    }
}

// The example displays the following output:
//     The match: This is the first sentence.
//         Group 0: This is the first sentence.
//             Capture 0: This is the first sentence.
//     The match: Is it the beginning of a literary masterpiece?
//         Group 0: Is it the beginning of a literary masterpiece?
//             Capture 0: Is it the beginning of a literary masterpiece?
//     The match: I think not.
//         Group 0: I think not.
//             Capture 0: I think not.
//     The match: Instead, it is a nonsensical paragraph.
//         Group 0: Instead, it is a nonsensical paragraph.
//             Capture 0: Instead, it is a nonsensical paragraph.
```

```

Imports System.Text.RegularExpressions

Module Explicit3Example
    Public Sub Main()
        Dim input As String = "This is the first sentence. Is it the beginning " +
                            "of a literary masterpiece? I think not. Instead, " +
                            "it is a nonsensical paragraph."
        Dim pattern As String = "\b\((?:n:(?:>\w+),?\s?)?+[\.!?\]\)\?""

        For Each match As Match In Regex.Matches(input, pattern)
            Console.WriteLine("The match: {0}", match.Value)
            Dim groupCtr As Integer = 0
            For Each group As Group In match.Groups
                Console.WriteLine("    Group {0}: {1}", groupCtr, group.Value)
                groupCtr += 1
                Dim captureCtr As Integer = 0
                For Each capture As Capture In group.Captures
                    Console.WriteLine("        Capture {0}: {1}", captureCtr, capture.Value)
                    captureCtr += 1
                Next
            Next
        Next
    End Sub
End Module
' The example displays the following output:
'   The match: This is the first sentence.
'     Group 0: This is the first sentence.
'       Capture 0: This is the first sentence.
'   The match: Is it the beginning of a literary masterpiece?
'     Group 0: Is it the beginning of a literary masterpiece?
'       Capture 0: Is it the beginning of a literary masterpiece?
'   The match: I think not.
'     Group 0: I think not.
'       Capture 0: I think not.
'   The match: Instead, it is a nonsensical paragraph.
'     Group 0: Instead, it is a nonsensical paragraph.
'       Capture 0: Instead, it is a nonsensical paragraph.

```

Compiled regular expressions

By default, regular expressions in .NET are interpreted. When a [Regex](#) object is instantiated or a static [Regex](#) method is called, the regular expression pattern is parsed into a set of custom opcodes, and an interpreter uses these opcodes to run the regular expression. This involves a tradeoff: The cost of initializing the regular expression engine is minimized at the expense of run-time performance.

You can use compiled instead of interpreted regular expressions by using the [RegexOptions.Compiled](#) option. In this case, when a pattern is passed to the regular expression engine, it is parsed into a set of opcodes and then converted to Microsoft intermediate language (MSIL), which can be passed directly to the common language runtime. Compiled regular expressions maximize run-time performance at the expense of initialization time.

NOTE

A regular expression can be compiled only by supplying the [RegexOptions.Compiled](#) value to the `options` parameter of a [Regex](#) class constructor or a static pattern-matching method. It is not available as an inline option.

You can use compiled regular expressions in calls to both static and instance regular expressions. In static regular expressions, the [RegexOptions.Compiled](#) option is passed to the `options` parameter of the regular expression pattern-matching method. In instance regular expressions, it is passed to the `options` parameter of the [Regex](#) class constructor. In both cases, it results in enhanced performance.

However, this improvement in performance occurs only under the following conditions:

- A [Regex](#) object that represents a particular regular expression is used in multiple calls to regular expression pattern-matching methods.
- The [Regex](#) object is not allowed to go out of scope, so it can be reused.
- A static regular expression is used in multiple calls to regular expression pattern-matching methods. (The performance improvement is possible because regular expressions used in static method calls are cached by the regular expression engine.)

NOTE

The [RegexOptions.Compiled](#) option is unrelated to the [Regex.CompileToAssembly](#) method, which creates a special-purpose assembly that contains predefined compiled regular expressions.

Ignore white space

By default, white space in a regular expression pattern is significant; it forces the regular expression engine to match a white-space character in the input string. Because of this, the regular expression "`\b\w+\s`" and "`\b\w+`" are roughly equivalent regular expressions. In addition, when the number sign (#) is encountered in a regular expression pattern, it is interpreted as a literal character to be matched.

The [RegexOptions.IgnorePatternWhitespace](#) option, or the `x` inline option, changes this default behavior as follows:

- Unescaped white space in the regular expression pattern is ignored. To be part of a regular expression pattern, white-space characters must be escaped (for example, as `\s` or "`\` ").
- The number sign (#) is interpreted as the beginning of a comment, rather than as a literal character. All text in the regular expression pattern from the `#` character to either the next `\n` character or to the end of the string, is interpreted as a comment.

However, in the following cases, white-space characters in a regular expression aren't ignored, even if you use the [RegexOptions.IgnorePatternWhitespace](#) option:

- White space within a character class is always interpreted literally. For example, the regular expression pattern `[.;,:]` matches any single white-space character, period, comma, semicolon, or colon.
- White space isn't allowed within a bracketed quantifier, such as `{ n }`, `{ n ,}`, and `{ n , m }`. For example, the regular expression pattern `\d{1, 3}` fails to match any sequences of digits from one to three digits because it contains a white-space character.
- White space isn't allowed within a character sequence that introduces a language element. For example:
 - The language element `(?: subexpression)` represents a noncapturing group, and the `(?:` portion of the element can't have embedded spaces. The pattern `(? : subexpression)` throws an [ArgumentException](#) at run time because the regular expression engine can't parse the pattern, and the pattern `(?: subexpression)` fails to match `subexpression`.
 - The language element `\p{ name }`, which represents a Unicode category or named block, can't include embedded spaces in the `\p{` portion of the element. If you do include a white space, the element throws an [ArgumentException](#) at run time.

Enabling this option helps simplify regular expressions that are often difficult to parse and to understand. It improves readability, and makes it possible to document a regular expression.

The following example defines the following regular expression pattern:

```
\b \(? ( (?>\w+),?\s? )+ [\.\!?] \)? # Matches an entire sentence.
```

This pattern is similar to the pattern defined in the [Explicit Captures Only](#) section, except that it uses the `RegexOptions.IgnorePatternWhitespace` option to ignore pattern white space.

```
using System;
using System.Text.RegularExpressions;

public class Whitespace1Example
{
    public static void Main()
    {
        string input = "This is the first sentence. Is it the beginning " +
                      "of a literary masterpiece? I think not. Instead, " +
                      "it is a nonsensical paragraph.";
        string pattern = @"\b \(? ( (?>\w+),?\s? )+ [\.\!?] \)? # Matches an entire sentence./";

        foreach (Match match in Regex.Matches(input, pattern, RegexOptions.IgnorePatternWhitespace))
            Console.WriteLine(match.Value);
    }
}

// The example displays the following output:
//      This is the first sentence.
//      Is it the beginning of a literary masterpiece?
//      I think not.
//      Instead, it is a nonsensical paragraph.
```

```
Imports System.Text.RegularExpressions

Module Whitespace1Example
    Public Sub Main()
        Dim input As String = "This is the first sentence. Is it the beginning " +
                              "of a literary masterpiece? I think not. Instead, " +
                              "it is a nonsensical paragraph."
        Dim pattern As String = "\b \(? ( (?>\w+),?\s? )+ [\.\!?] \)? # Matches an entire sentence./"

        For Each match As Match In Regex.Matches(input, pattern, RegexOptions.IgnorePatternWhitespace)
            Console.WriteLine(match.Value)
        Next
    End Sub
End Module

' The example displays the following output:
'      This is the first sentence.
'      Is it the beginning of a literary masterpiece?
'      I think not.
'      Instead, it is a nonsensical paragraph.
```

The following example uses the inline option `(?x)` to ignore pattern white space.

```

using System;
using System.Text.RegularExpressions;

public class Whitespace2Example
{
    public static void Main()
    {
        string input = "This is the first sentence. Is it the beginning " +
                      "of a literary masterpiece? I think not. Instead, " +
                      "it is a nonsensical paragraph.";
        string pattern = @"(?x)\b \(? ( (?>\w+),?\s? )+ [\.\!?\] \)? # Matches an entire sentence./";

        foreach (Match match in Regex.Matches(input, pattern))
            Console.WriteLine(match.Value);
    }
}

// The example displays the following output:
//      This is the first sentence.
//      Is it the beginning of a literary masterpiece?
//      I think not.
//      Instead, it is a nonsensical paragraph.

```

```

Imports System.Text.RegularExpressions

Module Whitespace2Example
    Public Sub Main()
        Dim input As String = "This is the first sentence. Is it the beginning " +
                              "of a literary masterpiece? I think not. Instead, " +
                              "it is a nonsensical paragraph."
        Dim pattern As String = "(?x)\b \(? ( (?>\w+),?\s? )+ [\.\!?\] \)? # Matches an entire sentence./"

        For Each match As Match In Regex.Matches(input, pattern)
            Console.WriteLine(match.Value)
        Next
    End Sub
End Module
' The example displays the following output:
'      This is the first sentence.
'      Is it the beginning of a literary masterpiece?
'      I think not.
'      Instead, it is a nonsensical paragraph.

```

Right-to-left mode

By default, the regular expression engine searches from left to right. You can reverse the search direction by using the [RegexOptions.RightToLeft](#) option. The right-to-left search automatically begins at the last character position of the string. For pattern-matching methods that include a starting position parameter, such as [Regex.Match\(String, Int32\)](#), the specified starting position is the index of the rightmost character position at which the search is to begin.

NOTE

Right-to-left pattern mode is available only by supplying the [RegexOptions.RightToLeft](#) value to the `options` parameter of a [Regex](#) class constructor or static pattern-matching method. It is not available as an inline option.

Example

The regular expression `\bb\w+\s` matches words with two or more characters that begin with the letter "b" and are followed by a white-space character. In the following example, the input string consists of three words that include one or more "b" characters. The first and second words begin with "b" and the third word ends with "b".

As the output from the right-to-left search example shows, only the first and second words match the regular expression pattern, with the second word being matched first.

```
using System;
using System.Text.RegularExpressions;

public class RTL1Example
{
    public static void Main()
    {
        string pattern = @"\bb\b\w+\s";
        string input = "build band tab";
        foreach (Match match in Regex.Matches(input, pattern, RegexOptions.RightToLeft))
            Console.WriteLine("{0} found at position {1}.", match.Value, match.Index);
    }
}
// The example displays the following output:
//      'band ' found at position 6.
//      'build ' found at position 0.
```

```
Imports System.Text.RegularExpressions

Module RTL1Example
    Public Sub Main()
        Dim pattern As String = "\bb\b\w+\s"
        Dim input As String = "build band tab"
        For Each match As Match In Regex.Matches(input, pattern, RegexOptions.RightToLeft)
            Console.WriteLine("{0} found at position {1}.", match.Value, match.Index)
        Next
    End Sub
End Module
' The example displays the following output:
'      'band ' found at position 6.
'      'build ' found at position 0.
```

Evaluation order

The [RegexOptions.RightToLeft](#) option changes the search direction and also reverses the order in which the regular expression pattern is evaluated. In a right-to-left search, the search pattern is read from right to left. This distinction is important because it can affect things like capture groups and [backreferences](#). For example, the expression `Regex.Match("abcabc", @"\1(abc)", RegexOptions.RightToLeft)` finds a match `abcabc`, but in a left-to-right search (`Regex.Match("abcabc", @"\1(abc)", RegexOptions.None)`), no match is found. That's because the `(abc)` element must be evaluated before the numbered capturing group element (`\1`) for a match to be found.

Lookahead and lookbehind assertions

The location of a match for a lookahead (`(?=subexpression)`) or lookbehind (`(?<=subexpression)`) assertion doesn't change in a right-to-left search. The lookahead assertions look to the right of the current match location; the lookbehind assertions look to the left of the current match location.

TIP

Whether a search is right-to-left or not, lookbehinds are implemented using a right-to-left search starting at the current match location.

For example, the regular expression `(?<=\d{1,2}\s)\w+, \s\d{4}` uses the lookbehind assertion to test for a date that precedes a month name. The regular expression then matches the month and the year. For information on lookahead and lookbehind assertions, see [Grouping Constructs](#).

```

using System;
using System.Text.RegularExpressions;

public class RTL2Example
{
    public static void Main()
    {
        string[] inputs = { "1 May, 1917", "June 16, 2003" };
        string pattern = @"(?<=\d{1,2}\s)\w+, \s\d{4}";

        foreach (string input in inputs)
        {
            Match match = Regex.Match(input, pattern, RegexOptions.RightToLeft);
            if (match.Success)
                Console.WriteLine("The date occurs in {0}.", match.Value);
            else
                Console.WriteLine("{0} does not match.", input);
        }
    }
}

// The example displays the following output:
//      The date occurs in May, 1917.
//      June 16, 2003 does not match.

```

```

Imports System.Text.RegularExpressions

Module RTL2Example
    Public Sub Main()
        Dim inputs() As String = {"1 May, 1917", "June 16, 2003"}
        Dim pattern As String = "(?<=\d{1,2}\s)\w+, \s\d{4}"

        For Each input As String In inputs
            Dim match As Match = Regex.Match(input, pattern, RegexOptions.RightToLeft)
            If match.Success Then
                Console.WriteLine("The date occurs in {0}.", match.Value)
            Else
                Console.WriteLine("{0} does not match.", input)
            End If
        Next
    End Sub
End Module

' The example displays the following output:
'      The date occurs in May, 1917.
'      June 16, 2003 does not match.

```

The regular expression pattern is defined as shown in the following table.

PATTERN	DESCRIPTION
(?<=\d{1,2}\s)	The beginning of the match must be preceded by one or two decimal digits followed by a space.
\w+	Match one or more word characters.
,	Match one comma character.
\s	Match a white-space character.

PATTERN	DESCRIPTION
<code>\d{4}</code>	Match four decimal digits.

ECMAScript matching behavior

By default, the regular expression engine uses canonical behavior when matching a regular expression pattern to input text. However, you can instruct the regular expression engine to use ECMAScript matching behavior by specifying the [RegexOptions.EMAScript](#) option.

NOTE

ECMAScript-compliant behavior is available only by supplying the [RegexOptions.EMAScript](#) value to the `options` parameter of a [Regex](#) class constructor or static pattern-matching method. It is not available as an inline option.

The [RegexOptions.EMAScript](#) option can be combined only with the [RegexOptions.IgnoreCase](#) and [RegexOptions.Multiline](#) options. The use of any other option in a regular expression results in an [ArgumentOutOfRangeException](#).

The behavior of ECMAScript and canonical regular expressions differs in three areas: character class syntax, self-referencing capturing groups, and octal versus backreference interpretation.

- Character class syntax. Because canonical regular expressions support Unicode whereas ECMAScript does not, character classes in ECMAScript have a more limited syntax, and some character class language elements have a different meaning. For example, ECMAScript does not support language elements such as the Unicode category or block elements `\p` and `\P`. Similarly, the `\w` element, which matches a word character, is equivalent to the `[a-zA-Z_0-9]` character class when using ECMAScript and `[\p{L}\p{Lu}\p{Lt}\p{Lo}\p{Nd}\p{Pc}\p{Lm}]` when using canonical behavior. For more information, see [Character Classes](#).

The following example illustrates the difference between canonical and ECMAScript pattern matching. It defines a regular expression, `\b(\w+\s*)+`, that matches words followed by white-space characters. The input consists of two strings, one that uses the Latin character set and the other that uses the Cyrillic character set. As the output shows, the call to the [Regex.IsMatch\(String, String, RegexOptions\)](#) method that uses ECMAScript matching fails to match the Cyrillic words, whereas the method call that uses canonical matching does match these words.

```

using System;
using System.Text.RegularExpressions;

public class EcmaScriptExample
{
    public static void Main()
    {
        string[] values = { "целый мир", "the whole world" };
        string pattern = @"\b(\w+\s*)+";
        foreach (var value in values)
        {
            Console.Write("Canonical matching: ");
            if (Regex.IsMatch(value, pattern))
                Console.WriteLine("'{}' matches the pattern.", value);
            else
                Console.WriteLine("{} does not match the pattern.", value);

            Console.Write("ECMAScript matching: ");
            if (Regex.IsMatch(value, pattern, RegexOptions.ECMAScript))
                Console.WriteLine("'{}' matches the pattern.", value);
            else
                Console.WriteLine("{} does not match the pattern.", value);
            Console.WriteLine();
        }
    }
}

// The example displays the following output:
// Canonical matching: 'целый мир' matches the pattern.
// ECMAScript matching: целый мир does not match the pattern.
//
// Canonical matching: 'the whole world' matches the pattern.
// ECMAScript matching: 'the whole world' matches the pattern.

```

```

Imports System.Text.RegularExpressions

Module Ecma1Example
    Public Sub Main()
        Dim values() As String = {"целый мир", "the whole world"}
        Dim pattern As String = "\b(\w+\s*)+"
        For Each value In values
            Console.Write("Canonical matching: ")
            If Regex.IsMatch(value, pattern) Then
                Console.WriteLine("{} matches the pattern.", value)
            Else
                Console.WriteLine("{} does not match the pattern.", value)
            End If

            Console.Write("ECMAScript matching: ")
            If Regex.IsMatch(value, pattern, RegexOptions.ECMAScript) Then
                Console.WriteLine("{} matches the pattern.", value)
            Else
                Console.WriteLine("{} does not match the pattern.", value)
            End If
            Console.WriteLine()
        Next
    End Sub
End Module
' The example displays the following output:
' Canonical matching: 'целый мир' matches the pattern.
' ECMAScript matching: целый мир does not match the pattern.
'
' Canonical matching: 'the whole world' matches the pattern.
' ECMAScript matching: 'the whole world' matches the pattern.

```

- Self-referencing capturing groups. A regular expression capture class with a backreference to itself must

be updated with each capture iteration. As the following example shows, this feature enables the regular expression `((a+)(\1) ?)+` to match the input string " aa aaaa aaaaaa " when using ECMAScript, but not when using canonical matching.

```

using System;
using System.Text.RegularExpressions;

public class EcmaScript2Example
{
    static string pattern;

    public static void Main()
    {
        string input = "aa aaaa aaaaaa ";
        pattern = @"((a+)(\1) ?)+";

        // Match input using canonical matching.
        AnalyzeMatch(Regex.Match(input, pattern));

        // Match input using ECMAScript.
        AnalyzeMatch(Regex.Match(input, pattern, RegexOptions.ECMAScript));
    }

    private static void AnalyzeMatch(Match m)
    {
        if (m.Success)
        {
            Console.WriteLine("'{0}' matches {1} at position {2}.",
                pattern, m.Value, m.Index);
            int grpCtr = 0;
            foreach (Group grp in m.Groups)
            {
                Console.WriteLine("  {0}: '{1}'", grpCtr, grp.Value);
                grpCtr++;
                int capCtr = 0;
                foreach (Capture cap in grp.Captures)
                {
                    Console.WriteLine("    {0}: '{1}'", capCtr, cap.Value);
                    capCtr++;
                }
            }
        }
        else
        {
            Console.WriteLine("No match found.");
        }
        Console.WriteLine();
    }
}

// The example displays the following output:
// No match found.
//
// '((a+)(\1) ?)+' matches aa aaaa aaaaaa  at position 0.
//   0: 'aa aaaa aaaaaa '
//   0: 'aa aaaa aaaaaa '
//   1: 'aaaaaa '
//   0: 'aa '
//   1: 'aaa '
//   2: 'aaaaa '
//   2: 'aa'
//   0: 'aa'
//   1: 'aa'
//   2: 'aa'
//   3: 'aaa '
//   0: ''
//   1: 'aa '
//   2: 'aaaa '

```

```

Imports System.Text.RegularExpressions

Module Ecma2Example
    Dim pattern As String

    Public Sub Main()
        Dim input As String = "aa aaaa aaaaaa"
        pattern = "((a+)(\1) ?)+"

        ' Match input using canonical matching.
        AnalyzeMatch(Regex.Match(input, pattern))

        ' Match input using ECMAScript.
        AnalyzeMatch(Regex.Match(input, pattern, RegexOptions.ECMAScript))
    End Sub

    Private Sub AnalyzeMatch(m As Match)
        If m.Success Then
            Console.WriteLine("'{0}' matches {1} at position {2}.",
                pattern, m.Value, m.Index)
            Dim grpCtr As Integer = 0
            For Each grp As Group In m.Groups
                Console.WriteLine("  {0}: '{1}'", grpCtr, grp.Value)
                grpCtr += 1
                Dim capCtr As Integer = 0
                For Each cap As Capture In grp.Captures
                    Console.WriteLine("    {0}: '{1}'", capCtr, cap.Value)
                    capCtr += 1
                Next
            Next
        Else
            Console.WriteLine("No match found.")
        End If
        Console.WriteLine()
    End Sub
End Module
' The example displays the following output:
' No match found.

'
' '((a+)(\1) ?)+' matches aa aaaa aaaaaa  at position 0.
'   0: 'aa aaaa aaaaaa '
'     0: 'aa aaaa aaaaaa '
'   1: 'aaaaaa '
'     0: 'aa '
'     1: 'aaaa '
'     2: 'aaaaaa '
'   2: 'aa'
'     0: 'aa'
'     1: 'aa'
'     2: 'aa'
'   3: 'aaaa '
'     0: ''
'     1: 'aa '
'     2: 'aaaa '

```

The regular expression is defined as shown in the following table.

PATTERN	DESCRIPTION
(a+)	Match the letter "a" one or more times. This is the second capturing group.
(\1)	Match the substring captured by the first capturing group. This is the third capturing group.

PATTERN	DESCRIPTION
?	Match zero or one space characters.
((a+)(\1) ?)+	Match the pattern of one or more "a" characters followed by a string that matches the first capturing group followed by zero or one space characters one or more times. This is the first capturing group.

- Resolution of ambiguities between octal escapes and backreferences. The following table summarizes the differences in octal versus backreference interpretation by canonical and ECMAScript regular expressions.

REGULAR EXPRESSION	CANONICAL BEHAVIOR	ECMASCRIPT BEHAVIOR
\0 followed by 0 to 2 octal digits	Interpret as an octal. For example, \044 is always interpreted as an octal value and means "\$".	Same behavior.
\ followed by a digit from 1 to 9, followed by no additional decimal digits,	Interpret as a backreference. For example, \9 always means backreference 9, even if a ninth capturing group does not exist. If the capturing group does not exist, the regular expression parser throws an ArgumentException .	If a single decimal digit capturing group exists, backreference to that digit. Otherwise, interpret the value as a literal.
\ followed by a digit from 1 to 9, followed by additional decimal digits	Interpret the digits as a decimal value. If that capturing group exists, interpret the expression as a backreference. Otherwise, interpret the leading octal digits up to octal 377; that is, consider only the low 8 bits of the value. Interpret the remaining digits as literals. For example, in the expression \3000, if capturing group 300 exists, interpret as backreference 300; if capturing group 300 does not exist, interpret as octal 300 followed by 0.	Interpret as a backreference by converting as many digits as possible to a decimal value that can refer to a capture. If no digits can be converted, interpret as an octal by using the leading octal digits up to octal 377; interpret the remaining digits as literals.

Compare using the invariant culture

By default, when the regular expression engine performs case-insensitive comparisons, it uses the casing conventions of the current culture to determine equivalent uppercase and lowercase characters.

However, this behavior is undesirable for some types of comparisons, particularly when comparing user input to the names of system resources, such as passwords, files, or URLs. The following example illustrates such a scenario. The code is intended to block access to any resource whose URL is prefaced with FILE://. The regular expression attempts a case-insensitive match with the string by using the regular expression \$FILE:// .

However, when the current system culture is tr-TR (Turkish-Turkey), "I" is not the uppercase equivalent of "i". As a result, the call to the [Regex.IsMatch](#) method returns `false`, and access to the file is allowed.

```

CultureInfo defaultCulture = Thread.CurrentThread.CurrentCulture;
Thread.CurrentThread.CurrentCulture = new CultureInfo("tr-TR");

string input = "file:///c:/Documents.MyReport.doc";
string pattern = "FILE:///";

Console.WriteLine("Culture-sensitive matching ({0} culture)...",
                  Thread.CurrentThread.CurrentCulture.Name);
if (Regex.IsMatch(input, pattern, RegexOptions.IgnoreCase))
    Console.WriteLine("URLs that access files are not allowed.");
else
    Console.WriteLine("Access to {0} is allowed.", input);

Thread.CurrentThread.CurrentCulture = defaultCulture;
// The example displays the following output:
//      Culture-sensitive matching (tr-TR culture)...
//      Access to file:///c:/Documents.MyReport.doc is allowed.

```

```

Dim defaultCulture As CultureInfo = Thread.CurrentThread.CurrentCulture
Thread.CurrentThread.CurrentCulture = New CultureInfo("tr-TR")

Dim input As String = "file:///c:/Documents.MyReport.doc"
Dim pattern As String = "$FILE://"

Console.WriteLine("Culture-sensitive matching ({0} culture)...",
                  Thread.CurrentThread.CurrentCulture.Name)
If Regex.IsMatch(input, pattern, RegexOptions.IgnoreCase) Then
    Console.WriteLine("URLs that access files are not allowed.")
Else
    Console.WriteLine("Access to {0} is allowed.", input)
End If

Thread.CurrentThread.CurrentCulture = defaultCulture
' The example displays the following output:
'      Culture-sensitive matching (tr-TR culture)...
'      Access to file:///c:/Documents.MyReport.doc is allowed.

```

NOTE

For more information about string comparisons that are case-sensitive and that use the invariant culture, see [Best Practices for Using Strings](#).

Instead of using the case-insensitive comparisons of the current culture, you can specify the [RegexOptions.CultureInvariant](#) option to ignore cultural differences in language and to use the conventions of the invariant culture.

NOTE

Comparison using the invariant culture is available only by supplying the [RegexOptions.CultureInvariant](#) value to the `options` parameter of a [Regex](#) class constructor or static pattern-matching method. It is not available as an inline option.

The following example is identical to the previous example, except that the static [Regex.IsMatch\(String, String, RegexOptions\)](#) method is called with options that include [RegexOptions.CultureInvariant](#). Even when the current culture is set to Turkish (Turkey), the regular expression engine is able to successfully match "FILE" and "file" and block access to the file resource.

```

CultureInfo defaultCulture = Thread.CurrentThread.CurrentCulture;
Thread.CurrentThread.CurrentCulture = new CultureInfo("tr-TR");

string input = "file:///c:/Documents.MyReport.doc";
string pattern = "FILE:///";

Console.WriteLine("Culture-insensitive matching...");
if (Regex.IsMatch(input, pattern,
    RegexOptions.IgnoreCase | RegexOptions.CultureInvariant))
    Console.WriteLine("URLs that access files are not allowed.");
else
    Console.WriteLine("Access to {0} is allowed.", input);

Thread.CurrentThread.CurrentCulture = defaultCulture;
// The example displays the following output:
//     Culture-insensitive matching...
//     URLs that access files are not allowed.

```

```

Dim defaultCulture As CultureInfo = Thread.CurrentThread.CurrentCulture
Thread.CurrentThread.CurrentCulture = New CultureInfo("tr-TR")

Dim input As String = "file:///c:/Documents.MyReport.doc"
Dim pattern As String = "$FILE://"

Console.WriteLine("Culture-insensitive matching...")
If Regex.IsMatch(input, pattern,
    RegexOptions.IgnoreCase Or RegexOptions.CultureInvariant) Then
    Console.WriteLine("URLs that access files are not allowed.")
Else
    Console.WriteLine("Access to {0} is allowed.", input)
End If
Thread.CurrentThread.CurrentCulture = defaultCulture
' The example displays the following output:
'     Culture-insensitive matching...
'     URLs that access files are not allowed.

```

See also

- [Regular Expression Language - Quick Reference](#)

Miscellaneous Constructs in Regular Expressions

9/20/2022 • 7 minutes to read • [Edit Online](#)

Regular expressions in .NET include three miscellaneous language constructs. One lets you enable or disable particular matching options in the middle of a regular expression pattern. The remaining two let you include comments in a regular expression.

Inline Options

You can set or disable specific pattern matching options for part of a regular expression by using the syntax

```
(?imnsx-imnsx)
```

You list the options you want to enable after the question mark, and the options you want to disable after the minus sign. The following table describes each option. For more information about each option, see [Regular Expression Options](#).

OPTION	DESCRIPTION
i	Case-insensitive matching.
m	Multiline mode.
n	Explicit captures only. (Parentheses do not act as capturing groups.)
s	Single-line mode.
x	Ignore unescaped white space, and allow x-mode comments.

Any change in regular expression options defined by the `(?imnsx-imnsx)` construct remains in effect until the end of the enclosing group.

NOTE

The `(?imnsx-imnsx: subexpression)` grouping construct provides identical functionality for a subexpression. For more information, see [Grouping Constructs](#).

The following example uses the `i`, `n`, and `x` options to enable case insensitivity and explicit captures, and to ignore white space in the regular expression pattern in the middle of a regular expression.

```
using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string pattern;
        string input = "double dare double Double a Drooling dog The Dreaded Deep";

        pattern = @"\b(D\w+)\s(d\w+)\b";
        // Match pattern using default options.
        foreach (Match match in Regex.Matches(input, pattern))
        {
            Console.WriteLine(match.Value);
            if (match.Groups.Count > 1)
                for (int ctr = 1; ctr < match.Groups.Count; ctr++)
                    Console.WriteLine("  Group {0}: {1}", ctr, match.Groups[ctr].Value);
        }
        Console.WriteLine();

        // Change regular expression pattern to include options.
        pattern = @"\b(D\w+)(?ixn) \s (d\w+) \b";
        // Match new pattern with options.
        foreach (Match match in Regex.Matches(input, pattern))
        {
            Console.WriteLine(match.Value);
            if (match.Groups.Count > 1)
                for (int ctr = 1; ctr < match.Groups.Count; ctr++)
                    Console.WriteLine("  Group {0}: '{1}'", ctr, match.Groups[ctr].Value);
        }
    }
}

// The example displays the following output:
//      Drooling dog
//          Group 1: Drooling
//          Group 2: dog
//
//      Drooling dog
//          Group 1: 'Drooling'
//      Dreaded Deep
//          Group 1: 'Dreaded'
```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim pattern As String
        Dim input As String = "double dare double Double a Drooling dog The Dreaded Deep"

        pattern = "\b(D\w+)\s(d\w+)\b"
        ' Match pattern using default options.
        For Each match As Match In Regex.Matches(input, pattern)
            Console.WriteLine(match.Value)
            If match.Groups.Count > 1 Then
                For ctr As Integer = 1 To match.Groups.Count - 1
                    Console.WriteLine("    Group {0}: {1}", ctr, match.Groups(ctr).Value)
                Next
            End If
        Next
        Console.WriteLine()

        ' Change regular expression pattern to include options.
        pattern = "\b(D\w+)(?ixn) \s (d\w+) \b"
        ' Match new pattern with options.
        For Each match As Match In Regex.Matches(input, pattern)
            Console.WriteLine(match.Value)
            If match.Groups.Count > 1 Then
                For ctr As Integer = 1 To match.Groups.Count - 1
                    Console.WriteLine("    Group {0}: '{1}'", ctr, match.Groups(ctr).Value)
                Next
            End If
        Next
    End Sub
End Module
' The example displays the following output:
'     Drooling dog
'         Group 1: Drooling
'         Group 2: dog
'
'     Drooling dog
'         Group 1: 'Drooling'
'     Dreaded Deep
'         Group 1: 'Dreaded'

```

The example defines two regular expressions. The first, `\b(D\w+)\s(d\w+)\b`, matches two consecutive words that begin with an uppercase "D" and a lowercase "d". The second regular expression, `\b(D\w+)(?ixn) \s (d\w+) \b`, uses inline options to modify this pattern, as described in the following table. A comparison of the results confirms the effect of the `(?ixn)` construct.

PATTERN	DESCRIPTION
<code>\b</code>	Start at a word boundary.
<code>(D\w+)</code>	Match a capital "D" followed by one or more word characters. This is the first capture group.
<code>(?ixn)</code>	From this point on, make comparisons case-insensitive, make only explicit captures, and ignore white space in the regular expression pattern.
<code>\s</code>	Match a white-space character.

PATTERN	DESCRIPTION
(d\w+)	Match an uppercase or lowercase "d" followed by one or more word characters. This group is not captured because the <code>n</code> (explicit capture) option was enabled..
\b	Match a word boundary.

Inline Comment

The `(?# comment)` construct lets you include an inline comment in a regular expression. The regular expression engine does not use any part of the comment in pattern matching, although the comment is included in the string that is returned by the [Regex.ToString](#) method. The comment ends at the first closing parenthesis.

The following example repeats the first regular expression pattern from the example in the previous section. It adds two inline comments to the regular expression to indicate whether the comparison is case-sensitive. The regular expression pattern,

`\b((?# case-sensitive comparison)D\w+)\s(?ixn)((?#case-insensitive comparison)d\w+)\b`, is defined as follows.

PATTERN	DESCRIPTION
\b	Start at a word boundary.
(?# case-sensitive comparison)	A comment. It does not affect pattern-matching behavior.
(D\w+)	Match a capital "D" followed by one or more word characters. This is the first capturing group.
\s	Match a white-space character.
(?ixn)	From this point on, make comparisons case-insensitive, make only explicit captures, and ignore white space in the regular expression pattern.
(?#case-insensitive comparison)	A comment. It does not affect pattern-matching behavior.
(d\w+)	Match an uppercase or lowercase "d" followed by one or more word characters. This is the second capture group.
\b	Match a word boundary.

```

using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string pattern = @"\b((?# case-sensitive comparison)D\w+)\s(?ixn)((?#case-insensitive
comparison)d\w+)\b";
        Regex rgx = new Regex(pattern);
        string input = "double dare double Double a Drooling dog The Dreaded Deep";

        Console.WriteLine("Pattern: " + pattern.ToString());
        // Match pattern using default options.
        foreach (Match match in rgx.Matches(input))
        {
            Console.WriteLine(match.Value);
            if (match.Groups.Count > 1)
            {
                for (int ctr = 1; ctr < match.Groups.Count; ctr++)
                    Console.WriteLine("    Group {0}: {1}", ctr, match.Groups[ctr].Value);
            }
        }
    }
}

// The example displays the following output:
// Pattern: \b((?# case-sensitive comparison)D\w+)\s(?ixn)((?#case-insensitive comp
// arison)d\w+)\b
// Drooling dog
//     Group 1: Drooling
//     Dreaded Deep
//     Group 1: Dreaded

```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim pattern As String = "\b((?# case-sensitive comparison)D\w+)\s(?ixn)((?#case-insensitive
comparison)d\w+)\b"
        Dim rgx As New Regex(pattern)
        Dim input As String = "double dare double Double a Drooling dog The Dreaded Deep"

        Console.WriteLine("Pattern: " + pattern.ToString())
        ' Match pattern using default options.
        For Each match As Match In rgx.Matches(input)
            Console.WriteLine(match.Value)
            If match.Groups.Count > 1 Then
                For ctr As Integer = 1 To match.Groups.Count - 1
                    Console.WriteLine("    Group {0}: {1}", ctr, match.Groups(ctr).Value)
                Next
            End If
        Next
    End Sub
End Module
' The example displays the following output:
' Pattern: \b((?# case-sensitive comparison)D\w+)\s(?ixn)((?#case-insensitive comp
' arison)d\w+)\b
' Drooling dog
'     Group 1: Drooling
'     Dreaded Deep
'     Group 1: Dreaded

```

End-of-Line Comment

A number sign (#) marks an x-mode comment, which starts at the unescaped # character at the end of the regular expression pattern and continues until the end of the line. To use this construct, you must either enable the `x` option (through inline options) or supply the `RegexOptions.IgnorePatternWhitespace` value to the `option` parameter when instantiating the `Regex` object or calling a static `Regex` method.

The following example illustrates the end-of-line comment construct. It determines whether a string is a composite format string that includes at least one format item. The following table describes the constructs in the regular expression pattern:

```
\{ \d+ (,-*\d+)* (\:\w{1,4}?) * \} (?x) # Looks for a composite format item.
```

PATTERN	DESCRIPTION
\{	Match an opening brace.
\d+	Match one or more decimal digits.
(,-*\d+)*	Match zero or one occurrence of a comma, followed by an optional minus sign, followed by one or more decimal digits.
(\:\w{1,4}?)*	Match zero or one occurrence of a colon, followed by one to four, but as few as possible, white-space characters.
\}	Match a closing brace.
(?x)	Enable the ignore pattern white-space option so that the end-of-line comment will be recognized.
# Looks for a composite format item.	An end-of-line comment.

```
using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string pattern = @"\{ \d+ (,-*\d+)* (\:\w{1,4}?) * \} (?x) # Looks for a composite format item.";
        string input = "{0,-3:F}";
        Console.WriteLine('{0}:', input);
        if (Regex.IsMatch(input, pattern))
            Console.WriteLine(" contains a composite format item.");
        else
            Console.WriteLine(" does not contain a composite format item.");
    }
}
// The example displays the following output:
//      '{0,-3:F}':
//          contains a composite format item.
```

```
Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim pattern As String = "\{\d+(-*\d+)*(\:\w{1,4}?)*\}(?x) # Looks for a composite format item."
        Dim input As String = "{0,-3:F}"
        Console.WriteLine("'{0}':", input)
        If Regex.IsMatch(input, pattern) Then
            Console.WriteLine("    contains a composite format item.")
        Else
            Console.WriteLine("    does not contain a composite format item.")
        End If
    End Sub
End Module
' The example displays the following output:
'{0,-3:F}':
contains a composite format item.
```

Note that, instead of providing the `(?x)` construct in the regular expression, the comment could also have been recognized by calling the [Regex.IsMatch\(String, String, RegexOptions\)](#) method and passing it the [RegexOptions.IgnorePatternWhitespace](#) enumeration value.

See also

- [Regular Expression Language - Quick Reference](#)

Best practices for regular expressions in .NET

9/20/2022 • 39 minutes to read • [Edit Online](#)

The regular expression engine in .NET is a powerful, full-featured tool that processes text based on pattern matches rather than on comparing and matching literal text. In most cases, it performs pattern matching rapidly and efficiently. However, in some cases, the regular expression engine can appear to be very slow. In extreme cases, it can even appear to stop responding as it processes a relatively small input over the course of hours or even days.

This topic outlines some of the best practices that developers can adopt to ensure that their regular expressions achieve optimal performance.

WARNING

When using `System.Text.RegularExpressions` to process untrusted input, pass a timeout. A malicious user can provide input to `RegularExpressions` causing a Denial-of-Service attack. ASP.NET Core framework APIs that use `RegularExpressions` pass a timeout.

Consider the input source

In general, regular expressions can accept two types of input: constrained or unconstrained. Constrained input is text that originates from a known or reliable source and follows a predefined format. Unconstrained input is text that originates from an unreliable source, such as a web user, and may not follow a predefined or expected format.

Regular expression patterns are typically written to match valid input. That is, developers examine the text that they want to match and then write a regular expression pattern that matches it. Developers then determine whether this pattern requires correction or further elaboration by testing it with multiple valid input items. When the pattern matches all presumed valid inputs, it is declared to be production-ready and can be included in a released application. This makes a regular expression pattern suitable for matching constrained input. However, it does not make it suitable for matching unconstrained input.

To match unconstrained input, a regular expression must be able to efficiently handle three kinds of text:

- Text that matches the regular expression pattern.
- Text that does not match the regular expression pattern.
- Text that nearly matches the regular expression pattern.

The last text type is especially problematic for a regular expression that has been written to handle constrained input. If that regular expression also relies on extensive [backtracking](#), the regular expression engine can spend an inordinate amount of time (in some cases, many hours or days) processing seemingly innocuous text.

WARNING

The following example uses a regular expression that is prone to excessive backtracking and that is likely to reject valid email addresses. You should not use it in an email validation routine. If you would like a regular expression that validates email addresses, see [How to: Verify that Strings Are in Valid Email Format](#).

For example, consider a very commonly used but extremely problematic regular expression for validating the

alias of an email address. The regular expression `^[0-9A-Z]([-.\\w]*[0-9A-Z])*$` is written to process what is considered to be a valid email address, which consists of an alphanumeric character, followed by zero or more characters that can be alphanumeric, periods, or hyphens. The regular expression must end with an alphanumeric character. However, as the following example shows, although this regular expression handles valid input easily, its performance is very inefficient when it is processing nearly valid input.

```
using System;
using System.Diagnostics;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        Stopwatch sw;
        string[] addresses = { "AAAAAAAAAAA@contoso.com",
                               "AAAAAAAAAAaaaaaaaaaa!@contoso.com" };
        // The following regular expression should not actually be used to
        // validate an email address.
        string pattern = @"^[0-9A-Z]([-.\\w]*[0-9A-Z])*$";
        string input;

        foreach (var address in addresses) {
            string mailbox = address.Substring(0, address.IndexOf("@"));
            int index = 0;
            for (int ctr = mailbox.Length - 1; ctr >= 0; ctr--) {
                index++;

                input = mailbox.Substring(ctr, index);
                sw = Stopwatch.StartNew();
                Match m = Regex.Match(input, pattern, RegexOptions.IgnoreCase);
                sw.Stop();
                if (m.Success)
                    Console.WriteLine("{0,2}. Matched '{1,25}' in {2}",
                                     index, m.Value, sw.Elapsed);
                else
                    Console.WriteLine("{0,2}. Failed  '{1,25}' in {2}",
                                     index, input, sw.Elapsed);
            }
            Console.WriteLine();
        }
    }

    // The example displays output similar to the following:
    //   1. Matched '          A' in 00:00:00.0007122
    //   2. Matched '          AA' in 00:00:00.0000282
    //   3. Matched '          AAA' in 00:00:00.0000042
    //   4. Matched '          AAAA' in 00:00:00.0000038
    //   5. Matched '          AAAAA' in 00:00:00.0000042
    //   6. Matched '          AAAAAA' in 00:00:00.0000042
    //   7. Matched '          AAAAAAA' in 00:00:00.0000042
    //   8. Matched '          AAAAAAAA' in 00:00:00.0000087
    //   9. Matched '          AAAAAAAA' in 00:00:00.0000045
    //  10. Matched '          AAAAAAAA' in 00:00:00.0000045
    //  11. Matched '          AAAAAAAA' in 00:00:00.0000045
    //
    //   1. Failed  '          !' in 00:00:00.0000447
    //   2. Failed  '          a!' in 00:00:00.0000071
    //   3. Failed  '          aa!' in 00:00:00.0000071
    //   4. Failed  '          aaa!' in 00:00:00.0000061
    //   5. Failed  '          aaaa!' in 00:00:00.0000081
    //   6. Failed  '          aaaaa!' in 00:00:00.0000126
    //   7. Failed  '          aaaaaa!' in 00:00:00.0000359
    //   8. Failed  '          aaaaaaa!' in 00:00:00.0000414
    //   9. Failed  '          aaaaaaaaa!' in 00:00:00.0000758
    //  10. Failed  '          aaaaaaaaaa!' in 00:00:00.0001162
```

```

// 10. Failed          aaaaaaaaaaaa: in 00:00:00.0001462
// 11. Failed          aaaaaaaaaaaa!' in 00:00:00.0002885
// 12. Failed          Aaaaaaaaaaaa!' in 00:00:00.0005780
// 13. Failed          AAaaaaaaaaaa!' in 00:00:00.0011628
// 14. Failed          AAAaaaaaaaaaa!' in 00:00:00.0022851
// 15. Failed          AAAAaaaaaaaaaa!' in 00:00:00.0045864
// 16. Failed          AAAAAAaaaaaaaaa!' in 00:00:00.0093168
// 17. Failed          AAAAAAAaaaaaaaaaa!' in 00:00:00.0185993
// 18. Failed          AAAAAAAAaaaaaaaaaa!' in 00:00:00.0366723
// 19. Failed          AAAAAAAAaaaaaaaaaa!' in 00:00:00.1370108
// 20. Failed          AAAAAAAAaaaaaaaaaa!' in 00:00:00.1553966
// 21. Failed          AAAAAAAAaaaaaaaaaa!' in 00:00:00.3223372

```

```

Imports System.Diagnostics
Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim sw As Stopwatch
        Dim addresses() As String =
            {"AAAAAAAAAAAA@contoso.com",
             "AAAAAAAAAAAaaaaaaaaa@contoso.com"}
        ' The following regular expression should not actually be used to
        ' validate an email address.
        Dim pattern As String = "[^0-9A-Z]([-.\w]*[0-9A-Z])*$"
        Dim input As String

        For Each address In addresses
            Dim mailbox As String = address.Substring(0, address.IndexOf("@"))
            Dim index As Integer = 0
            For ctr As Integer = mailbox.Length - 1 To 0 Step -1
                index += 1
                input = mailbox.Substring(ctr, index)
                sw = Stopwatch.StartNew()
                Dim m As Match = Regex.Match(input, pattern, RegexOptions.IgnoreCase)
                sw.Stop()
                If m.Success Then
                    Console.WriteLine("{0,2}. Matched '{1,25}' in {2}",
                                      index, m.Value, sw.Elapsed)
                Else
                    Console.WriteLine("{0,2}. Failed  '{1,25}' in {2}",
                                      index, input, sw.Elapsed)
                End If
            Next
            Console.WriteLine()
        Next
    End Sub
End Module
' The example displays output similar to the following:
' 1. Matched          A' in 00:00:00.0007122
' 2. Matched          AA' in 00:00:00.0000282
' 3. Matched          AAA' in 00:00:00.0000042
' 4. Matched          AAAA' in 00:00:00.0000038
' 5. Matched          AAAAA' in 00:00:00.0000042
' 6. Matched          AAAAAA' in 00:00:00.0000042
' 7. Matched          AAAAAAA' in 00:00:00.0000042
' 8. Matched          AAAAAAAA' in 00:00:00.0000087
' 9. Matched          AAAAAAAA' in 00:00:00.0000045
' 10. Matched         AAAAAAAA' in 00:00:00.0000045
' 11. Matched         AAAAAAAA' in 00:00:00.0000045
'
' 1. Failed           !' in 00:00:00.0000447
' 2. Failed           a!' in 00:00:00.0000071
' 3. Failed           aa!' in 00:00:00.0000071
' 4. Failed           aaa!' in 00:00:00.0000061
' 5. Failed           aaaa!' in 00:00:00.0000081
' 6. Failed           aaaaa!' in 00:00:00.0000126
' 7. Failed           aaaaaa!' in 00:00:00.0000359
' 8. Failed           aaaaaaa!' in 00:00:00.0000414

```

```

' 9. Failed      aaaaaaaaa! in 00:00:00.0000758
' 10. Failed     aaaaaaaaaa! in 00:00:00.0001462
' 11. Failed     aaaaaaaaaaa! in 00:00:00.0002885
' 12. Failed     Aaaaaaaaaaaa! in 00:00:00.0005780
' 13. Failed     Aaaaaaaaaaaaa! in 00:00:00.0011628
' 14. Failed     AAAaaaaaaaaaa! in 00:00:00.0022851
' 15. Failed     AAAaaaaaaaaaaa! in 00:00:00.0045864
' 16. Failed     AAAAaaaaaaaaaa! in 00:00:00.0093168
' 17. Failed     AAAAAAaaaaaaaaaa! in 00:00:00.0185993
' 18. Failed     AAAAAAaaaaaaaaaaa! in 00:00:00.0366723
' 19. Failed     AAAAAAaaaaaaaaaaaa! in 00:00:00.1370108
' 20. Failed     AAAAAAaaaaaaaaaaaaa! in 00:00:00.1553966
' 21. Failed     AAAAAAaaaaaaaaaaaaaa! in 00:00:00.3223372

```

As the output from the example shows, the regular expression engine processes the valid email alias in about the same time interval regardless of its length. On the other hand, when the nearly valid email address has more than five characters, processing time approximately doubles for each additional character in the string. This means that a nearly valid 28-character string would take over an hour to process, and a nearly valid 33-character string would take nearly a day to process.

Because this regular expression was developed solely by considering the format of input to be matched, it fails to take account of input that does not match the pattern. This, in turn, can allow unconstrained input that nearly matches the regular expression pattern to significantly degrade performance.

To solve this problem, you can do the following:

- When developing a pattern, you should consider how backtracking might affect the performance of the regular expression engine, particularly if your regular expression is designed to process unconstrained input. For more information, see the [Take Charge of Backtracking](#) section.
- Thoroughly test your regular expression using invalid and near-valid input as well as valid input. To generate input for a particular regular expression randomly, you can use [Rex](#), which is a regular expression exploration tool from Microsoft Research.

Handle object instantiation appropriately

At the heart of .NET's regular expression object model is the [System.Text.RegularExpressions.Regex](#) class, which represents the regular expression engine. Often, the single greatest factor that affects regular expression performance is the way in which the [Regex](#) engine is used. Defining a regular expression involves tightly coupling the regular expression engine with a regular expression pattern. That coupling process, whether it involves instantiating a [Regex](#) object by passing its constructor a regular expression pattern or calling a static method by passing it the regular expression pattern along with the string to be analyzed, is by necessity an expensive one.

NOTE

For a more detailed discussion of the performance implications of using interpreted and compiled regular expressions, see [Optimizing Regular Expression Performance, Part II: Taking Charge of Backtracking](#) in the BCL Team blog.

You can couple the regular expression engine with a particular regular expression pattern and then use the engine to match text in several ways:

- You can call a static pattern-matching method, such as [Regex.Match\(String, String\)](#). This does not require instantiation of a regular expression object.
- You can instantiate a [Regex](#) object and call an instance pattern-matching method of an interpreted regular expression. This is the default method for binding the regular expression engine to a regular expression

pattern. It results when a `Regex` object is instantiated without an `options` argument that includes the `Compiled` flag.

- You can instantiate a `Regex` object and call an instance pattern-matching method of a compiled regular expression. Regular expression objects represent compiled patterns when a `Regex` object is instantiated with an `options` argument that includes the `Compiled` flag.
- You can create a special-purpose `Regex` object that is tightly coupled with a particular regular expression pattern, compile it, and save it to a standalone assembly. You do this by calling the `Regex.CompileToAssembly` method.

The particular way in which you call regular expression matching methods can have a significant impact on your application. The following sections discuss when to use static method calls, interpreted regular expressions, and compiled regular expressions to improve your application's performance.

IMPORTANT

The form of the method call (static, interpreted, compiled) affects performance if the same regular expression is used repeatedly in method calls, or if an application makes extensive use of regular expression objects.

Static regular expressions

Static regular expression methods are recommended as an alternative to repeatedly instantiating a regular expression object with the same regular expression. Unlike regular expression patterns used by regular expression objects, either the operation codes or the compiled Microsoft intermediate language (MSIL) from patterns used in static method calls is cached internally by the regular expression engine.

For example, an event handler frequently calls another method to validate user input. This is reflected in the following code, in which a `Button` control's `Click` event is used to call a method named `IsValidCurrency`, which checks whether the user has entered a currency symbol followed by at least one decimal digit.

```
public void OKButton_Click(object sender, EventArgs e)
{
    if (! String.IsNullOrEmpty(sourceCurrency.Text))
        if (RegexLib.IsValidCurrency(sourceCurrency.Text))
            PerformConversion();
        else
            status.Text = "The source currency value is invalid.";
}
```

```
Public Sub OKButton_Click(sender As Object, e As EventArgs) _
    Handles OKButton.Click

    If Not String.IsNullOrEmpty(sourceCurrency.Text) Then
        If RegexLib.IsValidCurrency(sourceCurrency.Text) Then
            PerformConversion()
        Else
            status.Text = "The source currency value is invalid."
        End If
    End If
End Sub
```

A very inefficient implementation of the `IsValidCurrency` method is shown in the following example. Note that each method call reinstatiates a `Regex` object with the same pattern. This, in turn, means that the regular expression pattern must be recompiled each time the method is called.

```

using System;
using System.Text.RegularExpressions;

public class RegexLib
{
    public static bool IsValidCurrency(string currencyValue)
    {
        string pattern = @"\p{Sc}+\s*\d+";
        Regex currencyRegex = new Regex(pattern);
        return currencyRegex.IsMatch(currencyValue);
    }
}

```

```

Imports System.Text.RegularExpressions

Public Module RegexLib
    Public Function IsValidCurrency(currencyValue As String) As Boolean
        Dim pattern As String = "\p{Sc}+\s*\d+"
        Dim currencyRegex As New Regex(pattern)
        Return currencyRegex.IsMatch(currencyValue)
    End Function
End Module

```

You should replace this inefficient code with a call to the static [Regex.IsMatch\(String, String\)](#) method. This eliminates the need to instantiate a [Regex](#) object each time you want to call a pattern-matching method, and enables the regular expression engine to retrieve a compiled version of the regular expression from its cache.

```

using System;
using System.Text.RegularExpressions;

public class RegexLib
{
    public static bool IsValidCurrency(string currencyValue)
    {
        string pattern = @"\p{Sc}+\s*\d+";
        return Regex.IsMatch(currencyValue, pattern);
    }
}

```

```

Imports System.Text.RegularExpressions

Public Module RegexLib
    Public Function IsValidCurrency(currencyValue As String) As Boolean
        Dim pattern As String = "\p{Sc}+\s*\d+"
        Return Regex.IsMatch(currencyValue, pattern)
    End Function
End Module

```

By default, the last 15 most recently used static regular expression patterns are cached. For applications that require a larger number of cached static regular expressions, the size of the cache can be adjusted by setting the [Regex.CacheSize](#) property.

The regular expression `\p{Sc}+\s*\d+` that is used in this example verifies that the input string consists of a currency symbol and at least one decimal digit. The pattern is defined as shown in the following table.

PATTERN	DESCRIPTION
---------	-------------

PATTERN	DESCRIPTION
\p{Sc}+	Match one or more characters in the Unicode Symbol, Currency category.
\s*	Match zero or more white-space characters.
\d+	Match one or more decimal digits.

Interpreted vs. compiled regular expressions

Regular expression patterns that are not bound to the regular expression engine through the specification of the [Compiled](#) option are interpreted. When a regular expression object is instantiated, the regular expression engine converts the regular expression to a set of operation codes. When an instance method is called, the operation codes are converted to MSIL and executed by the JIT compiler. Similarly, when a static regular expression method is called and the regular expression cannot be found in the cache, the regular expression engine converts the regular expression to a set of operation codes and stores them in the cache. It then converts these operation codes to MSIL so that the JIT compiler can execute them. Interpreted regular expressions reduce startup time at the cost of slower execution time. Because of this, they are best used when the regular expression is used in a small number of method calls, or if the exact number of calls to regular expression methods is unknown but is expected to be small. As the number of method calls increases, the performance gain from reduced startup time is outstripped by the slower execution speed.

Regular expression patterns that are bound to the regular expression engine through the specification of the [Compiled](#) option are compiled. This means that, when a regular expression object is instantiated, or when a static regular expression method is called and the regular expression cannot be found in the cache, the regular expression engine converts the regular expression to an intermediary set of operation codes, which it then converts to MSIL. When a method is called, the JIT compiler executes the MSIL. In contrast to interpreted regular expressions, compiled regular expressions increase startup time but execute individual pattern-matching methods faster. As a result, the performance benefit that results from compiling the regular expression increases in proportion to the number of regular expression methods called.

To summarize, we recommend that you use interpreted regular expressions when you call regular expression methods with a specific regular expression relatively infrequently. You should use compiled regular expressions when you call regular expression methods with a specific regular expression relatively frequently. The exact threshold at which the slower execution speeds of interpreted regular expressions outweigh gains from their reduced startup time, or the threshold at which the slower startup times of compiled regular expressions outweigh gains from their faster execution speeds, is difficult to determine. It depends on a variety of factors, including the complexity of the regular expression and the specific data that it processes. To determine whether interpreted or compiled regular expressions offer the best performance for your particular application scenario, you can use the [Stopwatch](#) class to compare their execution times.

The following example compares the performance of compiled and interpreted regular expressions when reading the first ten sentences and when reading all the sentences in the text of Theodore Dreiser's *The Financier*. As the output from the example shows, when only ten calls are made to regular expression matching methods, an interpreted regular expression offers better performance than a compiled regular expression. However, a compiled regular expression offers better performance when a large number of calls (in this case, over 13,000) are made.

```
using System;
using System.Diagnostics;
using System.IO;
using System.Text.RegularExpressions;

public class Example
{
    static void Main()
    {
        string text = File.ReadAllText("TheFinancier.txt");
        Regex regex = new Regex(@"\b[a-zA-Z]+\b");
        MatchCollection matches = regex.Matches(text);
        Console.WriteLine("Number of matches: " + matches.Count);
    }
}
```

```

    public static void Main()
    {
        string pattern = @"\b(\w+((\r?\n)|,\?|\s))*\w+[.:;!]";
        Stopwatch sw;
        Match match;
        int ctr;

        StreamReader inFile = new StreamReader(@"..\Dreiser_TheFinancier.txt");
        string input = inFile.ReadToEnd();
        inFile.Close();

        // Read first ten sentences with interpreted regex.
        Console.WriteLine("10 Sentences with Interpreted Regex:");
        sw = Stopwatch.StartNew();
        Regex int10 = new Regex(pattern, RegexOptions.Singleline);
        match = int10.Match(input);
        for (ctr = 0; ctr <= 9; ctr++) {
            if (match.Success)
                // Do nothing with the match except get the next match.
                match = match.NextMatch();
            else
                break;
        }
        sw.Stop();
        Console.WriteLine(" {0} matches in {1}", ctr, sw.Elapsed);

        // Read first ten sentences with compiled regex.
        Console.WriteLine("10 Sentences with Compiled Regex:");
        sw = Stopwatch.StartNew();
        Regex comp10 = new Regex(pattern,
                               RegexOptions.Singleline | RegexOptions.Compiled);
        match = comp10.Match(input);
        for (ctr = 0; ctr <= 9; ctr++) {
            if (match.Success)
                // Do nothing with the match except get the next match.
                match = match.NextMatch();
            else
                break;
        }
        sw.Stop();
        Console.WriteLine(" {0} matches in {1}", ctr, sw.Elapsed);

        // Read all sentences with interpreted regex.
        Console.WriteLine("All Sentences with Interpreted Regex:");
        sw = Stopwatch.StartNew();
        Regex intAll = new Regex(pattern, RegexOptions.Singleline);
        match = intAll.Match(input);
        int matches = 0;
        while (match.Success) {
            matches++;
            // Do nothing with the match except get the next match.
            match = match.NextMatch();
        }
        sw.Stop();
        Console.WriteLine(" {0:N0} matches in {1}", matches, sw.Elapsed);

        // Read all sentences with compiled regex.
        Console.WriteLine("All Sentences with Compiled Regex:");
        sw = Stopwatch.StartNew();
        Regex compAll = new Regex(pattern,
                               RegexOptions.Singleline | RegexOptions.Compiled);
        match = compAll.Match(input);
        matches = 0;
        while (match.Success) {
            matches++;
            // Do nothing with the match except get the next match.
            match = match.NextMatch();
        }
    }
}

```

```

        sw.Stop();
        Console.WriteLine("  {0:N0} matches in {1}", matches, sw.Elapsed);
    }
}

// The example displays the following output:
//      10 Sentences with Interpreted Regex:
//          10 matches in 00:00:00.0047491
//      10 Sentences with Compiled Regex:
//          10 matches in 00:00:00.0141872
//      All Sentences with Interpreted Regex:
//          13,443 matches in 00:00:01.1929928
//      All Sentences with Compiled Regex:
//          13,443 matches in 00:00:00.7635869
//
//      >compare1
//      10 Sentences with Interpreted Regex:
//          10 matches in 00:00:00.0046914
//      10 Sentences with Compiled Regex:
//          10 matches in 00:00:00.0143727
//      All Sentences with Interpreted Regex:
//          13,443 matches in 00:00:01.1514100
//      All Sentences with Compiled Regex:
//          13,443 matches in 00:00:00.7432921

```

```

Imports System.Diagnostics
Imports System.IO
Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim pattern As String = "\b(\w+((\r?\n)|,\s))*\w+[.:;!]"
        Dim sw As Stopwatch
        Dim match As Match
        Dim ctr As Integer

        Dim inFile As New StreamReader(".\Dreiser_TheFinancier.txt")
        Dim input As String = inFile.ReadToEnd()
        inFile.Close()

        ' Read first ten sentences with interpreted regex.
        Console.WriteLine("10 Sentences with Interpreted Regex:")
        sw = Stopwatch.StartNew()
        Dim int10 As New Regex(pattern, RegexOptions.SingleLine)
        match = int10.Match(input)
        For ctr = 0 To 9
            If match.Success Then
                ' Do nothing with the match except get the next match.
                match = match.NextMatch()
            Else
                Exit For
            End If
        Next
        sw.Stop()
        Console.WriteLine("  {0} matches in {1}", ctr, sw.Elapsed)

        ' Read first ten sentences with compiled regex.
        Console.WriteLine("10 Sentences with Compiled Regex:")
        sw = Stopwatch.StartNew()
        Dim comp10 As New Regex(pattern,
                               RegexOptions.SingleLine Or RegexOptions.Compiled)
        match = comp10.Match(input)
        For ctr = 0 To 9
            If match.Success Then
                ' Do nothing with the match except get the next match.
                match = match.NextMatch()
            Else
                Exit For
            End If
        Next
    End Sub
End Module

```

```

    End If
Next
sw.Stop()
Console.WriteLine("  {0} matches in {1}", ctr, sw.Elapsed)

' Read all sentences with interpreted regex.
Console.WriteLine("All Sentences with Interpreted Regex:")
sw = Stopwatch.StartNew()
Dim intAll As New Regex(pattern, RegexOptions.SingleLine)
match = intAll.Match(input)
Dim matches As Integer = 0
Do While match.Success
    matches += 1
    ' Do nothing with the match except get the next match.
    match = match.NextMatch()
Loop
sw.Stop()
Console.WriteLine("  {0:N0} matches in {1}", matches, sw.Elapsed)

' Read all sentences with compiled regex.
Console.WriteLine("All Sentences with Compiled Regex:")
sw = Stopwatch.StartNew()
Dim compAll As New Regex(pattern,
                           RegexOptions.SingleLine Or RegexOptions.Compiled)
match = compAll.Match(input)
matches = 0
Do While match.Success
    matches += 1
    ' Do nothing with the match except get the next match.
    match = match.NextMatch()
Loop
sw.Stop()
Console.WriteLine("  {0:N0} matches in {1}", matches, sw.Elapsed)
End Sub
End Module
The example displays output like the following:
' 10 Sentences with Interpreted Regex:
'   10 matches in 00:00:00.0047491
' 10 Sentences with Compiled Regex:
'   10 matches in 00:00:00.0141872
' All Sentences with Interpreted Regex:
'   13,443 matches in 00:00:01.1929928
' All Sentences with Compiled Regex:
'   13,443 matches in 00:00:00.7635869
'

' >compare1
' 10 Sentences with Interpreted Regex:
'   10 matches in 00:00:00.0046914
' 10 Sentences with Compiled Regex:
'   10 matches in 00:00:00.0143727
' All Sentences with Interpreted Regex:
'   13,443 matches in 00:00:01.1514100
' All Sentences with Compiled Regex:
'   13,443 matches in 00:00:00.7432921

```

The regular expression pattern used in the example, `\b(\w+((\r?\n)|,\s))*\w+[.?:;!]`, is defined as shown in the following table.

PATTERN	DESCRIPTION
<code>\b</code>	Begin the match at a word boundary.
<code>\w+</code>	Match one or more word characters.

PATTERN	DESCRIPTION
(\r?\n) ,? \s)	Match either zero or one carriage return followed by a newline character, or zero or one comma followed by a white-space character.
(\w+((\r?\n) ,? \s))*	Match zero or more occurrences of one or more word characters that are followed either by zero or one carriage return and a newline character, or by zero or one comma followed by a white-space character.
\w+	Match one or more word characters.
[.?:;!]	Match a period, question mark, colon, semicolon, or exclamation point.

Regular expressions: Compiled to an assembly

.NET also enables you to create an assembly that contains compiled regular expressions. This moves the performance hit of regular expression compilation from run time to design time. However, it also involves some additional work: You must define the regular expressions in advance and compile them to an assembly. The compiler can then reference this assembly when compiling source code that uses the assembly's regular expressions. Each compiled regular expression in the assembly is represented by a class that derives from [Regex](#).

To compile regular expressions to an assembly, you call the [Regex.CompileToAssembly\(RegexCompilationInfo\[\], AssemblyName\)](#) method and pass it an array of [RegexCompilationInfo](#) objects that represent the regular expressions to be compiled, and an [AssemblyName](#) object that contains information about the assembly to be created.

We recommend that you compile regular expressions to an assembly in the following situations:

- If you are a component developer who wants to create a library of reusable regular expressions.
- If you expect your regular expression's pattern-matching methods to be called an indeterminate number of times -- anywhere from once or twice to thousands or tens of thousands of times. Unlike compiled or interpreted regular expressions, regular expressions that are compiled to separate assemblies offer performance that is consistent regardless of the number of method calls.

If you are using compiled regular expressions to optimize performance, you should not use reflection to create the assembly, load the regular expression engine, and execute its pattern-matching methods. This requires that you avoid building regular expression patterns dynamically, and that you specify any pattern-matching options (such as case-insensitive pattern matching) at the time the assembly is created. It also requires that you separate the code that creates the assembly from the code that uses the regular expression.

The following example shows how to create an assembly that contains a compiled regular expression. It creates an assembly named `RegexLib.dll` with a single regular expression class, `SentencePattern`, that contains the sentence-matching regular expression pattern used in the [Interpreted vs. Compiled Regular Expressions](#) section.

```

using System;
using System.Reflection;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        RegexCompilationInfo SentencePattern =
            new RegexCompilationInfo(@"\b(\w+((\r?\n)|,\?|\s))*\w+[.?;!]",
                RegexOptions.Multiline,
                "SentencePattern",
                "Utilities.RegularExpressions",
                true);
        RegexCompilationInfo[] regexes = { SentencePattern };
        AssemblyName assemName = new AssemblyName("RegexLib, Version=1.0.0.1001, Culture=neutral,
PublicKeyToken=null");
        Regex.CompileToAssembly(regexes, assemName);
    }
}

```

```

Imports System.Reflection
Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim SentencePattern As New RegexCompilationInfo("\b(\w+((\r?\n)|,\?|\s))*\w+[.?;!]",
            RegexOptions.Multiline,
            "SentencePattern",
            "Utilities.RegularExpressions",
            True)
        Dim regexes() As RegexCompilationInfo = {SentencePattern}
        Dim assemName As New AssemblyName("RegexLib, Version=1.0.0.1001, Culture=neutral,
PublicKeyToken=null")
        Regex.CompileToAssembly(regexes, assemName)
    End Sub
End Module

```

When the example is compiled to an executable and run, it creates an assembly named `RegexLib.dll`. The regular expression is represented by a class named `Utilities.RegularExpressions.SentencePattern` that is derived from `Regex`. The following example then uses the compiled regular expression to extract the sentences from the text of Theodore Dreiser's *The Financier*.

```

using System;
using System.IO;
using System.Text.RegularExpressions;
using Utilities.RegularExpressions;

public class Example
{
    public static void Main()
    {
        SentencePattern pattern = new SentencePattern();
        StreamReader inFile = new StreamReader(@"..\Dreiser_TheFinancier.txt");
        string input = inFile.ReadToEnd();
        inFile.Close();

        MatchCollection matches = pattern.Matches(input);
        Console.WriteLine("Found {0:N0} sentences.", matches.Count);
    }
}

// The example displays the following output:
//      Found 13,443 sentences.

```

```

Imports System.IO
Imports System.Text.RegularExpressions
Imports Utilities.RegularExpressions

Module Example
    Public Sub Main()
        Dim pattern As New SentencePattern()
        Dim inFile As New StreamReader(".\Dreiser_TheFinancier.txt")
        Dim input As String = inFile.ReadToEnd()
        inFile.Close()

        Dim matches As MatchCollection = pattern.Matches(input)
        Console.WriteLine("Found {0:N0} sentences.", matches.Count)
    End Sub
End Module

' The example displays the following output:
'      Found 13,443 sentences.

```

Take charge of backtracking

Ordinarily, the regular expression engine uses linear progression to move through an input string and compare it to a regular expression pattern. However, when indeterminate quantifiers such as `*`, `+`, and `?` are used in a regular expression pattern, the regular expression engine may give up a portion of successful partial matches and return to a previously saved state in order to search for a successful match for the entire pattern. This process is known as backtracking.

NOTE

For more information on backtracking, see [Details of Regular Expression Behavior](#) and [Backtracking](#). For a detailed discussion of backtracking, see [Optimizing Regular Expression Performance, Part II: Taking Charge of Backtracking](#) in the BCL Team blog.

Support for backtracking gives regular expressions power and flexibility. It also places the responsibility for controlling the operation of the regular expression engine in the hands of regular expression developers. Because developers are often not aware of this responsibility, their misuse of backtracking or reliance on excessive backtracking often plays the most significant role in degrading regular expression performance. In a worst-case scenario, execution time can double for each additional character in the input string. In fact, by using

backtracking excessively, it is easy to create the programmatic equivalent of an endless loop if input nearly matches the regular expression pattern; the regular expression engine may take hours or even days to process a relatively short input string.

Often, applications pay a performance penalty for using backtracking despite the fact that backtracking is not essential for a match. For example, the regular expression `\b\p{Lu}\w*\b` matches all words that begin with an uppercase character, as the following table shows.

PATTERN	DESCRIPTION
<code>\b</code>	Begin the match at a word boundary.
<code>\p{Lu}</code>	Match an uppercase character.
<code>\w*</code>	Match zero or more word characters.
<code>\b</code>	End the match at a word boundary.

Because a word boundary is not the same as, or a subset of, a word character, there is no possibility that the regular expression engine will cross a word boundary when matching word characters. This means that for this regular expression, backtracking can never contribute to the overall success of any match -- it can only degrade performance, because the regular expression engine is forced to save its state for each successful preliminary match of a word character.

If you determine that backtracking is not necessary, you can disable it by using the `(?>subexpression)` language element, known as an atomic group. The following example parses an input string by using two regular expressions. The first, `\b\p{Lu}\w*\b`, relies on backtracking. The second, `\b\p{Lu}(?>\w*)\b`, disables backtracking. As the output from the example shows, they both produce the same result.

```
using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string input = "This this word Sentence name Capital";
        string pattern = @"\b\p{Lu}\w*\b";
        foreach (Match match in Regex.Matches(input, pattern))
            Console.WriteLine(match.Value);

        Console.WriteLine();

        pattern = @"\b\p{Lu}(?>\w*)\b";
        foreach (Match match in Regex.Matches(input, pattern))
            Console.WriteLine(match.Value);
    }
}
// The example displays the following output:
//      This
//      Sentence
//      Capital
//
//      This
//      Sentence
//      Capital
```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim input As String = "This this word Sentence name Capital"
        Dim pattern As String = "\b\p{Lu}\w*\b"
        For Each match As Match In Regex.Matches(input, pattern)
            Console.WriteLine(match.Value)
        Next
        Console.WriteLine()

        pattern = "\b\p{Lu}(?>\w*)\b"
        For Each match As Match In Regex.Matches(input, pattern)
            Console.WriteLine(match.Value)
        Next
    End Sub
End Module
' The example displays the following output:
' This
' Sentence
' Capital
'
' This
' Sentence
' Capital

```

In many cases, backtracking is essential for matching a regular expression pattern to input text. However, excessive backtracking can severely degrade performance and create the impression that an application has stopped responding. In particular, this happens when quantifiers are nested and the text that matches the outer subexpression is a subset of the text that matches the inner subexpression.

WARNING

In addition to avoiding excessive backtracking, you should use the timeout feature to ensure that excessive backtracking does not severely degrade regular expression performance. For more information, see the [Use Time-out Values](#) section.

For example, the regular expression pattern `^[\0-9A-Z]([-.\w]*[\0-9A-Z])*\$` is intended to match a part number that consists of at least one alphanumeric character. Any additional characters can consist of an alphanumeric character, a hyphen, an underscore, or a period, though the last character must be alphanumeric. A dollar sign terminates the part number. In some cases, this regular expression pattern can exhibit extremely poor performance because quantifiers are nested, and because the subexpression `[\0-9A-Z]` is a subset of the subexpression `[-.\w]*`.

In these cases, you can optimize regular expression performance by removing the nested quantifiers and replacing the outer subexpression with a zero-width lookahead or lookbehind assertion. Lookahead and lookbehind assertions are anchors; they do not move the pointer in the input string, but instead look ahead or behind to check whether a specified condition is met. For example, the part number regular expression can be rewritten as `^[\0-9A-Z][-.\w]*(?<=[\0-9A-Z])\$`. This regular expression pattern is defined as shown in the following table.

PATTERN	DESCRIPTION
<code>^</code>	Begin the match at the beginning of the input string.
<code>[\0-9A-Z]</code>	Match an alphanumeric character. The part number must consist of at least this character.

PATTERN	DESCRIPTION
[- . \w]*	Match zero or more occurrences of any word character, hyphen, or period.
\\$	Match a dollar sign.
(?<=[0-9A-Z])	Look ahead of the ending dollar sign to ensure that the previous character is alphanumeric.
\$	End the match at the end of the input string.

The following example illustrates the use of this regular expression to match an array containing possible part numbers.

```
using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string pattern = @"^([0-9A-Z])[ - . \w]*(?<=[0-9A-Z])\$";
        string[] partNos = { "A1C$", "A4", "A4$", "A1603D$", "A1603D#" };

        foreach (var input in partNos) {
            Match match = Regex.Match(input, pattern);
            if (match.Success)
                Console.WriteLine(match.Value);
            else
                Console.WriteLine("Match not found.");
        }
    }
}
// The example displays the following output:
//      A1C$
//      Match not found.
//      A4$
//      A1603D$
//      Match not found.
```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim pattern As String = "^[0-9A-Z][-.\w]*(?<=[0-9A-Z])\$\$"
        Dim partNos() As String = {"A1C$", "A4", "A4$", "A1603D$",
                                   "A1603D#"}

        For Each input As String In partNos
            Dim match As Match = Regex.Match(input, pattern)
            If match.Success Then
                Console.WriteLine(match.Value)
            Else
                Console.WriteLine("Match not found.")
            End If
        Next
    End Sub
End Module
' The example displays the following output:
'   A1C$
'   Match not found.
'   A4$
'   A1603D$
'   Match not found.

```

The regular expression language in .NET includes the following language elements that you can use to eliminate nested quantifiers. For more information, see [Grouping Constructs](#).

LANGUAGE ELEMENT	DESCRIPTION
(?= subexpression)	Zero-width positive lookahead. Look ahead of the current position to determine whether subexpression matches the input string.
(?! subexpression)	Zero-width negative lookahead. Look ahead of the current position to determine whether subexpression does not match the input string.
(?=<= subexpression)	Zero-width positive lookbehind. Look behind the current position to determine whether subexpression matches the input string.
(?<! subexpression)	Zero-width negative lookbehind. Look behind the current position to determine whether subexpression does not match the input string.

Use time-out values

If your regular expressions processes input that nearly matches the regular expression pattern, it can often rely on excessive backtracking, which impacts its performance significantly. In addition to carefully considering your use of backtracking and testing the regular expression against near-matching input, you should always set a time-out value to ensure that the impact of excessive backtracking, if it occurs, is minimized.

The regular expression time-out interval defines the period of time that the regular expression engine will look for a single match before it times out. Depending on the regular expression pattern and the input text, the execution time may exceed the specified time-out interval, but it will not spend more time backtracking than the specified time-out interval. The default time-out interval is [Regex.InfiniteMatchTimeout](#), which means that the regular expression will not time out. You can override this value and define a time-out interval as follows:

- By providing a time-out value when you instantiate a [Regex](#) object by calling the [Regex\(String, RegexOptions, TimeSpan\)](#) constructor.
- By calling a static pattern matching method, such as [Regex.Match\(String, String, RegexOptions, TimeSpan\)](#) or [Regex.Replace\(String, String, String, RegexOptions, TimeSpan\)](#), that includes a `matchTimeout` parameter.
- For compiled regular expressions that are created by calling the [Regex.CompileToAssembly](#) method, by calling the constructor that has a parameter of type [TimeSpan](#).
- By setting a process-wide or AppDomain-wide value with code such as

```
AppDomain.CurrentDomain.SetData("REGEX_DEFAULT_MATCH_TIMEOUT", TimeSpan.FromMilliseconds(100)); .
```

If you have defined a time-out interval and a match is not found at the end of that interval, the regular expression method throws a [RegexMatchTimeoutException](#) exception. In your exception handler, you can choose to retry the match with a longer time-out interval, abandon the match attempt and assume that there is no match, or abandon the match attempt and log the exception information for future analysis.

The following example defines a `GetWordData` method that instantiates a regular expression with a time-out interval of 350 milliseconds to calculate the number of words and average number of characters in a word in a text document. If the matching operation times out, the time-out interval is increased by 350 milliseconds and the [Regex](#) object is re-instantiated. If the new time-out interval exceeds 1 second, the method re-throws the exception to the caller.

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        RegexUtilities util = new RegexUtilities();
        string title = "Doyle - The Hound of the Baskervilles.txt";
        try {
            var info = util.GetWordData(title);
            Console.WriteLine("Words: {0:N0}", info.Item1);
            Console.WriteLine("Average Word Length: {0:N2} characters", info.Item2);
        }
        catch (IOException e) {
            Console.WriteLine("IOException reading file '{0}'", title);
            Console.WriteLine(e.Message);
        }
        catch (RegexMatchTimeoutException e) {
            Console.WriteLine("The operation timed out after {0:N0} milliseconds",
                e.MatchTimeout.TotalMilliseconds);
        }
    }
}

public class RegexUtilities
{
    public Tuple<int, double> GetWordData(string filename)
    {
        const int MAX_TIMEOUT = 1000; // Maximum timeout interval in milliseconds.
        const int INCREMENT = 350; // Milliseconds increment of timeout.

        List<string> exclusions = new List<string>( new string[] { "a", "an", "the" } );
        int[] wordLengths = new int[29]; // Allocate an array of more than ample size.
        string input = null;
        StreamReader sr = null;
        try {
            sr = new StreamReader(filename);
            input = sr.ReadToEnd();
```

```

        }

        catch (FileNotFoundException e) {
            string msg = String.Format("Unable to find the file '{0}'", filename);
            throw new IOException(msg, e);
        }

        catch (IOException e) {
            throw new IOException(e.Message, e);
        }

        finally {
            if (sr != null) sr.Close();
        }

        int timeoutInterval = INCREMENT;
        bool init = false;
        Regex rgx = null;
        Match m = null;
        int indexPos = 0;
        do {
            try {
                if (!init) {
                    rgx = new Regex(@"\b\w+\b", RegexOptions.None,
                        TimeSpan.FromMilliseconds(timeoutInterval));
                    m = rgx.Match(input, indexPos);
                    init = true;
                }
                else {
                    m = m.NextMatch();
                }
                if (m.Success) {
                    if ( !exclusions.Contains(m.Value.ToLower()))
                        wordLengths[m.Value.Length]++;
                }
                indexPos += m.Length + 1;
            }
        }

        catch (RegexMatchTimeoutException e) {
            if (e.MatchTimeout.TotalMilliseconds < MAX_TIMEOUT) {
                timeoutInterval += INCREMENT;
                init = false;
            }
            else {
                // Rethrow the exception.
                throw;
            }
        }
    } while (m.Success);

    // If regex completed successfully, calculate number of words and average length.
    int nWords = 0;
    long totalLength = 0;

    for (int ctr = wordLengths.GetLowerBound(0); ctr <= wordLengths.GetUpperBound(0); ctr++) {
        nWords += wordLengths[ctr];
        totalLength += ctr * wordLengths[ctr];
    }
    return new Tuple<int, double>(nWords, totalLength/nWords);
}
}

```

```

Imports System.Collections.Generic
Imports System.IO
Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim util As New RegexUtilities()
        Dim title As String = "Doyle - The Hound of the Baskervilles.txt"

```

```

Try
    Dim info = util.GetWordData(title)
    Console.WriteLine("Words: {0:N0}", info.Item1)
    Console.WriteLine("Average Word Length: {0:N2} characters", info.Item2)
Catch e As IOException
    Console.WriteLine("IOException reading file '{0}'", title)
    Console.WriteLine(e.Message)
Catch e As RegexMatchTimeoutException
    Console.WriteLine("The operation timed out after {0:N0} milliseconds",
                      e.MatchTimeout.TotalMilliseconds)
End Try
End Sub
End Module

Public Class RegexUtilities
    Public Function GetWordData(filename As String) As Tuple(Of Integer, Double)
        Const MAX_TIMEOUT As Integer = 1000      ' Maximum timeout interval in milliseconds.
        Const INCREMENT As Integer = 350        ' Milliseconds increment of timeout.

        Dim exclusions As New List(Of String)( {"a", "an", "the"})
        Dim wordLengths(30) As Integer          ' Allocate an array of more than ample size.
        Dim input As String = Nothing
        Dim sr As StreamReader = Nothing
        Try
            sr = New StreamReader(filename)
            input = sr.ReadToEnd()
        Catch e As FileNotFoundException
            Dim msg As String = String.Format("Unable to find the file '{0}'", filename)
            Throw New IOException(msg, e)
        Catch e As IOException
            Throw New IOException(e.Message, e)
        Finally
            If sr IsNot Nothing Then sr.Close()
        End Try

        Dim timeoutInterval As Integer = INCREMENT
        Dim init As Boolean = False
        Dim rgx As Regex = Nothing
        Dim m As Match = Nothing
        Dim indexPos As Integer = 0
        Do
            Try
                If Not init Then
                    rgx = New Regex("\b\w+\b", RegexOptions.None,
                                    TimeSpan.FromMilliseconds(timeoutInterval))
                    m = rgx.Match(input, indexPos)
                    init = True
                Else
                    m = m.NextMatch()
                End If
                If m.Success Then
                    If Not exclusions.Contains(m.Value.ToLower()) Then
                        wordLengths(m.Value.Length) += 1
                    End If
                    indexPos += m.Length + 1
                End If
            Catch e As RegexMatchTimeoutException
                If e.MatchTimeout.TotalMilliseconds < MAX_TIMEOUT Then
                    timeoutInterval += INCREMENT
                    init = False
                Else
                    ' Rethrow the exception.
                    Throw
                End If
            End Try
        Loop While m.Success
        ' If regex completed successfully, calculate number of words and average length.
        Dim nWords As Integer

```

```

Dim totalLength As Long

For ctr As Integer = wordLengths.GetLowerBound(0) To wordLengths.GetUpperBound(0)
    nWords += wordLengths(ctr)
    totalLength += ctr * wordLengths(ctr)
Next
Return New Tuple(Of Integer, Double)(nWords, totalLength / nWords)
End Function
End Class

```

Capture only when necessary

Regular expressions in .NET support a number of grouping constructs, which let you group a regular expression pattern into one or more subexpressions. The most commonly used grouping constructs in .NET regular expression language are `(subexpression)`, which defines a numbered capturing group, and `(?< name > subexpression)`, which defines a named capturing group. Grouping constructs are essential for creating backreferences and for defining a subexpression to which a quantifier is applied.

However, the use of these language elements has a cost. They cause the [GroupCollection](#) object returned by the [Match.Groups](#) property to be populated with the most recent unnamed or named captures, and if a single grouping construct has captured multiple substrings in the input string, they also populate the [CaptureCollection](#) object returned by the [Group.Captures](#) property of a particular capturing group with multiple [Capture](#) objects.

Often, grouping constructs are used in a regular expression only so that quantifiers can be applied to them, and the groups captured by these subexpressions are not subsequently used. For example, the regular expression `\b(\w+[, ,]?\s?)+[.?!]` is designed to capture an entire sentence. The following table describes the language elements in this regular expression pattern and their effect on the [Match](#) object's [Match.Groups](#) and [Group.Captures](#) collections.

PATTERN	DESCRIPTION
<code>\b</code>	Begin the match at a word boundary.
<code>\w+</code>	Match one or more word characters.
<code>[;,]?</code>	Match zero or one comma or semicolon.
<code>\s?</code>	Match zero or one white-space character.
<code>(\w+[, ,]?\s?)^</code>	Match one or more occurrences of one or more word characters followed by an optional comma or semicolon followed by an optional white-space character. This defines the first capturing group, which is necessary so that the combination of multiple word characters (that is, a word) followed by an optional punctuation symbol will be repeated until the regular expression engine reaches the end of a sentence.
<code>[.?!]</code>	Match a period, question mark, or exclamation point.

As the following example shows, when a match is found, both the [GroupCollection](#) and [CaptureCollection](#) objects are populated with captures from the match. In this case, the capturing group `(\w+[, ,]?\s?)` exists so that the `^` quantifier can be applied to it, which enables the regular expression pattern to match each word in a sentence. Otherwise, it would match the last word in a sentence.

```
using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string input = "This is one sentence. This is another.";
        string pattern = @"\b(\w+[, ]?\s?)+[.?!]";

        foreach (Match match in Regex.Matches(input, pattern)) {
            Console.WriteLine("Match: '{0}' at index {1}.",
                match.Value, match.Index);

            int grpCtr = 0;
            foreach (Group grp in match.Groups) {
                Console.WriteLine("  Group {0}: '{1}' at index {2}.",
                    grpCtr, grp.Value, grp.Index);
                grpCtr++;

                int capCtr = 0;
                foreach (Capture cap in grp.Captures) {
                    Console.WriteLine("    Capture {0}: '{1}' at {2}.",
                        capCtr, cap.Value, cap.Index);
                    capCtr++;
                }
            }
            Console.WriteLine();
        }
    }
}

// The example displays the following output:
//     Match: 'This is one sentence.' at index 0.
//         Group 0: 'This is one sentence.' at index 0.
//             Capture 0: 'This is one sentence.' at 0.
//             Group 1: 'sentence' at index 12.
//                 Capture 0: 'This ' at 0.
//                 Capture 1: 'is ' at 5.
//                 Capture 2: 'one ' at 8.
//                 Capture 3: 'sentence' at 12.
//
//     Match: 'This is another.' at index 22.
//         Group 0: 'This is another.' at index 22.
//             Capture 0: 'This is another.' at 22.
//             Group 1: 'another' at index 30.
//                 Capture 0: 'This ' at 22.
//                 Capture 1: 'is ' at 27.
//                 Capture 2: 'another' at 30.
```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim input As String = "This is one sentence. This is another."
        Dim pattern As String = "\b(\w+[,]?[s?]+[.?!])"

        For Each match As Match In Regex.Matches(input, pattern)
            Console.WriteLine("Match: '{0}' at index {1}.",
                match.Value, match.Index)

            Dim grpCtr As Integer = 0
            For Each grp As Group In match.Groups
                Console.WriteLine("    Group {0}: '{1}' at index {2}.",
                    grpCtr, grp.Value, grp.Index)
                Dim capCtr As Integer = 0
                For Each cap As Capture In grp.Captures
                    Console.WriteLine("        Capture {0}: '{1}' at {2}.",
                        capCtr, cap.Value, cap.Index)
                    capCtr += 1
                Next
                grpCtr += 1
            Next
            Console.WriteLine()
        Next
    End Sub
End Module
' The example displays the following output:
'   Match: 'This is one sentence.' at index 0.
'       Group 0: 'This is one sentence.' at index 0.
'           Capture 0: 'This is one sentence.' at 0.
'           Group 1: 'sentence' at index 12.
'               Capture 0: 'This ' at 0.
'               Capture 1: 'is ' at 5.
'               Capture 2: 'one ' at 8.
'               Capture 3: 'sentence' at 12.
'
'   Match: 'This is another.' at index 22.
'       Group 0: 'This is another.' at index 22.
'           Capture 0: 'This is another.' at 22.
'           Group 1: 'another' at index 30.
'               Capture 0: 'This ' at 22.
'               Capture 1: 'is ' at 27.
'               Capture 2: 'another' at 30.
'
```

When you use subexpressions only to apply quantifiers to them, and you are not interested in the captured text, you should disable group captures. For example, the `(?:subexpression)` language element prevents the group to which it applies from capturing matched substrings. In the following example, the regular expression pattern from the previous example is changed to `\b(?:\w+[,]?[s?]+[.?!])`. As the output shows, it prevents the regular expression engine from populating the `GroupCollection` and `CaptureCollection` collections.

```
using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string input = "This is one sentence. This is another.";
        string pattern = @"\b(?:\w+[,]\s?)+[.?!]";

        foreach (Match match in Regex.Matches(input, pattern)) {
            Console.WriteLine("Match: '{0}' at index {1}.",
                match.Value, match.Index);

            int grpCtr = 0;
            foreach (Group grp in match.Groups) {
                Console.WriteLine("  Group {0}: '{1}' at index {2}.",
                    grpCtr, grp.Value, grp.Index);
                grpCtr++;

                int capCtr = 0;
                foreach (Capture cap in grp.Captures) {
                    Console.WriteLine("    Capture {0}: '{1}' at {2}.",
                        capCtr, cap.Value, cap.Index);
                    capCtr++;
                }
            }
            Console.WriteLine();
        }
    }
}

// The example displays the following output:
//      Match: 'This is one sentence.' at index 0.
//          Group 0: 'This is one sentence.' at index 0.
//              Capture 0: 'This is one sentence.' at 0.
//
//      Match: 'This is another.' at index 22.
//          Group 0: 'This is another.' at index 22.
//              Capture 0: 'This is another.' at 22.
```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim input As String = "This is one sentence. This is another."
        Dim pattern As String = "\b(?:\w+[, ]?\s?)+[.?!"]

        For Each match As Match In Regex.Matches(input, pattern)
            Console.WriteLine("Match: '{0}' at index {1}.",
                match.Value, match.Index)

            Dim grpCtr As Integer = 0
            For Each grp As Group In match.Groups
                Console.WriteLine("    Group {0}: '{1}' at index {2}.",
                    grpCtr, grp.Value, grp.Index)
                Dim capCtr As Integer = 0
                For Each cap As Capture In grp.Captures
                    Console.WriteLine("        Capture {0}: '{1}' at {2}.",
                        capCtr, cap.Value, cap.Index)
                    capCtr += 1
                Next
                grpCtr += 1
            Next
            Console.WriteLine()
        Next
    End Sub
End Module
' The example displays the following output:
'     Match: 'This is one sentence.' at index 0.
'         Group 0: 'This is one sentence.' at index 0.
'             Capture 0: 'This is one sentence.' at 0.
'
'     Match: 'This is another.' at index 22.
'         Group 0: 'This is another.' at index 22.
'             Capture 0: 'This is another.' at 22.

```

You can disable captures in one of the following ways:

- Use the `(?:subexpression)` language element. This element prevents the capture of matched substrings in the group to which it applies. It does not disable substring captures in any nested groups.
- Use the [ExplicitCapture](#) option. It disables all unnamed or implicit captures in the regular expression pattern. When you use this option, only substrings that match named groups defined with the `(?<name>subexpression)` language element can be captured. The [ExplicitCapture](#) flag can be passed to the `options` parameter of a [Regex](#) class constructor or to the `options` parameter of a [Regex](#) static matching method.
- Use the `n` option in the `(?imnsx)` language element. This option disables all unnamed or implicit captures from the point in the regular expression pattern at which the element appears. Captures are disabled either until the end of the pattern or until the `(-n)` option enables unnamed or implicit captures. For more information, see [Miscellaneous Constructs](#).
- Use the `n` option in the `(?imnsx:subexpression)` language element. This option disables all unnamed or implicit captures in `subexpression`. Captures by any unnamed or implicit nested capturing groups are disabled as well.

Related topics

TITLE	DESCRIPTION
-------	-------------

TITLE	DESCRIPTION
Details of Regular Expression Behavior	Examines the implementation of the regular expression engine in .NET. The topic focuses on the flexibility of regular expressions and explains the developer's responsibility for ensuring the efficient and robust operation of the regular expression engine.
Backtracking	Explains what backtracking is and how it affects regular expression performance, and examines language elements that provide alternatives to backtracking.
Regular Expression Language - Quick Reference	Describes the elements of the regular expression language in .NET and provides links to detailed documentation for each language element.

The Regular Expression Object Model

9/20/2022 • 29 minutes to read • [Edit Online](#)

This topic describes the object model used in working with .NET regular expressions. It contains the following sections:

- [The Regular Expression Engine](#)
- [The MatchCollection and Match Objects](#)
- [The Group Collection](#)
- [The Captured Group](#)
- [The Capture Collection](#)
- [The Individual Capture](#)

The Regular Expression Engine

The regular expression engine in .NET is represented by the [Regex](#) class. The regular expression engine is responsible for parsing and compiling a regular expression, and for performing operations that match the regular expression pattern with an input string. The engine is the central component in the .NET regular expression object model.

You can use the regular expression engine in either of two ways:

- By calling the static methods of the [Regex](#) class. The method parameters include the input string and the regular expression pattern. The regular expression engine caches regular expressions that are used in static method calls, so repeated calls to static regular expression methods that use the same regular expression offer relatively good performance.
- By instantiating a [Regex](#) object, by passing a regular expression to the class constructor. In this case, the [Regex](#) object is immutable (read-only) and represents a regular expression engine that is tightly coupled with a single regular expression. Because regular expressions used by [Regex](#) instances are not cached, you should not instantiate a [Regex](#) object multiple times with the same regular expression.

You can call the methods of the [Regex](#) class to perform the following operations:

- Determine whether a string matches a regular expression pattern.
- Extract a single match or the first match.
- Extract all matches.
- Replace a matched substring.
- Split a single string into an array of strings.

These operations are described in the following sections.

Matching a Regular Expression Pattern

The [Regex.IsMatch](#) method returns `true` if the string matches the pattern, or `false` if it does not. The [IsMatch](#) method is often used to validate string input. For example, the following code ensures that a string matches a valid social security number in the United States.

```

using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string[] values = { "111-22-3333", "111-2-3333" };
        string pattern = @"^\d{3}-\d{2}-\d{4}$";
        foreach (string value in values)
        {
            if (Regex.IsMatch(value, pattern))
                Console.WriteLine("{0} is a valid SSN.", value);
            else
                Console.WriteLine("{0}: Invalid", value);
        }
    }
}

// The example displays the following output:
//      111-22-3333 is a valid SSN.
//      111-2-3333: Invalid

```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim values() As String = {"111-22-3333", "111-2-3333"}
        Dim pattern As String = "^\d{3}-\d{2}-\d{4}$"
        For Each value As String In values
            If Regex.IsMatch(value, pattern) Then
                Console.WriteLine("{0} is a valid SSN.", value)
            Else
                Console.WriteLine("{0}: Invalid", value)
            End If
        Next
    End Sub
End Module
' The example displays the following output:
'      111-22-3333 is a valid SSN.
'      111-2-3333: Invalid

```

The regular expression pattern `^\d{3}-\d{2}-\d{4}$` is interpreted as shown in the following table.

PATTERN	DESCRIPTION
<code>^</code>	Match the beginning of the input string.
<code>\d{3}</code>	Match three decimal digits.
<code>-</code>	Match a hyphen.
<code>\d{2}</code>	Match two decimal digits.
<code>-</code>	Match a hyphen.
<code>\d{4}</code>	Match four decimal digits.
<code>\$</code>	Match the end of the input string.

Extracting a Single Match or the First Match

The `Regex.Match` method returns a `Match` object that contains information about the first substring that matches a regular expression pattern. If the `Match.Success` property returns `true`, indicating that a match was found, you can retrieve information about subsequent matches by calling the `Match.NextMatch` method. These method calls can continue until the `Match.Success` property returns `false`. For example, the following code uses the `Regex.Match(String, String)` method to find the first occurrence of a duplicated word in a string. It then calls the `Match.NextMatch` method to find any additional occurrences. The example examines the `Match.Success` property after each method call to determine whether the current match was successful and whether a call to the `Match.NextMatch` method should follow.

```
using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string input = "This is a a farm that that raises dairy cattle.";
        string pattern = @"\b(\w+)\W+(\1)\b";
        Match match = Regex.Match(input, pattern);
        while (match.Success)
        {
            Console.WriteLine("Duplicate '{0}' found at position {1}.",
                match.Groups[1].Value, match.Groups[2].Index);
            match = match.NextMatch();
        }
    }
}
// The example displays the following output:
//     Duplicate 'a' found at position 10.
//     Duplicate 'that' found at position 22.
```

```
Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim input As String = "This is a a farm that that raises dairy cattle."
        Dim pattern As String = "\b(\w+)\W+(\1)\b"
        Dim match As Match = Regex.Match(input, pattern)
        Do While match.Success
            Console.WriteLine("Duplicate '{0}' found at position {1}.",
                match.Groups(1).Value, match.Groups(2).Index)
            match = match.NextMatch()
        Loop
    End Sub
End Module
' The example displays the following output:
'     Duplicate 'a' found at position 10.
'     Duplicate 'that' found at position 22.
```

The regular expression pattern `\b(\w+)\W+(\1)\b` is interpreted as shown in the following table.

PATTERN	DESCRIPTION
<code>\b</code>	Begin the match on a word boundary.
<code>(\w+)</code>	Match one or more word characters. This is the first capturing group.
<code>\W+</code>	Match one or more non-word characters.

PATTERN	DESCRIPTION
(\1)	Match the first captured string. This is the second capturing group.
\b	End the match on a word boundary.

Extracting All Matches

The [Regex.Matches](#) method returns a [MatchCollection](#) object that contains information about all matches that the regular expression engine found in the input string. For example, the previous example could be rewritten to call the [Matches](#) method instead of the [Match](#) and [NextMatch](#) methods.

```
using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string input = "This is a a farm that that raises dairy cattle.";
        string pattern = @"\b(\w+)\W+(\1)\b";
        foreach (Match match in Regex.Matches(input, pattern))
            Console.WriteLine("Duplicate '{0}' found at position {1}.",
                match.Groups[1].Value, match.Groups[2].Index);
    }
}
// The example displays the following output:
//     Duplicate 'a' found at position 10.
//     Duplicate 'that' found at position 22.
```

```
Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim input As String = "This is a a farm that that raises dairy cattle."
        Dim pattern As String = "\b(\w+)\W+(\1)\b"
        For Each match As Match In Regex.Matches(input, pattern)
            Console.WriteLine("Duplicate '{0}' found at position {1}.",
                match.Groups(1).Value, match.Groups(2).Index)
        Next
    End Sub
End Module
' The example displays the following output:
'     Duplicate 'a' found at position 10.
'     Duplicate 'that' found at position 22.
```

Replacing a Matched Substring

The [Regex.Replace](#) method replaces each substring that matches the regular expression pattern with a specified string or regular expression pattern, and returns the entire input string with replacements. For example, the following code adds a U.S. currency symbol before a decimal number in a string.

```

using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string pattern = @"\b\d+\.\d{2}\b";
        string replacement = "$$$&";
        string input = "Total Cost: 103.64";
        Console.WriteLine(Regex.Replace(input, pattern, replacement));
    }
}

// The example displays the following output:
//      Total Cost: $103.64

```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim pattern As String = "\b\d+\.\d{2}\b"
        Dim replacement As String = "$$$&"
        Dim input As String = "Total Cost: 103.64"
        Console.WriteLine(Regex.Replace(input, pattern, replacement))
    End Sub
End Module

' The example displays the following output:
'      Total Cost: $103.64

```

The regular expression pattern `\b\d+\.\d{2}\b` is interpreted as shown in the following table.

PATTERN	DESCRIPTION
<code>\b</code>	Begin the match at a word boundary.
<code>\d+</code>	Match one or more decimal digits.
<code>\.</code>	Match a period.
<code>\d{2}</code>	Match two decimal digits.
<code>\b</code>	End the match at a word boundary.

The replacement pattern `$$$&` is interpreted as shown in the following table.

PATTERN	REPLACEMENT STRING
<code>\$\$</code>	The dollar sign (\$) character.
<code>&</code>	The entire matched substring.

Splitting a Single String into an Array of Strings

The [Regex.Split](#) method splits the input string at the positions defined by a regular expression match. For example, the following code places the items in a numbered list into a string array.

```

using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string input = "1. Eggs 2. Bread 3. Milk 4. Coffee 5. Tea";
        string pattern = @"\b\d{1,2}\.\s";
        foreach (string item in Regex.Split(input, pattern))
        {
            if (!String.IsNullOrEmpty(item))
                Console.WriteLine(item);
        }
    }
}

// The example displays the following output:
//      Eggs
//      Bread
//      Milk
//      Coffee
//      Tea

```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim input As String = "1. Eggs 2. Bread 3. Milk 4. Coffee 5. Tea"
        Dim pattern As String = "\b\d{1,2}\.\s"
        For Each item As String In Regex.Split(input, pattern)
            If Not String.IsNullOrEmpty(item) Then
                Console.WriteLine(item)
            End If
        Next
    End Sub
End Module

' The example displays the following output:
'      Eggs
'      Bread
'      Milk
'      Coffee
'      Tea

```

The regular expression pattern `\b\d{1,2}\.\s` is interpreted as shown in the following table.

PATTERN	DESCRIPTION
<code>\b</code>	Begin the match at a word boundary.
<code>\d{1,2}</code>	Match one or two decimal digits.
<code>\.</code>	Match a period.
<code>\s</code>	Match a white-space character.

The MatchCollection and Match Objects

Regex methods return two objects that are part of the regular expression object model: the [MatchCollection](#) object, and the [Match](#) object.

The Match Collection

The [Regex.Matches](#) method returns a [MatchCollection](#) object that contains [Match](#) objects that represent all the matches that the regular expression engine found, in the order in which they occur in the input string. If there are no matches, the method returns a [MatchCollection](#) object with no members. The [MatchCollection.Item\[\]](#) property lets you access individual members of the collection by index, from zero to one less than the value of the [MatchCollection.Count](#) property. [Item\[\]](#) is the collection's indexer (in C#) and default property (in Visual Basic).

By default, the call to the [Regex.Matches](#) method uses lazy evaluation to populate the [MatchCollection](#) object. Access to properties that require a fully populated collection, such as the [MatchCollection.Count](#) and [MatchCollection.Item\[\]](#) properties, may involve a performance penalty. As a result, we recommend that you access the collection by using the [IEnumerator](#) object that is returned by the [MatchCollection.GetEnumerator](#) method. Individual languages provide constructs, such as `For Each` in Visual Basic and `foreach` in C#, that wrap the collection's [IEnumerator](#) interface.

The following example uses the [Regex.Matches\(String\)](#) method to populate a [MatchCollection](#) object with all the matches found in an input string. The example enumerates the collection, copies the matches to a string array, and records the character positions in an integer array.

```
using System;
using System.Collections.Generic;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        MatchCollection matches;
        List<string> results = new List<string>();
        List<int> matchposition = new List<int>();

        // Create a new Regex object and define the regular expression.
        Regex r = new Regex("abc");
        // Use the Matches method to find all matches in the input string.
        matches = r.Matches("123abc4abcd");
        // Enumerate the collection to retrieve all matches and positions.
        foreach (Match match in matches)
        {
            // Add the match string to the string array.
            results.Add(match.Value);
            // Record the character position where the match was found.
            matchposition.Add(match.Index);
        }
        // List the results.
        for (int ctr = 0; ctr < results.Count; ctr++)
            Console.WriteLine("{0} found at position {1}.",
                results[ctr], matchposition[ctr]);
    }
}

// The example displays the following output:
//      'abc' found at position 3.
//      'abc' found at position 7.
```

```

Imports System.Collections.Generic
Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim matches As MatchCollection
        Dim results As New List(Of String)
        Dim matchposition As New List(Of Integer)

        ' Create a new Regex object and define the regular expression.
        Dim r As New Regex("abc")
        ' Use the Matches method to find all matches in the input string.
        matches = r.Matches("123abc4abcd")
        ' Enumerate the collection to retrieve all matches and positions.
        For Each match As Match In matches
            ' Add the match string to the string array.
            results.Add(match.Value)
            ' Record the character position where the match was found.
            matchposition.Add(match.Index)
        Next
        ' List the results.
        For ctr As Integer = 0 To results.Count - 1
            Console.WriteLine("{0} found at position {1}.", _
                results(ctr), matchposition(ctr))
        Next
    End Sub
End Module
' The example displays the following output:
'     'abc' found at position 3.
'     'abc' found at position 7.

```

The Match

The [Match](#) class represents the result of a single regular expression match. You can access [Match](#) objects in two ways:

- By retrieving them from the [MatchCollection](#) object that is returned by the [Regex.Matches](#) method. To retrieve individual [Match](#) objects, iterate the collection by using a `foreach` (in C#) or `For Each ... Next` (in Visual Basic) construct, or use the [MatchCollection.Item\[\]](#) property to retrieve a specific [Match](#) object either by index or by name. You can also retrieve individual [Match](#) objects from the collection by iterating the collection by index, from zero to one less than the number of objects in the collection. However, this method does not take advantage of lazy evaluation, because it accesses the [MatchCollection.Count](#) property.

The following example retrieves individual [Match](#) objects from a [MatchCollection](#) object by iterating the collection using the `foreach` or `For Each ... Next` construct. The regular expression simply matches the string "abc" in the input string.

```

using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string pattern = "abc";
        string input = "abc123abc456abc789";
        foreach (Match match in Regex.Matches(input, pattern))
            Console.WriteLine("{0} found at position {1}.",
                match.Value, match.Index);
    }
}
// The example displays the following output:
//      abc found at position 0.
//      abc found at position 6.
//      abc found at position 12.

```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim pattern As String = "abc"
        Dim input As String = "abc123abc456abc789"
        For Each match As Match In Regex.Matches(input, pattern)
            Console.WriteLine("{0} found at position {1}.",
                match.Value, match.Index)
        Next
    End Sub
End Module
' The example displays the following output:
'      abc found at position 0.
'      abc found at position 6.
'      abc found at position 12.

```

- By calling the [Regex.Match](#) method, which returns a [Match](#) object that represents the first match in a string or a portion of a string. You can determine whether the match has been found by retrieving the value of the [Match.Success](#) property. To retrieve [Match](#) objects that represent subsequent matches, call the [Match.NextMatch](#) method repeatedly, until the [Success](#) property of the returned [Match](#) object is [false](#).

The following example uses the [Regex.Match\(String, String\)](#) and [Match.NextMatch](#) methods to match the string "abc" in the input string.

```

using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string pattern = "abc";
        string input = "abc123abc456abc789";
        Match match = Regex.Match(input, pattern);
        while (match.Success)
        {
            Console.WriteLine("{0} found at position {1}.",
                match.Value, match.Index);
            match = match.NextMatch();
        }
    }
}
// The example displays the following output:
//      abc found at position 0.
//      abc found at position 6.
//      abc found at position 12.

```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim pattern As String = "abc"
        Dim input As String = "abc123abc456abc789"
        Dim match As Match = Regex.Match(input, pattern)
        Do While match.Success
            Console.WriteLine("{0} found at position {1}.",
                match.Value, match.Index)
            match = match.NextMatch()
        Loop
    End Sub
End Module
' The example displays the following output:
'      abc found at position 0.
'      abc found at position 6.
'      abc found at position 12.

```

Two properties of the [Match](#) class return collection objects:

- The [Match.Groups](#) property returns a [GroupCollection](#) object that contains information about the substrings that match capturing groups in the regular expression pattern.
- The [Match.Captures](#) property returns a [CaptureCollection](#) object that is of limited use. The collection is not populated for a [Match](#) object whose [Success](#) property is `false`. Otherwise, it contains a single [Capture](#) object that has the same information as the [Match](#) object.

For more information about these objects, see [The Group Collection](#) and [The Capture Collection](#) sections later in this topic.

Two additional properties of the [Match](#) class provide information about the match. The [Match.Value](#) property returns the substring in the input string that matches the regular expression pattern. The [Match.Index](#) property returns the zero-based starting position of the matched string in the input string.

The [Match](#) class also has two pattern-matching methods:

- The [Match.NextMatch](#) method finds the match after the match represented by the current [Match](#) object, and returns a [Match](#) object that represents that match.

- The `Match.Result` method performs a specified replacement operation on the matched string and returns the result.

The following example uses the `Match.Result` method to prepend a \$ symbol and a space before every number that includes two fractional digits.

```
using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string pattern = @"\b\d+(,\d{3})*\.\d{2}\b";
        string input = "16.32\n194.03\n1,903,672.08";

        foreach (Match match in Regex.Matches(input, pattern))
            Console.WriteLine(match.Result("$$ $&"));
    }
}
// The example displays the following output:
//      $ 16.32
//      $ 194.03
//      $ 1,903,672.08
```

```
Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim pattern As String = "\b\d+(,\d{3})*\.\d{2}\b"
        Dim input As String = "16.32" + vbCrLf + "194.03" + vbCrLf + "1,903,672.08"

        For Each match As Match In Regex.Matches(input, pattern)
            Console.WriteLine(match.Result("$$ $&"))
        Next
    End Sub
End Module
' The example displays the following output:
'      $ 16.32
'      $ 194.03
'      $ 1,903,672.08
```

The regular expression pattern `\b\d+(,\d{3})*\.\d{2}\b` is defined as shown in the following table.

PATTERN	DESCRIPTION
<code>\b</code>	Begin the match at a word boundary.
<code>\d+</code>	Match one or more decimal digits.
<code>(,\d{3})*</code>	Match zero or more occurrences of a comma followed by three decimal digits.
<code>\.</code>	Match the decimal point character.
<code>\d{2}</code>	Match two decimal digits.
<code>\b</code>	End the match at a word boundary.

The replacement pattern `$$ $&` indicates that the matched substring should be replaced by a dollar sign (\$) symbol (the `$$` pattern), a space, and the value of the match (the `$&` pattern).

[Back to top](#)

The Group Collection

The [Match.Groups](#) property returns a [GroupCollection](#) object that contains [Group](#) objects that represent captured groups in a single match. The first [Group](#) object in the collection (at index 0) represents the entire match. Each object that follows represents the results of a single capturing group.

You can retrieve individual [Group](#) objects in the collection by using the [GroupCollection.Item\[\]](#) property. You can retrieve unnamed groups by their ordinal position in the collection, and retrieve named groups either by name or by ordinal position. Unnamed captures appear first in the collection, and are indexed from left to right in the order in which they appear in the regular expression pattern. Named captures are indexed after unnamed captures, from left to right in the order in which they appear in the regular expression pattern. To determine what numbered groups are available in the collection returned for a particular regular expression matching method, you can call the instance [Regex.GetGroupNumbers](#) method. To determine what named groups are available in the collection, you can call the instance [Regex.GetGroupNames](#) method. Both methods are particularly useful in general-purpose routines that analyze the matches found by any regular expression.

The [GroupCollection.Item\[\]](#) property is the indexer of the collection in C# and the collection object's default property in Visual Basic. This means that individual [Group](#) objects can be accessed by index (or by name, in the case of named groups) as follows:

```
Group group = match.Groups[ctr];
```

```
Dim group As Group = match.Groups(ctrl)
```

The following example defines a regular expression that uses grouping constructs to capture the month, day, and year of a date.

```
using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string pattern = @"\b(\w+)\s(\d{1,2}),\s(\d{4})\b";
        string input = "Born: July 28, 1989";
        Match match = Regex.Match(input, pattern);
        if (match.Success)
            for (int ctr = 0; ctr < match.Groups.Count; ctr++)
                Console.WriteLine("Group {0}: {1}", ctr, match.Groups[ctr].Value);
    }
}
// The example displays the following output:
//      Group 0: Born: July 28, 1989
//      Group 1: July
//      Group 2: 28
//      Group 3: 1989
```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim pattern As String = "\b(\w+)\s(\d{1,2}),\s(\d{4})\b"
        Dim input As String = "Born: July 28, 1989"
        Dim match As Match = Regex.Match(input, pattern)
        If match.Success Then
            For ctr As Integer = 0 To match.Groups.Count - 1
                Console.WriteLine("Group {0}: {1}", ctr, match.Groups(ctr).Value)
            Next
        End If
    End Sub
End Module
' The example displays the following output:
' Group 0: July 28, 1989
' Group 1: July
' Group 2: 28
' Group 3: 1989

```

The regular expression pattern `\b(\w+)\s(\d{1,2}),\s(\d{4})\b` is defined as shown in the following table.

PATTERN	DESCRIPTION
<code>\b</code>	Begin the match at a word boundary.
<code>(\w+)</code>	Match one or more word characters. This is the first capturing group.
<code>\s</code>	Match a white-space character.
<code>(\d{1,2})</code>	Match one or two decimal digits. This is the second capturing group.
<code>,</code>	Match a comma.
<code>\s</code>	Match a white-space character.
<code>(\d{4})</code>	Match four decimal digits. This is the third capturing group.
<code>\b</code>	End the match on a word boundary.

[Back to top](#)

The Captured Group

The [Group](#) class represents the result from a single capturing group. Group objects that represent the capturing groups defined in a regular expression are returned by the [Item\[\]](#) property of the [GroupCollection](#) object returned by the [Match.Groups](#) property. The [Item\[\]](#) property is the indexer (in C#) and the default property (in Visual Basic) of the [Group](#) class. You can also retrieve individual members by iterating the collection using the [foreach](#) or [For Each](#) construct. For an example, see the previous section.

The following example uses nested grouping constructs to capture substrings into groups. The regular expression pattern `(a(b))c` matches the string "abc". It assigns the substring "ab" to the first capturing group, and the substring "b" to the second capturing group.

```

var matchposition = new List<int>();
var results = new List<string>();
// Define substrings abc, ab, b.
var r = new Regex("(a(b))c");
Match m = r.Match("abdabc");
for (int i = 0; m.Groups[i].Value != ""; i++)
{
    // Add groups to string array.
    results.Add(m.Groups[i].Value);
    // Record character position.
    matchposition.Add(m.Groups[i].Index);
}

// Display the capture groups.
for (int ctr = 0; ctr < results.Count; ctr++)
    Console.WriteLine("{0} at position {1}",
                      results[ctr], matchposition[ctr]);
// The example displays the following output:
//      abc at position 3
//      ab at position 3
//      b at position 4

```

```

Dim matchposition As New List(Of Integer)
Dim results As New List(Of String)
' Define substrings abc, ab, b.
Dim r As New Regex("(a(b))c")
Dim m As Match = r.Match("abdabc")
Dim i As Integer = 0
While Not (m.Groups(i).Value = "")
    ' Add groups to string array.
    results.Add(m.Groups(i).Value)
    ' Record character position.
    matchposition.Add(m.Groups(i).Index)
    i += 1
End While

' Display the capture groups.
For ctr As Integer = 0 to results.Count - 1
    Console.WriteLine("{0} at position {1}",
                      results(ctr), matchposition(ctr))
Next
' The example displays the following output:
'      abc at position 3
'      ab at position 3
'      b at position 4

```

The following example uses named grouping constructs to capture substrings from a string that contains data in the format "DATANAME:VALUE", which the regular expression splits at the colon (:).

```

var r = new Regex(@"^(?<name>\w+):(?<value>\w+)");
Match m = r.Match("Section1:119900");
Console.WriteLine(m.Groups["name"].Value);
Console.WriteLine(m.Groups["value"].Value);
// The example displays the following output:
//      Section1
//      119900

```

```

Dim r As New Regex("^(<name>\w+):(<value>\w+)")
Dim m As Match = r.Match("Section1:119900")
Console.WriteLine(m.Groups("name").Value)
Console.WriteLine(m.Groups("value").Value)
' The example displays the following output:
'     Section1
'     119900

```

The regular expression pattern `^(<name>\w+):(<value>\w+)` is defined as shown in the following table.

PATTERN	DESCRIPTION
<code>^</code>	Begin the match at the beginning of the input string.
<code>(<name>\w+)</code>	Match one or more word characters. The name of this capturing group is <code>name</code> .
<code>:</code>	Match a colon.
<code>(<value>\w+)</code>	Match one or more word characters. The name of this capturing group is <code>value</code> .

The properties of the [Group](#) class provide information about the captured group: The `Group.Value` property contains the captured substring, the `Group.Index` property indicates the starting position of the captured group in the input text, the `Group.Length` property contains the length of the captured text, and the `Group.Success` property indicates whether a substring matched the pattern defined by the capturing group.

Applying quantifiers to a group (for more information, see [Quantifiers](#)) modifies the relationship of one capture per capturing group in two ways:

- If the `*` or `*?` quantifier (which specifies zero or more matches) is applied to a group, a capturing group may not have a match in the input string. When there is no captured text, the properties of the [Group](#) object are set as shown in the following table.

GROUP PROPERTY	VALUE
<code>Success</code>	<code>false</code>
<code>Value</code>	<code>String.Empty</code>
<code>Length</code>	0

The following example provides an illustration. In the regular expression pattern `aaa(bbb)*ccc`, the first capturing group (the substring "bbb") can be matched zero or more times. Because the input string "aaaccc" matches the pattern, the capturing group does not have a match.

```

using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string pattern = "aaa(bbb)*ccc";
        string input = "aaaccc";
        Match match = Regex.Match(input, pattern);
        Console.WriteLine("Match value: {0}", match.Value);
        if (match.Groups[1].Success)
            Console.WriteLine("Group 1 value: {0}", match.Groups[1].Value);
        else
            Console.WriteLine("The first capturing group has no match.");
    }
}
// The example displays the following output:
//      Match value: aaaccc
//      The first capturing group has no match.

```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim pattern As String = "aaa(bbb)*ccc"
        Dim input As String = "aaaccc"
        Dim match As Match = Regex.Match(input, pattern)
        Console.WriteLine("Match value: {0}", match.Value)
        If match.Groups(1).Success Then
            Console.WriteLine("Group 1 value: {0}", match.Groups(1).Value)
        Else
            Console.WriteLine("The first capturing group has no match.")
        End If
    End Sub
End Module
' The example displays the following output:
'      Match value: aaaccc
'      The first capturing group has no match.

```

- Quantifiers can match multiple occurrences of a pattern that is defined by a capturing group. In this case, the `Value` and `Length` properties of a `Group` object contain information only about the last captured substring. For example, the following regular expression matches a single sentence that ends in a period. It uses two grouping constructs: The first captures individual words along with a white-space character; the second captures individual words. As the output from the example shows, although the regular expression succeeds in capturing an entire sentence, the second capturing group captures only the last word.

```

using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string pattern = @"\b((\w+)\s?)+\." ;
        string input = "This is a sentence. This is another sentence.";
        Match match = Regex.Match(input, pattern);
        if (match.Success)
        {
            Console.WriteLine("Match: " + match.Value);
            Console.WriteLine("Group 2: " + match.Groups[2].Value);
        }
    }
}

// The example displays the following output:
//      Match: This is a sentence.
//      Group 2: sentence

```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim pattern As String = "\b((\w+)\s?)+\."
        Dim input As String = "This is a sentence. This is another sentence."
        Dim match As Match = Regex.Match(input, pattern)
        If match.Success Then
            Console.WriteLine("Match: " + match.Value)
            Console.WriteLine("Group 2: " + match.Groups(2).Value)
        End If
    End Sub
End Module

' The example displays the following output:
'      Match: This is a sentence.
'      Group 2: sentence

```

[Back to top](#)

The Capture Collection

The [Group](#) object contains information only about the last capture. However, the entire set of captures made by a capturing group is still available from the [CaptureCollection](#) object that is returned by the [Group.Captures](#) property. Each member of the collection is a [Capture](#) object that represents a capture made by that capturing group, in the order in which they were captured (and, therefore, in the order in which the captured strings were matched from left to right in the input string). You can retrieve individual [Capture](#) objects from the collection in either of two ways:

- By iterating through the collection using a construct such as `foreach` (in C#) or `For Each` (in Visual Basic).
- By using the [CaptureCollection.Item\[\]](#) property to retrieve a specific object by index. The [Item\[\]](#) property is the [CaptureCollection](#) object's default property (in Visual Basic) or indexer (in C#).

If a quantifier is not applied to a capturing group, the [CaptureCollection](#) object contains a single [Capture](#) object that is of little interest, because it provides information about the same match as its [Group](#) object. If a quantifier is applied to a capturing group, the [CaptureCollection](#) object contains all captures made by the capturing group, and the last member of the collection represents the same capture as the [Group](#) object.

For example, if you use the regular expression pattern `((a(b))c)+` (where the `+` quantifier specifies one or more matches) to capture matches from the string "abcabcabc", the [CaptureCollection](#) object for each [Group](#) object contains three members.

```
using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string pattern = "((a(b))c)+";
        string input = "abcabcabc";

        Match match = Regex.Match(input, pattern);
        if (match.Success)
        {
            Console.WriteLine("Match: '{0}' at position {1}",
                match.Value, match.Index);
            GroupCollection groups = match.Groups;
            for (int ctr = 0; ctr < groups.Count; ctr++) {
                Console.WriteLine("    Group {0}: '{1}' at position {2}",
                    ctr, groups[ctr].Value, groups[ctr].Index);
                CaptureCollection captures = groups[ctr].Captures;
                for (int ctr2 = 0; ctr2 < captures.Count; ctr2++) {
                    Console.WriteLine("        Capture {0}: '{1}' at position {2}",
                        ctr2, captures[ctr2].Value, captures[ctr2].Index);
                }
            }
        }
    }
}

// The example displays the following output:
//      Match: 'abcabcabc' at position 0
//          Group 0: 'abcabcabc' at position 0
//              Capture 0: 'abcabcabc' at position 0
//          Group 1: 'abc' at position 6
//              Capture 0: 'abc' at position 0
//              Capture 1: 'abc' at position 3
//              Capture 2: 'abc' at position 6
//          Group 2: 'ab' at position 6
//              Capture 0: 'ab' at position 0
//              Capture 1: 'ab' at position 3
//              Capture 2: 'ab' at position 6
//          Group 3: 'b' at position 7
//              Capture 0: 'b' at position 1
//              Capture 1: 'b' at position 4
//              Capture 2: 'b' at position 7
```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim pattern As String = "((a(b))c)+"
        Dim input As String = "abcabcabc"

        Dim match As Match = Regex.Match(input, pattern)
        If match.Success Then
            Console.WriteLine("Match: '{0}' at position {1}", _
                match.Value, match.Index)
            Dim groups As GroupCollection = match.Groups
            For ctr As Integer = 0 To groups.Count - 1
                Console.WriteLine("    Group {0}: '{1}' at position {2}", _
                    ctr, groups(ctr).Value, groups(ctr).Index)
                Dim captures As CaptureCollection = groups(ctr).Captures
                For ctr2 As Integer = 0 To captures.Count - 1
                    Console.WriteLine("        Capture {0}: '{1}' at position {2}", _
                        ctr2, captures(ctr2).Value, captures(ctr2).Index)
                Next
            Next
        End If
    End Sub
End Module
' The example displays the following output:
'     Match: 'abcabcabc' at position 0
'         Group 0: 'abcabcabc' at position 0
'             Capture 0: 'abcabcabc' at position 0
'         Group 1: 'abc' at position 6
'             Capture 0: 'abc' at position 0
'             Capture 1: 'abc' at position 3
'             Capture 2: 'abc' at position 6
'         Group 2: 'ab' at position 6
'             Capture 0: 'ab' at position 0
'             Capture 1: 'ab' at position 3
'             Capture 2: 'ab' at position 6
'         Group 3: 'b' at position 7
'             Capture 0: 'b' at position 1
'             Capture 1: 'b' at position 4
'             Capture 2: 'b' at position 7

```

The following example uses the regular expression `(Abc)+` to find one or more consecutive runs of the string "Abc" in the string "XYZAbcAbcAbcXYZAbcAb". The example illustrates the use of the [Group.Captures](#) property to return multiple groups of captured substrings.

```
int counter;
Match m;
CaptureCollection cc;
GroupCollection gc;

// Look for groupings of "Abc".
var r = new Regex("(Abc)+");
// Define the string to search.
m = r.Match("XYZAbcAbcAbcXYZAbcAb");
gc = m.Groups;

// Display the number of groups.
Console.WriteLine("Captured groups = " + gc.Count.ToString());

// Loop through each group.
for (int i = 0; i < gc.Count; i++)
{
    cc = gc[i].Captures;
    counter = cc.Count;

    // Display the number of captures in this group.
    Console.WriteLine("Captures count = " + counter.ToString());

    // Loop through each capture in the group.
    for (int ii = 0; ii < counter; ii++)
    {
        // Display the capture and its position.
        Console.WriteLine(cc[ii] + " Starts at character " +
            cc[ii].Index);
    }
}

// The example displays the following output:
//     Captured groups = 2
//     Captures count = 1
//     AbcAbcAbc Starts at character 3
//     Captures count = 3
//     Abc Starts at character 3
//     Abc Starts at character 6
//     Abc Starts at character 9
```

```

Dim counter As Integer
Dim m As Match
Dim cc As CaptureCollection
Dim gc As GroupCollection

' Look for groupings of "Abc".
Dim r As New Regex("(Abc)+")
' Define the string to search.
m = r.Match("XYZAbcAbcAbcXYZAbcAb")
gc = m.Groups

' Display the number of groups.
Console.WriteLine("Captured groups = " & gc.Count.ToString())

' Loop through each group.
Dim i, ii As Integer
For i = 0 To gc.Count - 1
    cc = gc(i).Captures
    counter = cc.Count

    ' Display the number of captures in this group.
    Console.WriteLine("Captures count = " & counter.ToString())

    ' Loop through each capture in the group.
    For ii = 0 To counter - 1
        ' Display the capture and its position.
        Console.WriteLine(cc(ii).ToString() _
            & "    Starts at character " & cc(ii).Index.ToString())
    Next ii
Next i
' The example displays the following output:
'     Captured groups = 2
'     Captures count = 1
'     AbcAbcAbc    Starts at character 3
'     Captures count = 3
'     Abc    Starts at character 3
'     Abc    Starts at character 6
'     Abc    Starts at character 9

```

[Back to top](#)

The Individual Capture

The [Capture](#) class contains the results from a single subexpression capture. The [Capture.Value](#) property contains the matched text, and the [Capture.Index](#) property indicates the zero-based position in the input string at which the matched substring begins.

The following example parses an input string for the temperature of selected cities. A comma (",") is used to separate a city and its temperature, and a semicolon (";") is used to separate each city's data. The entire input string represents a single match. In the regular expression pattern `((\w+(\s\w+)*),(\d+);)+`, which is used to parse the string, the city name is assigned to the second capturing group, and the temperature is assigned to the fourth capturing group.

```

using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string input = "Miami,78;Chicago,62;New York,67;San Francisco,59;Seattle,58;";
        string pattern = @"((\w+(\s\w+)*),(\d+);)+";
        Match match = Regex.Match(input, pattern);
        if (match.Success)
        {
            Console.WriteLine("Current temperatures:");
            for (int ctr = 0; ctr < match.Groups[2].Captures.Count; ctr++)
                Console.WriteLine("{0,-20} {1,3}", match.Groups[2].Captures[ctr].Value,
                    match.Groups[4].Captures[ctr].Value);
        }
    }
}

// The example displays the following output:
//      Current temperatures:
//      Miami          78
//      Chicago        62
//      New York       67
//      San Francisco  59
//      Seattle         58

```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim input As String = "Miami,78;Chicago,62;New York,67;San Francisco,59;Seattle,58;"
        Dim pattern As String = @"((\w+(\s\w+)*),(\d+);)+"
        Dim match As Match = Regex.Match(input, pattern)
        If match.Success Then
            Console.WriteLine("Current temperatures:")
            For ctr As Integer = 0 To match.Groups(2).Captures.Count - 1
                Console.WriteLine("{0,-20} {1,3}", match.Groups(2).Captures(ctr).Value, _
                    match.Groups(4).Captures(ctr).Value)
            Next
        End If
    End Sub
End Module

' The example displays the following output:
'      Current temperatures:
'      Miami          78
'      Chicago        62
'      New York       67
'      San Francisco  59
'      Seattle         58

```

The regular expression is defined as shown in the following table.

PATTERN	DESCRIPTION
\w+	Match one or more word characters.
(\s\w+)*	Match zero or more occurrences of a white-space character followed by one or more word characters. This pattern matches multi-word city names. This is the third capturing group.

PATTERN	DESCRIPTION
(\w+(\s\w+)*)	Match one or more word characters followed by zero or more occurrences of a white-space character and one or more word characters. This is the second capturing group.
,	Match a comma.
(\d+)	Match one or more digits. This is the fourth capturing group.
;	Match a semicolon.
((\w+(\s\w+)*),(\d+);)+	Match the pattern of a word followed by any additional words followed by a comma, one or more digits, and a semicolon, one or more times. This is the first capturing group.

See also

- [System.Text.RegularExpressions](#)
- [.NET Regular Expressions](#)
- [Regular Expression Language - Quick Reference](#)

Details of regular expression behavior

9/20/2022 • 18 minutes to read • [Edit Online](#)

The .NET regular expression engine is a backtracking regular expression matcher that incorporates a traditional Nondeterministic Finite Automaton (NFA) engine such as that used by Perl, Python, Emacs, and Tcl. This distinguishes it from faster, but more limited, pure regular expression Deterministic Finite Automaton (DFA) engines such as those found in awk, egrep, or lex. This also distinguishes it from standardized, but slower, POSIX NFAs. The following section describes the three types of regular expression engines, and explains why regular expressions in .NET are implemented by using a traditional NFA engine.

Benefits of the NFA engine

When DFA engines perform pattern matching, their processing order is driven by the input string. The engine begins at the beginning of the input string and proceeds sequentially to determine whether the next character matches the regular expression pattern. They can guarantee to match the longest string possible. Because they never test the same character twice, DFA engines do not support backtracking. However, because a DFA engine contains only finite state, it cannot match a pattern with backreferences, and because it does not construct an explicit expansion, it cannot capture subexpressions.

Unlike DFA engines, when traditional NFA engines perform pattern matching, their processing order is driven by the regular expression pattern. As it processes a particular language element, the engine uses greedy matching; that is, it matches as much of the input string as it possibly can. But it also saves its state after successfully matching a subexpression. If a match eventually fails, the engine can return to a saved state so it can try additional matches. This process of abandoning a successful subexpression match so that later language elements in the regular expression can also match is known as *backtracking*. NFA engines use backtracking to test all possible expansions of a regular expression in a specific order and accept the first match. Because a traditional NFA engine constructs a specific expansion of the regular expression for a successful match, it can capture subexpression matches and matching backreferences. However, because a traditional NFA backtracks, it can visit the same state multiple times if it arrives at the state over different paths. As a result, it can run exponentially slowly in the worst case. Because a traditional NFA engine accepts the first match it finds, it can also leave other (possibly longer) matches undiscovered.

POSIX NFA engines are like traditional NFA engines, except that they continue to backtrack until they can guarantee that they have found the longest match possible. As a result, a POSIX NFA engine is slower than a traditional NFA engine, and when you use a POSIX NFA engine, you cannot favor a shorter match over a longer one by changing the order of the backtracking search.

Traditional NFA engines are favored by programmers because they offer greater control over string matching than either DFA or POSIX NFA engines. Although, in the worst case, they can run slowly, you can steer them to find matches in linear or polynomial time by using patterns that reduce ambiguities and limit backtracking. In other words, although NFA engines trade performance for power and flexibility, in most cases they offer good to acceptable performance if a regular expression is well written and avoids cases in which backtracking degrades performance exponentially.

NOTE

For information about the performance penalty caused by excessive backtracking and ways to craft a regular expression to work around them, see [Backtracking](#).

.NET engine capabilities

To take advantage of the benefits of a traditional NFA engine, the .NET regular expression engine includes a complete set of constructs to enable programmers to steer the backtracking engine. These constructs can be used to find matches faster or to favor specific expansions over others.

Other features of the .NET regular expression engine include the following:

- Lazy quantifiers: `??`, `*?`, `+?`, `{ n }`, `{ n, m }`. These constructs tell the backtracking engine to search the minimum number of repetitions first. In contrast, ordinary greedy quantifiers try to match the maximum number of repetitions first. The following example illustrates the difference between the two. A regular expression matches a sentence that ends in a number, and a capturing group is intended to extract that number. The regular expression `.+(\\d+)\\.` includes the greedy quantifier `.+`, which causes the regular expression engine to capture only the last digit of the number. In contrast, the regular expression `.+?(\\d+)\\.` includes the lazy quantifier `.+?`, which causes the regular expression engine to capture the entire number.

```
using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string greedyPattern = @".(\\d+)\\.";
        string lazyPattern = @".+?(\\d+)\\.";
        string input = "This sentence ends with the number 107325.";
        Match match;

        // Match using greedy quantifier .
        match = Regex.Match(input, greedyPattern);
        if (match.Success)
            Console.WriteLine("Number at end of sentence (greedy): {0}",
                match.Groups[1].Value);
        else
            Console.WriteLine("{0} finds no match.", greedyPattern);

        // Match using lazy quantifier .+?.
        match = Regex.Match(input, lazyPattern);
        if (match.Success)
            Console.WriteLine("Number at end of sentence (lazy): {0}",
                match.Groups[1].Value);
        else
            Console.WriteLine("{0} finds no match.", lazyPattern);
    }
}

// The example displays the following output:
//      Number at end of sentence (greedy): 5
//      Number at end of sentence (lazy): 107325
```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim greedyPattern As String = ".+(\d+)."
        Dim lazyPattern As String = ".+?(d+)."
        Dim input As String = "This sentence ends with the number 107325."
        Dim match As Match

        ' Match using greedy quantifier .+.
        match = Regex.Match(input, greedyPattern)
        If match.Success Then
            Console.WriteLine("Number at end of sentence (greedy): {0}",
                match.Groups(1).Value)
        Else
            Console.WriteLine("{0} finds no match.", greedyPattern)
        End If

        ' Match using lazy quantifier .+?.
        match = Regex.Match(input, lazyPattern)
        If match.Success Then
            Console.WriteLine("Number at end of sentence (lazy): {0}",
                match.Groups(1).Value)
        Else
            Console.WriteLine("{0} finds no match.", lazyPattern)
        End If
    End Sub
End Module

' The example displays the following output:
'     Number at end of sentence (greedy): 5
'     Number at end of sentence (lazy): 107325

```

The greedy and lazy versions of this regular expression are defined as shown in the following table:

PATTERN	DESCRIPTION
.+ (greedy quantifier)	Match at least one occurrence of any character. This causes the regular expression engine to match the entire string, and then to backtrack as needed to match the remainder of the pattern.
.+? (lazy quantifier)	Match at least one occurrence of any character, but match as few as possible.
(\d+)	Match at least one numeric character, and assign it to the first capturing group.
\.	Match a period.

For more information about lazy quantifiers, see [Quantifiers](#).

- Positive lookahead: `(?= subexpression)`. This feature allows the backtracking engine to return to the same spot in the text after matching a subexpression. It is useful for searching throughout the text by verifying multiple patterns that start from the same position. It also allows the engine to verify that a substring exists at the end of the match without including the substring in the matched text. The following example uses positive lookahead to extract the words in a sentence that are not followed by punctuation symbols.

```

using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string pattern = @"\b[A-Z]+\b(?=\P{P})";
        string input = "If so, what comes next?";
        foreach (Match match in Regex.Matches(input, pattern, RegexOptions.IgnoreCase))
            Console.WriteLine(match.Value);
    }
}
// The example displays the following output:
//      If
//      what
//      comes

```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim pattern As String = "\b[A-Z]+\b(?=\P{P})"
        Dim input As String = "If so, what comes next?"
        For Each match As Match In Regex.Matches(input, pattern, RegexOptions.IgnoreCase)
            Console.WriteLine(match.Value)
        Next
    End Sub
End Module
' The example displays the following output:
'      If
'      what
'      comes

```

The regular expression `\b[A-Z]+\b(?=\P{P})` is defined as shown in the following table.

PATTERN	DESCRIPTION
<code>\b</code>	Begin the match at a word boundary.
<code>[A-Z]+</code>	Match any alphabetic character one or more times. Because the <code>Regex.Matches</code> method is called with the <code>RegexOptions.IgnoreCase</code> option, the comparison is case-insensitive.
<code>\b</code>	End the match at a word boundary.
<code>(?=\P{P})</code>	Look ahead to determine whether the next character is a punctuation symbol. If it is not, the match succeeds.

For more information about positive lookahead assertions, see [Grouping Constructs](#).

- Negative lookahead: `(?! subexpression)`. This feature adds the ability to match an expression only if a subexpression fails to match. This is powerful for pruning a search, because it is often simpler to provide an expression for a case that should be eliminated than an expression for cases that must be included. For example, it is difficult to write an expression for words that do not begin with "non". The following example uses negative lookahead to exclude them.

```

using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string pattern = @"\b(?!non)\w+\b";
        string input = "Nonsense is not always non-functional.";
        foreach (Match match in Regex.Matches(input, pattern, RegexOptions.IgnoreCase))
            Console.WriteLine(match.Value);
    }
}
// The example displays the following output:
//     is
//     not
//     always
//     functional

```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim pattern As String = "\b(?!non)\w+\b"
        Dim input As String = "Nonsense is not always non-functional."
        For Each match As Match In Regex.Matches(input, pattern, RegexOptions.IgnoreCase)
            Console.WriteLine(match.Value)
        Next
    End Sub
End Module
' The example displays the following output:
'     is
'     not
'     always
'     functional

```

The regular expression pattern `\b(?!non)\w+\b` is defined as shown in the following table.

PATTERN	DESCRIPTION
<code>\b</code>	Begin the match at a word boundary.
<code>(?!non)</code>	Look ahead to ensure that the current string does not begin with "non". If it does, the match fails.
<code>(\w+)</code>	Match one or more word characters.
<code>\b</code>	End the match at a word boundary.

For more information about negative lookahead assertions, see [Grouping Constructs](#).

- Conditional evaluation: `(?(<expression>) yes | nc)` and `(?(<name>) yes | no)`, where *expression* is a subexpression to match, *name* is the name of a capturing group, *yes* is the string to match if *expression* is matched or *name* is a valid, non-empty captured group, and *no* is the subexpression to match if *expression* is not matched or *name* is not a valid, non-empty captured group. This feature allows the engine to search by using more than one alternate pattern, depending on the result of a previous subexpression match or the result of a zero-width assertion. This allows a more powerful form of backreference that permits, for example, matching a subexpression based on whether a previous subexpression was matched. The regular expression in the following example matches paragraphs that

are intended for both public and internal use. Paragraphs intended only for internal use begin with a <PRIVATE> tag. The regular expression pattern

`^(?<Pvt>\<PRIVATE\>\s)?(?(Pvt)((\w+\p{P}\?\s)+)|((\w+\p{P}\?\s)+))\r?$` uses conditional evaluation to assign the contents of paragraphs intended for public and for internal use to separate capturing groups. These paragraphs can then be handled differently.

```
using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string input = "<PRIVATE> This is not for public consumption." + Environment.NewLine +
                      "But this is for public consumption." + Environment.NewLine +
                      "<PRIVATE> Again, this is confidential.\n";
        string pattern = @"^(?<Pvt>\<PRIVATE\>\s)?(?(Pvt)((\w+\p{P}\?\s)+)|((\w+\p{P}\?\s)+))\r?$";
        string publicDocument = null, privateDocument = null;

        foreach (Match match in Regex.Matches(input, pattern, RegexOptions.Multiline))
        {
            if (match.Groups[1].Success)
            {
                privateDocument += match.Groups[1].Value + "\n";
            }
            else
            {
                publicDocument += match.Groups[3].Value + "\n";
                privateDocument += match.Groups[3].Value + "\n";
            }
        }

        Console.WriteLine("Private Document:");
        Console.WriteLine(privateDocument);
        Console.WriteLine("Public Document:");
        Console.WriteLine(publicDocument);
    }
}

// The example displays the following output:
// Private Document:
// This is not for public consumption.
// But this is for public consumption.
// Again, this is confidential.
//
// Public Document:
// But this is for public consumption.
```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim input As String = "<PRIVATE> This is not for public consumption." + vbCrLf +
                            "But this is for public consumption." + vbCrLf +
                            "<PRIVATE> Again, this is confidential." + vbCrLf
        Dim pattern As String = "^(?<Pvt>\<PRIVATE\>\s)?(?:Pvt)((\w+\p{P}?\s+)|((\w+\p{P}?\s+))\r?
$"
        Dim publicDocument As String = Nothing
        Dim privateDocument As String = Nothing

        For Each match As Match In Regex.Matches(input, pattern, RegexOptions.Multiline)
            If match.Groups(1).Success Then
                privateDocument += match.Groups(1).Value + vbCrLf
            Else
                publicDocument += match.Groups(3).Value + vbCrLf
                privateDocument += match.Groups(3).Value + vbCrLf
            End If
        Next

        Console.WriteLine("Private Document:")
        Console.WriteLine(privateDocument)
        Console.WriteLine("Public Document:")
        Console.WriteLine(publicDocument)
    End Sub
End Module
' The example displays the following output:
'   Private Document:
'   This is not for public consumption.
'   But this is for public consumption.
'   Again, this is confidential.
'
'   Public Document:
'   But this is for public consumption.

```

The regular expression pattern is defined as shown in the following table.

PATTERN	DESCRIPTION
^	Begin the match at the beginning of a line.
(?<Pvt>\<PRIVATE\>\s)?	Match zero or one occurrence of the string <PRIVATE> followed by a white-space character. Assign the match to a capturing group named Pvt .
(?<Pvt>((\w+\p{P}?\s+))	If the Pvt capturing group exists, match one or more occurrences of one or more word characters followed by zero or one punctuation separator followed by a white-space character. Assign the substring to the first capturing group.
((\w+\p{P}?\s+))	If the Pvt capturing group does not exist, match one or more occurrences of one or more word characters followed by zero or one punctuation separator followed by a white-space character. Assign the substring to the third capturing group.
\r?\$	Match the end of a line or the end of the string.

For more information about conditional evaluation, see [Alternation Constructs](#).

- Balancing group definitions: `(?< name1 - name2 > subexpression)`. This feature allows the regular expression engine to keep track of nested constructs such as parentheses or opening and closing brackets. For an example, see [Grouping Constructs](#).
- Atomic groups: `(?> subexpression)`. This feature allows the backtracking engine to guarantee that a subexpression matches only the first match found for that subexpression, as if the expression were running independent of its containing expression. If you do not use this construct, backtracking searches from the larger expression can change the behavior of a subexpression. For example, the regular expression `(a+)\w` matches one or more "a" characters, along with a word character that follows the sequence of "a" characters, and assigns the sequence of "a" characters to the first capturing group. However, if the final character of the input string is also an "a", it is matched by the `\w` language element and is not included in the captured group.

```
using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string[] inputs = { "aaaaa", "aaaaab" };
        string backtrackingPattern = @"(a+)\w";
        Match match;

        foreach (string input in inputs)
        {
            Console.WriteLine("Input: {0}", input);
            match = Regex.Match(input, backtrackingPattern);
            Console.WriteLine("    Pattern: {0}", backtrackingPattern);
            if (match.Success)
            {
                Console.WriteLine("        Match: {0}", match.Value);
                Console.WriteLine("        Group 1: {0}", match.Groups[1].Value);
            }
            else
            {
                Console.WriteLine("        Match failed.");
            }
        }
        Console.WriteLine();
    }
}
// The example displays the following output:
//      Input: aaaaa
//      Pattern: (a+)\w
//      Match: aaaaa
//      Group 1: aaaa
//      Input: aaaaab
//      Pattern: (a+)\w
//      Match: aaaaab
//      Group 1: aaaaa
```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim inputs() As String = {"aaaaa", "aaaaaab"}
        Dim backtrackingPattern As String = "(a+)\w"
        Dim match As Match

        For Each input As String In inputs
            Console.WriteLine("Input: {0}", input)
            match = Regex.Match(input, backtrackingPattern)
            Console.WriteLine("    Pattern: {0}", backtrackingPattern)
            If match.Success Then
                Console.WriteLine("        Match: {0}", match.Value)
                Console.WriteLine("        Group 1: {0}", match.Groups(1).Value)
            Else
                Console.WriteLine("        Match failed.")
            End If
        Next
        Console.WriteLine()
    End Sub
End Module
' The example displays the following output:
'     Input: aaaaa
'         Pattern: (a+)\w
'         Match: aaaaa
'         Group 1: aaaa
'     Input: aaaaab
'         Pattern: (a+)\w
'         Match: aaaaab
'         Group 1: aaaaa

```

The regular expression `((?>a+))\w` prevents this behavior. Because all consecutive "a" characters are matched without backtracking, the first capturing group includes all consecutive "a" characters. If the "a" characters are not followed by at least one more character other than "a", the match fails.

```
using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string[] inputs = { "aaaaa", "aaaaaab" };
        string nonbacktrackingPattern = @"((?>a+))\w";
        Match match;

        foreach (string input in inputs)
        {
            Console.WriteLine("Input: {0}", input);
            match = Regex.Match(input, nonbacktrackingPattern);
            Console.WriteLine("    Pattern: {0}", nonbacktrackingPattern);
            if (match.Success)
            {
                Console.WriteLine("        Match: {0}", match.Value);
                Console.WriteLine("        Group 1: {0}", match.Groups[1].Value);
            }
            else
            {
                Console.WriteLine("        Match failed.");
            }
        }
        Console.WriteLine();
    }
}

// The example displays the following output:
//      Input: aaaaa
//      Pattern: ((?>a+))\w
//      Match failed.
//      Input: aaaaab
//      Pattern: ((?>a+))\w
//      Match: aaaaab
//      Group 1: aaaaa
```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim inputs() As String = {"aaaaa", "aaaaaab"}
        Dim nonbacktrackingPattern As String = "((?>a+))\w"
        Dim match As Match

        For Each input As String In inputs
            Console.WriteLine("Input: {0}", input)
            match = Regex.Match(input, nonbacktrackingPattern)
            Console.WriteLine("    Pattern: {0}", nonbacktrackingPattern)
            If match.Success Then
                Console.WriteLine("        Match: {0}", match.Value)
                Console.WriteLine("        Group 1: {0}", match.Groups(1).Value)
            Else
                Console.WriteLine("        Match failed.")
            End If
        Next
        Console.WriteLine()
    End Sub
End Module
' The example displays the following output:
'     Input: aaaaa
'         Pattern: ((?>a+))\w
'         Match failed.
'     Input: aaaaab
'         Pattern: ((?>a+))\w
'         Match: aaaaab
'         Group 1: aaaa

```

For more information about atomic groups, see [Grouping Constructs](#).

- Right-to-left matching, which is specified by supplying the `RegexOptions.RightToLeft` option to a `Regex` class constructor or static instance matching method. This feature is useful when searching from right to left instead of from left to right, or in cases where it is more efficient to begin a match at the right part of the pattern instead of the left. As the following example illustrates, using right-to-left matching can change the behavior of greedy quantifiers. The example conducts two searches for a sentence that ends in a number. The left-to-right search that uses the greedy quantifier `+` matches one of the six digits in the sentence, whereas the right-to-left search matches all six digits. For a description of the regular expression pattern, see the example that illustrates lazy quantifiers earlier in this section.

```

using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string greedyPattern = @"^.+(\d+)\." ;
        string input = "This sentence ends with the number 107325." ;
        Match match;

        // Match from left-to-right using lazy quantifier .+?.
        match = Regex.Match(input, greedyPattern);
        if (match.Success)
            Console.WriteLine("Number at end of sentence (left-to-right): {0}",
                match.Groups[1].Value);
        else
            Console.WriteLine("{0} finds no match.", greedyPattern);

        // Match from right-to-left using greedy quantifier .+.
        match = Regex.Match(input, greedyPattern, RegexOptions.RightToLeft);
        if (match.Success)
            Console.WriteLine("Number at end of sentence (right-to-left): {0}",
                match.Groups[1].Value);
        else
            Console.WriteLine("{0} finds no match.", greedyPattern);
    }
}

// The example displays the following output:
//      Number at end of sentence (left-to-right): 5
//      Number at end of sentence (right-to-left): 107325

```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim greedyPattern As String = ".+(\d+)\."
        Dim input As String = "This sentence ends with the number 107325."
        Dim match As Match

        ' Match from left-to-right using lazy quantifier .+?.
        match = Regex.Match(input, greedyPattern)
        If match.Success Then
            Console.WriteLine("Number at end of sentence (left-to-right): {0}",
                match.Groups(1).Value)
        Else
            Console.WriteLine("{0} finds no match.", greedyPattern)
        End If

        ' Match from right-to-left using greedy quantifier .+.
        match = Regex.Match(input, greedyPattern, RegexOptions.RightToLeft)
        If match.Success Then
            Console.WriteLine("Number at end of sentence (right-to-left): {0}",
                match.Groups(1).Value)
        Else
            Console.WriteLine("{0} finds no match.", greedyPattern)
        End If
    End Sub
End Module

```

' The example displays the following output:

' Number at end of sentence (left-to-right): 5

' Number at end of sentence (right-to-left): 107325

For more information about right-to-left matching, see [Regular Expression Options](#).

- Positive and negative lookbehind: `(?<= subexpression)` for positive lookbehind, and `(?<! subexpression)` for negative lookbehind. This feature is similar to lookahead, which is discussed earlier in this topic. Because the regular expression engine allows complete right-to-left matching, regular expressions allow unrestricted lookbehinds. Positive and negative lookbehind can also be used to avoid nesting quantifiers when the nested subexpression is a superset of an outer expression. Regular expressions with such nested quantifiers often offer poor performance. For example, the following example verifies that a string begins and ends with an alphanumeric character, and that any other character in the string is one of a larger subset. It forms a portion of the regular expression used to validate email addresses; for more information, see [How to: Verify that Strings Are in Valid Email Format](#).

```
using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string[] inputs = { "jack.sprat", "dog#", "dog#1", "me.myself",
                           "me.myself!" };
        string pattern = @"^([A-Z0-9][^!#$%&.*+/=?^`{}|~\w])*(<=[A-Z0-9])$";
        foreach (string input in inputs)
        {
            if (Regex.IsMatch(input, pattern, RegexOptions.IgnoreCase))
                Console.WriteLine("{0}: Valid", input);
            else
                Console.WriteLine("{0}: Invalid", input);
        }
    }
}

// The example displays the following output:
//      jack.sprat: Valid
//      dog#: Invalid
//      dog#1: Valid
//      me.myself: Valid
//      me.myself!: Invalid
```

```
Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim inputs() As String = {"jack.sprat", "dog#", "dog#1", "me.myself",
                                  "me.myself!"}
        Dim pattern As String = "([A-Z0-9][^!#$%&.*+/=?^`{}|~\w])*(<=[A-Z0-9])$"
        For Each input As String In inputs
            If Regex.IsMatch(input, pattern, RegexOptions.IgnoreCase) Then
                Console.WriteLine("{0}: Valid", input)
            Else
                Console.WriteLine("{0}: Invalid", input)
            End If
        Next
    End Sub
End Module

' The example displays the following output:
'      jack.sprat: Valid
'      dog#: Invalid
'      dog#1: Valid
'      me.myself: Valid
'      me.myself!: Invalid
```

The regular expression `^([A-Z0-9][^!#$%&.*+/=?^`{}|~\w])*(<=[A-Z0-9])$` is defined as shown in the following table.

PATTERN	DESCRIPTION
<code>^</code>	Begin the match at the beginning of the string.
<code>[A-Z0-9]</code>	Match any numeric or alphanumeric character. (The comparison is case-insensitive.)
<code>([-!#\$%&'.*+/=?^`{} ~\w])*</code>	Match zero or more occurrences of any word character, or any of the following characters: <code>-</code> , <code>!</code> , <code>#</code> , <code>\$</code> , <code>%</code> , <code>&</code> , <code>'</code> , <code>.</code> , <code>*</code> , <code>/</code> , <code>=</code> , <code>?</code> , <code>^</code> , <code>`</code> , <code>{</code> , <code>}</code> , <code> </code> , or <code>~</code> .
<code>(?<=[A-Z0-9])</code>	Look behind to the previous character, which must be numeric or alphanumeric. (The comparison is case-insensitive.)
<code>\$</code>	End the match at the end of the string.

For more information about positive and negative lookbehind, see [Grouping Constructs](#).

Related articles

TITLE	DESCRIPTION
Backtracking	Provides information about how regular expression backtracking branches to find alternative matches.
Compilation and Reuse	Provides information about compiling and reusing regular expressions to increase performance.
Thread Safety	Provides information about regular expression thread safety and explains when you should synchronize access to regular expression objects.
.NET Regular Expressions	Provides an overview of the programming language aspect of regular expressions.
The Regular Expression Object Model	Provides information and code examples illustrating how to use the regular expression classes.
Regular Expression Language - Quick Reference	Provides information about the set of characters, operators, and constructs that you can use to define regular expressions.

Reference

- [System.Text.RegularExpressions](#)

Backtracking in Regular Expressions

9/20/2022 • 22 minutes to read • [Edit Online](#)

Backtracking occurs when a regular expression pattern contains optional [quantifiers](#) or [alternation constructs](#), and the regular expression engine returns to a previous saved state to continue its search for a match.

Backtracking is central to the power of regular expressions; it makes it possible for expressions to be powerful and flexible, and to match very complex patterns. At the same time, this power comes at a cost. Backtracking is often the single most important factor that affects the performance of the regular expression engine.

Fortunately, the developer has control over the behavior of the regular expression engine and how it uses backtracking. This topic explains how backtracking works and how it can be controlled.

NOTE

In general, a Nondeterministic Finite Automaton (NFA) engine like .NET regular expression engine places the responsibility for crafting efficient, fast regular expressions on the developer.

Linear Comparison Without Backtracking

If a regular expression pattern has no optional quantifiers or alternation constructs, the regular expression engine executes in linear time. That is, after the regular expression engine matches the first language element in the pattern with text in the input string, it tries to match the next language element in the pattern with the next character or group of characters in the input string. This continues until the match either succeeds or fails. In either case, the regular expression engine advances by one character at a time in the input string.

The following example provides an illustration. The regular expression `e{2}\w\b` looks for two occurrences of the letter "e" followed by any word character followed by a word boundary.

```
using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string input = "needing a reed";
        string pattern = @"e{2}\w\b";
        foreach (Match match in Regex.Matches(input, pattern))
            Console.WriteLine("{0} found at position {1}",
                match.Value, match.Index);
    }
}
// The example displays the following output:
//     eed found at position 11
```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim input As String = "needing a reed"
        Dim pattern As String = "e{2}\w\b"
        For Each match As Match In Regex.Matches(input, pattern)
            Console.WriteLine("{0} found at position {1}", _
                match.Value, match.Index)
        Next
    End Sub
End Module
' The example displays the following output:
'     eed found at position 11

```

Although this regular expression includes the quantifier `{2}`, it is evaluated in a linear manner. The regular expression engine does not backtrack because `{2}` is not an optional quantifier; it specifies an exact number and not a variable number of times that the previous subexpression must match. As a result, the regular expression engine tries to match the regular expression pattern with the input string as shown in the following table.

OPERATION	POSITION IN PATTERN	POSITION IN STRING	RESULT
1	e	"needing a reed" (index 0)	No match.
2	e	"eeding a reed" (index 1)	Possible match.
3	e{2}	"eding a reed" (index 2)	Possible match.
4	\w	"ding a reed" (index 3)	Possible match.
5	\b	"ing a reed" (index 4)	Possible match fails.
6	e	"eding a reed" (index 2)	Possible match.
7	e{2}	"ding a reed" (index 3)	Possible match fails.
8	e	"ding a reed" (index 3)	Match fails.
9	e	"ing a reed" (index 4)	No match.
10	e	"ng a reed" (index 5)	No match.
11	e	"g a reed" (index 6)	No match.
12	e	" a reed" (index 7)	No match.
13	e	"a reed" (index 8)	No match.
14	e	" reed" (index 9)	No match.
15	e	"reed" (index 10)	No match
16	e	"eed" (index 11)	Possible match.

OPERATION	POSITION IN PATTERN	POSITION IN STRING	RESULT
17	e{2}	"ed" (index 12)	Possible match.
18	\w	"d" (index 13)	Possible match.
19	\b	"" (index 14)	Match.

If a regular expression pattern includes no optional quantifiers or alternation constructs, the maximum number of comparisons required to match the regular expression pattern with the input string is roughly equivalent to the number of characters in the input string. In this case, the regular expression engine uses 19 comparisons to identify possible matches in this 13-character string. In other words, the regular expression engine runs in near-linear time if it contains no optional quantifiers or alternation constructs.

Backtracking with Optional Quantifiers or Alternation Constructs

When a regular expression includes optional quantifiers or alternation constructs, the evaluation of the input string is no longer linear. Pattern matching with an NFA engine is driven by the language elements in the regular expression and not by the characters to be matched in the input string. Therefore, the regular expression engine tries to fully match optional or alternative subexpressions. When it advances to the next language element in the subexpression and the match is unsuccessful, the regular expression engine can abandon a portion of its successful match and return to an earlier saved state in the interest of matching the regular expression as a whole with the input string. This process of returning to a previous saved state to find a match is known as backtracking.

For example, consider the regular expression pattern `.*(es)`, which matches the characters "es" and all the characters that precede it. As the following example shows, if the input string is "Essential services are provided by regular expressions.", the pattern matches the whole string up to and including the "es" in "expressions".

```
using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string input = "Essential services are provided by regular expressions.";
        string pattern = ".*(es)";
        Match m = Regex.Match(input, pattern, RegexOptions.IgnoreCase);
        if (m.Success) {
            Console.WriteLine("'{0}' found at position {1}",
                m.Value, m.Index);
            Console.WriteLine("'es' found at position {0}",
                m.Groups[1].Index);
        }
    }
}
// 'Essential services are provided by regular expres' found at position 0
// 'es' found at position 47
```

```

Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim input As String = "Essential services are provided by regular expressions."
        Dim pattern As String = ".*(es)"
        Dim m As Match = Regex.Match(input, pattern, RegexOptions.IgnoreCase)
        If m.Success Then
            Console.WriteLine("'{0}' found at position {1}", _
                m.Value, m.Index)
            Console.WriteLine("'{1}' found at position {0}", _
                m.Groups(1).Index)
        End If
    End Sub
End Module
'      'Essential services are provided by regular expr' found at position 0
'      'es' found at position 47

```

To do this, the regular expression engine uses backtracking as follows:

- It matches the `.*` (which matches zero, one, or more occurrences of any character) with the whole input string.
- It attempts to match "e" in the regular expression pattern. However, the input string has no remaining characters available to match.
- It backtracks to its last successful match, "Essential services are provided by regular expressions", and attempts to match "e" with the period at the end of the sentence. The match fails.
- It continues to backtrack to a previous successful match one character at a time until the tentatively matched substring is "Essential services are provided by regular expr". It then compares the "e" in the pattern to the second "e" in "expressions" and finds a match.
- It compares "s" in the pattern to the "s" that follows the matched "e" character (the first "s" in "expressions"). The match is successful.

When you use backtracking, matching the regular expression pattern with the input string, which is 55 characters long, requires 67 comparison operations. Generally, if a regular expression pattern has a single alternation construct or a single optional quantifier, the number of comparison operations required to match the pattern is more than twice the number of characters in the input string.

Backtracking with Nested Optional Quantifiers

The number of comparison operations required to match a regular expression pattern can increase exponentially if the pattern includes a large number of alternation constructs, if it includes nested alternation constructs, or, most commonly, if it includes nested optional quantifiers. For example, the regular expression pattern `^(a+)+$` is designed to match a complete string that contains one or more "a" characters. The example provides two input strings of identical length, but only the first string matches the pattern. The [System.Diagnostics.Stopwatch](#) class is used to determine how long the match operation takes.

```

using System;
using System.Diagnostics;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string pattern = "^(a+)+$";
        string[] inputs = { "aaaaaa", "aaaaa!" };
        Regex rgx = new Regex(pattern);
        Stopwatch sw;

        foreach (string input in inputs) {
            sw = Stopwatch.StartNew();
            Match match = rgx.Match(input);
            sw.Stop();
            if (match.Success)
                Console.WriteLine("Matched {0} in {1}", match.Value, sw.Elapsed);
            else
                Console.WriteLine("No match found in {0}", sw.Elapsed);
        }
    }
}

```

```

Imports System.Diagnostics
Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim pattern As String = "^(a+)+$"
        Dim inputs() As String = {"aaaaaa", "aaaaa!"}
        Dim rgx As New Regex(pattern)
        Dim sw As Stopwatch

        For Each input As String In inputs
            sw = Stopwatch.StartNew()
            Dim match As Match = rgx.Match(input)
            sw.Stop()
            If match.Success Then
                Console.WriteLine("Matched {0} in {1}", match.Value, sw.Elapsed)
            Else
                Console.WriteLine("No match found in {0}", sw.Elapsed)
            End If
        Next
    End Sub
End Module

```

As the output from the example shows, the regular expression engine took about twice as long to find that an input string did not match the pattern as it did to identify a matching string. This is because an unsuccessful match always represents a worst-case scenario. The regular expression engine must use the regular expression to follow all possible paths through the data before it can conclude that the match is unsuccessful, and the nested parentheses create many additional paths through the data. The regular expression engine concludes that the second string did not match the pattern by doing the following:

- It checks that it was at the beginning of the string, and then matches the first five characters in the string with the pattern `a+`. It then determines that there are no additional groups of "a" characters in the string. Finally, it tests for the end of the string. Because one additional character remains in the string, the match fails. This failed match requires 9 comparisons. The regular expression engine also saves state information from its matches of "a" (which we will call match 1), "aa" (match 2), "aaa" (match 3), and "aaaa" (match 4).

- It returns to the previously saved match 4. It determines that there is one additional "a" character to assign to an additional captured group. Finally, it tests for the end of the string. Because one additional character remains in the string, the match fails. This failed match requires 4 comparisons. So far, a total of 13 comparisons have been performed.
- It returns to the previously saved match 3. It determines that there are two additional "a" characters to assign to an additional captured group. However, the end-of-string test fails. It then returns to match3 and tries to match the two additional "a" characters in two additional captured groups. The end-of-string test still fails. These failed matches require 12 comparisons. So far, a total of 25 comparisons have been performed.

Comparison of the input string with the regular expression continues in this way until the regular expression engine has tried all possible combinations of matches, and then concludes that there is no match. Because of the nested quantifiers, this comparison is an $O(2^n)$ or an exponential operation, where n is the number of characters in the input string. This means that in the worst case, an input string of 30 characters requires approximately 1,073,741,824 comparisons, and an input string of 40 characters requires approximately 1,099,511,627,776 comparisons. If you use strings of these or even greater lengths, regular expression methods can take an extremely long time to complete when they process input that does not match the regular expression pattern.

Controlling Backtracking

Backtracking lets you create powerful, flexible regular expressions. However, as the previous section showed, these benefits may be coupled with unacceptably poor performance. To prevent excessive backtracking, you should define a time-out interval when you instantiate a [Regex](#) object or call a static regular expression matching method. This is discussed in the next section. In addition, .NET supports three regular expression language elements that limit or suppress backtracking and that support complex regular expressions with little or no performance penalty: [atomic groups](#), [lookbehind assertions](#), and [lookahead assertions](#). For more information about each language element, see [Grouping Constructs](#).

Defining a Time-out Interval

Starting with .NET Framework 4.5, you can set a time-out value that represents the longest interval the regular expression engine will search for a single match before it abandons the attempt and throws a [RegexMatchTimeoutException](#) exception. You specify the time-out interval by supplying a [TimeSpan](#) value to the [Regex\(String, RegexOptions, TimeSpan\)](#) constructor for instance regular expressions. In addition, each static pattern matching method has an overload with a [TimeSpan](#) parameter that allows you to specify a time-out value.

If you do not set a time-out value explicitly, the default time-out value is determined as follows:

- By using the application-wide time-out value, if one exists. This can be any time-out value that applies to the application domain in which the [Regex](#) object is instantiated or the static method call is made. You can set the application-wide time-out value by calling the [AppDomain.SetData](#) method to assign the string representation of a [TimeSpan](#) value to the "REGEX_DEFAULT_MATCH_TIMEOUT" property.
- By using the value [InfiniteMatchTimeout](#), if no application-wide time-out value has been set.

By default, the time-out interval is set to [Regex.InfiniteMatchTimeout](#) and the regular expression engine does not time out.

IMPORTANT

We recommend that you always set a time-out interval if your regular expression relies on backtracking.

A [RegexMatchTimeoutException](#) exception indicates that the regular expression engine was unable to find a match within the specified time-out interval but does not indicate why the exception was thrown. The reason

might be excessive backtracking, but it is also possible that the time-out interval was set too low given the system load at the time the exception was thrown. When you handle the exception, you can choose to abandon further matches with the input string or increase the time-out interval and retry the matching operation.

For example, the following code calls the [Regex\(String, RegexOptions, TimeSpan\)](#) constructor to instantiate a [Regex](#) object with a time-out value of one second. The regular expression pattern `(a+)+$`, which matches one or more sequences of one or more "a" characters at the end of a line, is subject to excessive backtracking. If a [RegexMatchTimeoutException](#) is thrown, the example increases the time-out value up to a maximum interval of three seconds. After that, it abandons the attempt to match the pattern.

```
using System;
using System.ComponentModel;
using System.Diagnostics;
using System.Security;
using System.Text.RegularExpressions;
using System.Threading;

public class Example
{
    const int MaxTimeoutInSeconds = 3;

    public static void Main()
    {
        string pattern = @"(a+)+$";      // DO NOT REUSE THIS PATTERN.
        Regex rgx = new Regex(pattern, RegexOptions.IgnoreCase, TimeSpan.FromSeconds(1));
        Stopwatch? sw = null;

        string[] inputs = { "aa", "aaaa>",
                            "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa",
                            "aaaaaaaaaaaaaaaaaaaaa>",
                            "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa>" };

        foreach (var inputValue in inputs)
        {
            Console.WriteLine("Processing {0}", inputValue);
            bool timedOut = false;
            do
            {
                try
                {
                    sw = Stopwatch.StartNew();
                    // Display the result.
                    if (rgx.IsMatch(inputValue))
                    {
                        sw.Stop();
                        Console.WriteLine(@"Valid: '{0}' ({1:ss\.fffff} seconds)",
                                         inputValue, sw.Elapsed);
                    }
                    else
                    {
                        sw.Stop();
                        Console.WriteLine(@"'{0}' is not a valid string. ({1:ss\.fffff} seconds)",
                                         inputValue, sw.Elapsed);
                    }
                }
            }
            catch (RegexMatchTimeoutException e)
            {
                sw.Stop();
                // Display the elapsed time until the exception.
                Console.WriteLine(@"Timeout with '{0}' after {1:ss\.fffff}",
                                 inputValue, sw.Elapsed);
                Thread.Sleep(1500);      // Pause for 1.5 seconds.

                // Increase the timeout interval and retry.
                TimeSpan timeout = e.MatchTimeout.Add(TimeSpan.FromSeconds(1));
                if (timeout.TotalSeconds > MaxTimeoutInSeconds)
```

```

        {
            Console.WriteLine("Maximum timeout interval of {0} seconds exceeded.",
                MaxTimeoutInSeconds);
            timedOut = false;
        }
        else
        {
            Console.WriteLine("Changing the timeout interval to {0}",
                timeout);
            rgx = new Regex(pattern, RegexOptions.IgnoreCase, timeout);
            timedOut = true;
        }
    }
} while (timedOut);
Console.WriteLine();
}
}

// The example displays output like the following :
// Processing aa
// Valid: 'aa' (00.0000779 seconds)
//
// Processing aaaa>
// 'aaaaa>' is not a valid string. (00.00005 seconds)
//
// Processingaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
// Valid: 'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa' (00.0000043 seconds)
//
// Processingaaaaaaaaaaaaaaaaaaaa>
// Timeout with 'aaaaaaaaaaaaaaaaaaaa>' after 01.00469
// Changing the timeout interval to 00:00:02
// Timeout with 'aaaaaaaaaaaaaaaaaaaa>' after 02.01202
// Changing the timeout interval to 00:00:03
// Timeout with 'aaaaaaaaaaaaaaaaaaaa>' after 03.01043
// Maximum timeout interval of 3 seconds exceeded.
//
// Processingaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa>
// Timeout with 'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa>' after 03.01018
// Maximum timeout interval of 3 seconds exceeded.

```

```

Imports System.ComponentModel
Imports System.Diagnostics
Imports System.Security
Imports System.Text.RegularExpressions
Imports System.Threading

Module Example
    Const MaxTimeoutInSeconds As Integer = 3

    Public Sub Main()
        Dim pattern As String = "(a+)+$"      ' DO NOT REUSE THIS PATTERN.
        Dim rgx As New Regex(pattern, RegexOptions.IgnoreCase, TimeSpan.FromSeconds(1))
        Dim sw As Stopwatch = Nothing

        Dim inputs() As String = {"aa", "aaaa>",
                                "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa",
                                "aaaaaaaaaaaaaaaaaaaa>",
                                "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa>"}

        For Each inputValue In inputs
            Console.WriteLine("Processing {0}", inputValue)
            Dim timedOut As Boolean = False
            Do
                Try
                    sw = Stopwatch.StartNew()
                    ' Display the result.
                    If rgx.IsMatch(inputValue) Then

```

```

        sw.Stop()
        Console.WriteLine("Valid: '{0}' ({1:ss\.fffffff} seconds)",
                           inputValue, sw.Elapsed)
    Else
        sw.Stop()
        Console.WriteLine("{0} is not a valid string. ({1:ss\.fffff} seconds)",
                           inputValue, sw.Elapsed)
    End If
Catch e As RegexMatchTimeoutException
    sw.Stop()
    ' Display the elapsed time until the exception.
    Console.WriteLine("Timeout with '{0}' after {1:ss\.fffff} seconds",
                      inputValue, sw.Elapsed)
    Thread.Sleep(1500)      ' Pause for 1.5 seconds.

    ' Increase the timeout interval and retry.
    Dim timeout As TimeSpan = e.MatchTimeout.Add(TimeSpan.FromSeconds(1))
    If timeout.TotalSeconds > MaxTimeoutInSeconds Then
        Console.WriteLine("Maximum timeout interval of {0} seconds exceeded.",
                           MaxTimeoutInSeconds)
        timedOut = False
    Else
        Console.WriteLine("Changing the timeout interval to {0}",
                          timeout)
        rgx = New Regex(pattern, RegexOptions.IgnoreCase, timeout)
        timedOut = True
    End If
End Try
Loop While timedOut
Console.WriteLine()

Next
End Sub
End Module
' The example displays output like the following:
' Processing aa
' Valid: 'aa' (00.0000779 seconds)
'
' Processing aaaa>
' 'aaaaa>' is not a valid string. (00.00005 seconds)
'
' Processingaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
' Valid: 'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa' (00.0000043 seconds)
'
' Processingaaaaaaaaaaaaaaaaaaaa>
' Timeout with 'aaaaaaaaaaaaaaaaaaaa>' after 01.00469
' Changing the timeout interval to 00:00:02
' Timeout with 'aaaaaaaaaaaaaaaaaaaa>' after 02.01202
' Changing the timeout interval to 00:00:03
' Timeout with 'aaaaaaaaaaaaaaaaaaaa>' after 03.01043
' Maximum timeout interval of 3 seconds exceeded.
'
' Processingaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa>
' Timeout with 'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa>' after 03.01018
' Maximum timeout interval of 3 seconds exceeded.

```

Atomic groups

The `(?> subexpression)` language element suppresses backtracking into the subexpression. Once it has successfully matched, it will not give up any part of its match to subsequent backtracking. For example, in the pattern `(?>\w*\d*)1`, if the `1` cannot be matched, the `\d*` will not give up any of its match even if that means it would allow the `1` to successfully match. Atomic groups can help prevent the performance problems associated with failed matches.

The following example illustrates how suppressing backtracking improves performance when using nested quantifiers. It measures the time required for the regular expression engine to determine that an input string does not match two regular expressions. The first regular expression uses backtracking to attempt to match a

string that contains one or more occurrences of one or more hexadecimal digits, followed by a colon, followed by one or more hexadecimal digits, followed by two colons. The second regular expression is identical to the first, except that it disables backtracking. As the output from the example shows, the performance improvement from disabling backtracking is significant.

```
using System;
using System.Diagnostics;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string input = "b51:4:1DB:9EE1:5:27d60:f44:D4:cd:E:5:0A5:4a:D24:41Ad:";
        bool matched;
        Stopwatch sw;

        Console.WriteLine("With backtracking:");
        string backPattern = "^(([0-9a-fA-F]{1,4}:)*([0-9a-fA-F]{1,4}))*(::)$";
        sw = Stopwatch.StartNew();
        matched = Regex.IsMatch(input, backPattern);
        sw.Stop();
        Console.WriteLine("Match: {0} in {1}", Regex.IsMatch(input, backPattern), sw.Elapsed);
        Console.WriteLine();

        Console.WriteLine("Without backtracking:");
        string noBackPattern = "^((?>[0-9a-fA-F]{1,4}:)*(?>[0-9a-fA-F]{1,4}))*(::)$";
        sw = Stopwatch.StartNew();
        matched = Regex.IsMatch(input, noBackPattern);
        sw.Stop();
        Console.WriteLine("Match: {0} in {1}", Regex.IsMatch(input, noBackPattern), sw.Elapsed);
    }
}

// The example displays output like the following:
//      With backtracking:
//      Match: False in 00:00:27.4282019
//
//      Without backtracking:
//      Match: False in 00:00:00.0001391
```

```

Imports System.Diagnostics
Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim input As String = "b51:4:1DB:9EE1:5:27d60:f44:D4:cd:E:5:0A5:4a:D24:41Ad:"
        Dim matched As Boolean
        Dim sw As Stopwatch

        Console.WriteLine("With backtracking:")
        Dim backPattern As String = "^(([0-9a-fA-F]{1,4}:)*([0-9a-fA-F]{1,4}))*(::)$"
        sw = Stopwatch.StartNew()
        matched = Regex.IsMatch(input, backPattern)
        sw.Stop()
        Console.WriteLine("Match: {0} in {1}", Regex.IsMatch(input, backPattern), sw.Elapsed)
        Console.WriteLine()

        Console.WriteLine("Without backtracking:")
        Dim noBackPattern As String = "^((?>[0-9a-fA-F]{1,4}:)*(?>[0-9a-fA-F]{1,4}))*(::)$"
        sw = Stopwatch.StartNew()
        matched = Regex.IsMatch(input, noBackPattern)
        sw.Stop()
        Console.WriteLine("Match: {0} in {1}", Regex.IsMatch(input, noBackPattern), sw.Elapsed)
    End Sub
End Module
' The example displays the following output:
'   With backtracking:
'     Match: False in 00:00:27.4282019
'
'   Without backtracking:
'     Match: False in 00:00:00.0001391

```

Lookbehind Assertions

.NET includes two language elements, `(?<= subexpression)` and `(?<! subexpression)`, that match the previous character or characters in the input string. Both language elements are zero-width assertions; that is, they determine whether the character or characters that immediately precede the current character can be matched by *subexpression*, without advancing or backtracking.

`(?<= subexpression)` is a positive lookbehind assertion; that is, the character or characters before the current position must match *subexpression*. `(?<! subexpression)` is a negative lookbehind assertion; that is, the character or characters before the current position must not match *subexpression*. Both positive and negative lookbehind assertions are most useful when *subexpression* is a subset of the previous subexpression.

The following example uses two equivalent regular expression patterns that validate the user name in an email address. The first pattern is subject to poor performance because of excessive backtracking. The second pattern modifies the first regular expression by replacing a nested quantifier with a positive lookbehind assertion. The output from the example displays the execution time of the `Regex.IsMatch` method.

```

using System;
using System.Diagnostics;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        Stopwatch sw;
        string input = "test@contoso.com";
        bool result;

        string pattern = @"^[\w-]+@[a-zA-Z]+\.\w+";
        sw = Stopwatch.StartNew();
        result = Regex.IsMatch(input, pattern, RegexOptions.IgnoreCase);
        sw.Stop();
        Console.WriteLine("Match: {0} in {1}", result, sw.Elapsed);

        string behindPattern = @"^[\w-]+@[a-zA-Z]+\.\w+(?<=[\w-])";
        sw = Stopwatch.StartNew();
        result = Regex.IsMatch(input, behindPattern, RegexOptions.IgnoreCase);
        sw.Stop();
        Console.WriteLine("Match with Lookbehind: {0} in {1}", result, sw.Elapsed);
    }
}

// The example displays output similar to the following:
//      Match: True in 00:00:00.0017549
//      Match with Lookbehind: True in 00:00:00.0000659

```

```

Module Example
    Public Sub Main()
        Dim sw As Stopwatch
        Dim input As String = "test@contoso.com"
        Dim result As Boolean

        Dim pattern As String = "^\w+@[a-zA-Z]+\.\w+"
        sw = Stopwatch.StartNew()
        result = Regex.IsMatch(input, pattern, RegexOptions.IgnoreCase)
        sw.Stop()
        Console.WriteLine("Match: {0} in {1}", result, sw.Elapsed)

        Dim behindPattern As String = "^\w+@[a-zA-Z]+\.\w+(?<=[\w-])"
        sw = Stopwatch.StartNew()
        result = Regex.IsMatch(input, behindPattern, RegexOptions.IgnoreCase)
        sw.Stop()
        Console.WriteLine("Match with Lookbehind: {0} in {1}", result, sw.Elapsed)
    End Sub
End Module
' The example displays output similar to the following:
'      Match: True in 00:00:00.0017549
'      Match with Lookbehind: True in 00:00:00.0000659

```

The first regular expression pattern, `^[\w-]+@[a-zA-Z]+\.\w+`, is defined as shown in the following table.

PATTERN	DESCRIPTION
<code>^</code>	Start the match at the beginning of the string.
<code>[\w-]+</code>	Match an alphanumeric character. This comparison is case-insensitive, because the <code>Regex.IsMatch</code> method is called with the <code>RegexOptions.IgnoreCase</code> option.

PATTERN	DESCRIPTION
<code>[-.\w]*</code>	Match zero, one, or more occurrences of a hyphen, period, or word character.
<code>[0-9A-Z]</code>	Match an alphanumeric character.
<code>([-.\w]*[0-9A-Z])*</code>	Match zero or more occurrences of the combination of zero or more hyphens, periods, or word characters, followed by an alphanumeric character. This is the first capturing group.
<code>@</code>	Match an at sign ("@").

The second regular expression pattern, `^[0-9A-Z][-.\w]*(?<=[0-9A-Z])@`, uses a positive lookbehind assertion. It is defined as shown in the following table.

PATTERN	DESCRIPTION
<code>^</code>	Start the match at the beginning of the string.
<code>[0-9A-Z]</code>	Match an alphanumeric character. This comparison is case-insensitive, because the Regex.IsMatch method is called with the RegexOptions.IgnoreCase option.
<code>[-.\w]*</code>	Match zero or more occurrences of a hyphen, period, or word character.
<code>(?<=[0-9A-Z])</code>	Look back at the last matched character and continue the match if it is alphanumeric. Note that alphanumeric characters are a subset of the set that consists of periods, hyphens, and all word characters.
<code>@</code>	Match an at sign ("@").

Lookahead Assertions

.NET includes two language elements, `(?= subexpression)` and `(?! subexpression)`, that match the next character or characters in the input string. Both language elements are zero-width assertions; that is, they determine whether the character or characters that immediately follow the current character can be matched by *subexpression*, without advancing or backtracking.

`(?= subexpression)` is a positive lookahead assertion; that is, the character or characters after the current position must match *subexpression*. `(?! subexpression)` is a negative lookahead assertion; that is, the character or characters after the current position must not match *subexpression*. Both positive and negative lookahead assertions are most useful when *subexpression* is a subset of the next subexpression.

The following example uses two equivalent regular expression patterns that validate a fully qualified type name. The first pattern is subject to poor performance because of excessive backtracking. The second modifies the first regular expression by replacing a nested quantifier with a positive lookahead assertion. The output from the example displays the execution time of the [Regex.IsMatch](#) method.

```

using System;
using System.Diagnostics;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string input = "aaaaaaaaaaaaaaaaaaaaaa.";
        bool result;
        Stopwatch sw;

        string pattern = @"^(([A-Z]\w*)+\.)*[A-Z]\w*$/";
        sw = Stopwatch.StartNew();
        result = Regex.IsMatch(input, pattern, RegexOptions.IgnoreCase);
        sw.Stop();
        Console.WriteLine("{0} in {1}", result, sw.Elapsed);

        string aheadPattern = @"^((?=([A-Z])\w+\.)*[A-Z]\w*$/";
        sw = Stopwatch.StartNew();
        result = Regex.IsMatch(input, aheadPattern, RegexOptions.IgnoreCase);
        sw.Stop();
        Console.WriteLine("{0} in {1}", result, sw.Elapsed);
    }
}

// The example displays the following output:
//      False in 00:00:03.8003793
//      False in 00:00:00.0000866

```

```

Imports System.Diagnostics
Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim input As String = "aaaaaaaaaaaaaaaaaaaaaa."
        Dim result As Boolean
        Dim sw As Stopwatch

        Dim pattern As String = "^(([A-Z]\w*)+\.)*[A-Z]\w*$/"
        sw = Stopwatch.StartNew()
        result = Regex.IsMatch(input, pattern, RegexOptions.IgnoreCase)
        sw.Stop()
        Console.WriteLine("{0} in {1}", result, sw.Elapsed)

        Dim aheadPattern As String = "^((?=([A-Z])\w+\.)*[A-Z]\w*$/"
        sw = Stopwatch.StartNew()
        result = Regex.IsMatch(input, aheadPattern, RegexOptions.IgnoreCase)
        sw.Stop()
        Console.WriteLine("{0} in {1}", result, sw.Elapsed)
    End Sub
End Module

' The example displays the following output:
'      False in 00:00:03.8003793
'      False in 00:00:00.0000866

```

The first regular expression pattern, `^(([A-Z]\w*)+\.)*[A-Z]\w*$/`, is defined as shown in the following table.

PATTERN	DESCRIPTION
<code>^</code>	Start the match at the beginning of the string.

PATTERN	DESCRIPTION
([A-Z]\w*)+\.	Match an alphabetical character (A-Z) followed by zero or more word characters one or more times, followed by a period. This comparison is case-insensitive, because the Regex.IsMatch method is called with the RegexOptions.IgnoreCase option.
(([A-Z]\w*)+\.)*	Match the previous pattern zero or more times.
[A-Z]\w*	Match an alphabetical character followed by zero or more word characters.
\$	End the match at the end of the input string.

The second regular expression pattern, `^((?=([A-Z])\w+\.)*[A-Z]\w*\$)`, uses a positive lookahead assertion. It is defined as shown in the following table.

PATTERN	DESCRIPTION
^	Start the match at the beginning of the string.
(?=([A-Z])\w+\.)*	Look ahead to the first character and continue the match if it is alphabetical (A-Z). This comparison is case-insensitive, because the Regex.IsMatch method is called with the RegexOptions.IgnoreCase option.
\w+\.	Match one or more word characters followed by a period.
((?=([A-Z])\w+\.)*\w+\.)*	Match the pattern of one or more word characters followed by a period zero or more times. The initial word character must be alphabetical.
[A-Z]\w*	Match an alphabetical character followed by zero or more word characters.
\$	End the match at the end of the input string.

See also

- [.NET Regular Expressions](#)
- [Regular Expression Language - Quick Reference](#)
- [Quantifiers](#)
- [Alternation Constructs](#)
- [Grouping Constructs](#)

Compilation and Reuse in Regular Expressions

9/20/2022 • 2 minutes to read • [Edit Online](#)

You can optimize the performance of applications that make extensive use of regular expressions by understanding how the regular expression engine compiles expressions and by understanding how regular expressions are cached. This topic discusses both compilation and caching.

Compiled Regular Expressions

By default, the regular expression engine compiles a regular expression to a sequence of internal instructions (these are high-level codes that are different from Microsoft intermediate language, or MSIL). When the engine executes a regular expression, it interprets the internal codes.

If a [Regex](#) object is constructed with the [RegexOptions.Compiled](#) option, it compiles the regular expression to explicit MSIL code instead of high-level regular expression internal instructions. This allows .NET's just-in-time (JIT) compiler to convert the expression to native machine code for higher performance. The cost of constructing the [Regex](#) object may be higher, but the cost of performing matches with it is likely to be much smaller.

An alternative is to use precompiled regular expressions. You can compile all of your expressions into a reusable DLL by using the [CompileToAssembly](#) method. This avoids the need to compile at run time while still benefiting from the speed of compiled regular expressions.

The Regular Expressions Cache

To improve performance, the regular expression engine maintains an application-wide cache of compiled regular expressions. The cache stores regular expression patterns that are used only in static method calls. (Regular expression patterns supplied to instance methods are not cached.) This avoids the need to reparse an expression into high-level byte code each time it is used.

The maximum number of cached regular expressions is determined by the value of the `static` (`Shared` in Visual Basic) [Regex.CacheSize](#) property. By default, the regular expression engine caches up to 15 compiled regular expressions. If the number of compiled regular expressions exceeds the cache size, the least recently used regular expression is discarded and the new regular expression is cached.

Your application can reuse regular expressions in one of the following two ways:

- By using a static method of the [Regex](#) object to define the regular expression. If you're using a regular expression pattern that has already been defined by another static method call, the regular expression engine will try to retrieve it from the cache. If it's not available in the cache, the engine will compile the regular expression and add it to the cache.
- By reusing an existing [Regex](#) object as long as its regular expression pattern is needed.

Because of the overhead of object instantiation and regular expression compilation, creating and rapidly destroying numerous [Regex](#) objects is a very expensive process. For applications that use a large number of different regular expressions, you can optimize performance by using calls to static [Regex](#) methods and possibly by increasing the size of the regular expression cache.

See also

- [.NET Regular Expressions](#)

Thread Safety in Regular Expressions

9/20/2022 • 2 minutes to read • [Edit Online](#)

The [Regex](#) class itself is thread safe and immutable (read-only). That is, **Regex** objects can be created on any thread and shared between threads; matching methods can be called from any thread and never alter any global state.

However, result objects (**Match** and **MatchCollection**) returned by **Regex** should be used on a single thread. Although many of these objects are logically immutable, their implementations could delay computation of some results to improve performance, and as a result, callers must serialize access to them.

If there is a need to share **Regex** result objects on multiple threads, these objects can be converted to thread-safe instances by calling their synchronized methods. With the exception of enumerators, all regular expression classes are thread safe or can be converted into thread-safe objects by a synchronized method.

Enumerators are the only exception. An application must serialize calls to collection enumerators. The rule is that if a collection can be enumerated on more than one thread simultaneously, you should synchronize enumerator methods on the root object of the collection traversed by the enumerator.

See also

- [.NET Regular Expressions](#)

Regular expression example: Scanning for HREFs

9/20/2022 • 3 minutes to read • [Edit Online](#)

The following example searches an input string and displays all the href="..." values and their locations in the string.

WARNING

When using `System.Text.RegularExpressions` to process untrusted input, pass a timeout. A malicious user can provide input to `RegularExpressions` causing a Denial-of-Service attack. ASP.NET Core framework APIs that use `RegularExpressions` pass a timeout.

The Regex object

Because the `DumpHRefs` method can be called multiple times from user code, it uses the `static` (`Shared` in Visual Basic) `Regex.Match(String, String, RegexOptions)` method. This enables the regular expression engine to cache the regular expression and avoids the overhead of instantiating a new `Regex` object each time the method is called. A `Match` object is then used to iterate through all matches in the string.

```
private static void DumpHRefs(string inputString)
{
    string hrefPattern = @"href\s*=\s*(?:['"](?<1>[^'"]*)['"]|(?<1>[^>\s]+))";

    try
    {
        Match regexMatch = Regex.Match(inputString, hrefPattern,
            RegexOptions.IgnoreCase | RegexOptions.Compiled,
            TimeSpan.FromSeconds(1));
        while (regexMatch.Success)
        {
            Console.WriteLine($"Found href {regexMatch.Groups[1]} at {regexMatch.Groups[1].Index}");
            regexMatch = regexMatch.NextMatch();
        }
    }
    catch (RegexMatchTimeoutException)
    {
        Console.WriteLine("The matching operation timed out.");
    }
}
```

```

Private Sub DumpHRefs(inputString As String)
    Dim hrefPattern As String = "href\s*=\s*(?:[''](?'>[^''']*)['']|(?<1>[^>]\s+))"

    Try
        Dim regexMatch = Regex.Match(inputString, hrefPattern,
            RegexOptions.IgnoreCase Or RegexOptions.Compiled,
            TimeSpan.FromSeconds(1))
        Do While regexMatch.Success
            Console.WriteLine($"Found href {regexMatch.Groups(1)} at {regexMatch.Groups(1).Index}.")
            regexMatch = regexMatch.NextMatch()
        Loop
    Catch e As RegexMatchTimeoutException
        Console.WriteLine("The matching operation timed out.")
    End Try
End Sub

```

The following example then illustrates a call to the `DumpHRefs` method.

```

public static void Main()
{
    string inputString = "My favorite web sites include:</P>" +
        "<A HREF=\"https://docs.microsoft.com/en-us/dotnet/\">>" +
        ".NET Documentation</A></P>" +
        "<A HREF=\"http://www.microsoft.com\">>" +
        "Microsoft Corporation Home Page</A></P>" +
        "<A HREF=\"https://devblogs.microsoft.com/dotnet/\">>" +
        ".NET Blog</A></P>";
    DumpHRefs(inputString);
}
// The example displays the following output:
//     Found href https://docs.microsoft.com/dotnet/ at 43
//     Found href http://www.microsoft.com at 114
//     Found href https://devblogs.microsoft.com/dotnet/ at 188

```

```

Public Sub Main()
    Dim inputString As String = "My favorite web sites include:</P>" &
        "<A HREF=""https://docs.microsoft.com/en-us/dotnet/"">" &
        ".NET Documentation</A></P>" &
        "<A HREF=""http://www.microsoft.com"">" &
        "Microsoft Corporation Home Page</A></P>" &
        "<A HREF=""https://devblogs.microsoft.com/dotnet/"">" &
        ".NET Blog</A></P>"

    DumpHRefs(inputString)
End Sub
' The example displays the following output:
'     Found href https://docs.microsoft.com/dotnet/ at 43
'     Found href http://www.microsoft.com at 114
'     Found href https://devblogs.microsoft.com/dotnet/ at 188

```

The regular expression pattern `href\s*=\s*(?:[''](?'>[^''']*)['']|(?<1>[^>]\s+))` is interpreted as shown in the following table.

PATTERN	DESCRIPTION
<code>href</code>	Match the literal string "href". The match is case-insensitive.
<code>\s*</code>	Match zero or more white-space characters.
<code>=</code>	Match the equals sign.

PATTERN	DESCRIPTION
\s*	Match zero or more white-space characters.
(?:	Start a non-capturing group.
["'](?<1>[^"']*)["']	Match a quotation mark or apostrophe, followed by a capturing group that matches any character other than a quotation mark or apostrophe, followed by a quotation mark or apostrophe. The group named <code>1</code> is included in this pattern.
	Boolean OR that matches either the previous expression or the next expression.
(?<1>[^>\s]+)	A capturing group that uses a negated set to match any character other than a greater-than sign or a whitespace character. The group named <code>1</code> is included in this pattern.
)	End the non-capturing group.

Match result class

The results of a search are stored in the [Match](#) class, which provides access to all the substrings extracted by the search. It also remembers the string being searched and the regular expression being used, so it can call the [Match.NextMatch](#) method to perform another search starting where the last one ended.

Explicitly named captures

In traditional regular expressions, capturing parentheses are automatically numbered sequentially. This leads to two problems. First, if a regular expression is modified by inserting or removing a set of parentheses, all code that refers to the numbered captures must be rewritten to reflect the new numbering. Second, because different sets of parentheses often are used to provide two alternative expressions for an acceptable match, it might be difficult to determine which of the two expressions actually returned a result.

To address these problems, the [Regex](#) class supports the syntax `(?<name>...)` for capturing a match into a specified slot (the slot can be named using a string or an integer; integers can be recalled more quickly). Thus, alternative matches for the same string all can be directed to the same place. In case of a conflict, the last match dropped into a slot is the successful match. (However, a complete list of multiple matches for a single slot is available. See the [Group.Captures](#) collection for details.)

See also

- [.NET Regular Expressions](#)

Regular Expression Example: Changing Date Formats

9/20/2022 • 2 minutes to read • [Edit Online](#)

The following code example uses the `Regex.Replace` method to replace dates that have the form `mm/dd/yy` with dates that have the form `dd-mm-yy`.

WARNING

When using `System.Text.RegularExpressions` to process untrusted input, pass a timeout. A malicious user can provide input to `RegularExpressions` causing a Denial-of-Service attack. ASP.NET Core framework APIs that use `RegularExpressions` pass a timeout.

Example

```
static string MDYToDMY(string input)
{
    try {
        return Regex.Replace(input,
            @"\b(?:month)\d{1,2})/(?:day)\d{1,2})/(?:year)\d{2,4})\b",
            "${day}-${month}-${year}", RegexOptions.None,
            TimeSpan.FromMilliseconds(150));
    }
    catch (RegexMatchTimeoutException) {
        return input;
    }
}
```

```
Function MDYToDMY(input As String) As String
    Try
        Return Regex.Replace(input, _
            @"\b(?:month)\d{1,2})/(?:day)\d{1,2})/(?:year)\d{2,4})\b", _
            "${day}-${month}-${year}", RegexOptions.None,
            TimeSpan.FromMilliseconds(150))
    Catch e As RegexMatchTimeoutException
        Return input
    End Try
End Function
```

The following code shows how the `MDYToDMY` method can be called in an application.

```

using System;
using System.Globalization;
using System.Text.RegularExpressions;

public class Class1
{
    public static void Main()
    {
        string dateString = DateTime.Today.ToString("d",
            DateTimeFormatInfo.InvariantInfo);
        string resultString = MDYToDMY(dateString);
        Console.WriteLine("Converted {0} to {1}.", dateString, resultString);
    }

    static string MDYToDMY(string input)
    {
        try {
            return Regex.Replace(input,
                @"\b(?:month)\d{1,2})/(?:day)\d{1,2})/(?:year)\d{2,4})\b",
                "${day}-${month}-${year}", RegexOptions.None,
                TimeSpan.FromMilliseconds(150));
        }
        catch (RegexMatchTimeoutException) {
            return input;
        }
    }
}

// The example displays the following output to the console if run on 8/21/2007:
//      Converted 08/21/2007 to 21-08-2007.

```

```

Imports System.Globalization
Imports System.Text.RegularExpressions

Module DateFormatReplacement
    Public Sub Main()
        Dim dateString As String = Date.Today.ToString("d", _
            DateTimeFormatInfo.InvariantInfo)
        Dim resultString As String = MDYToDMY(dateString)
        Console.WriteLine("Converted {0} to {1}.", dateString, resultString)
    End Sub

    Function MDYToDMY(input As String) As String
        Try
            Return Regex.Replace(input, _
                @"\b(?:month)\d{1,2})/(?:day)\d{1,2})/(?:year)\d{2,4})\b",
                "${day}-${month}-${year}", RegexOptions.None,
                TimeSpan.FromMilliseconds(150))
        Catch e As RegexMatchTimeoutException
            Return input
        End Try
    End Function
End Module

' The example displays the following output to the console if run on 8/21/2007:
'      Converted 08/21/2007 to 21-08-2007.

```

Comments

The regular expression pattern `\b(?:month)\d{1,2})/(?:day)\d{1,2})/(?:year)\d{2,4})\b` is interpreted as shown in the following table.

PATTERN	DESCRIPTION
\b	Begin the match at a word boundary.
(?<month>\d{1,2})	Match one or two decimal digits. This is the <code>month</code> captured group.
/	Match the slash mark.
(?<day>\d{1,2})	Match one or two decimal digits. This is the <code>day</code> captured group.
/	Match the slash mark.
(?<year>\d{2,4})	Match from two to four decimal digits. This is the <code>year</code> captured group.
\b	End the match at a word boundary.

The pattern `${day}-${month}-${year}` defines the replacement string as shown in the following table.

PATTERN	DESCRIPTION
<code>\$(day)</code>	Add the string captured by the <code>day</code> capturing group.
<code>-</code>	Add a hyphen.
<code>\$(month)</code>	Add the string captured by the <code>month</code> capturing group.
<code>-</code>	Add a hyphen.
<code>\$(year)</code>	Add the string captured by the <code>year</code> capturing group.

See also

- [.NET Regular Expressions](#)

How to: Extract a Protocol and Port Number from a URL

9/20/2022 • 2 minutes to read • [Edit Online](#)

The following example extracts a protocol and port number from a URL.

WARNING

When using `System.Text.RegularExpressions` to process untrusted input, pass a timeout. A malicious user can provide input to `RegularExpressions` causing a Denial-of-Service attack. ASP.NET Core framework APIs that use `RegularExpressions` pass a timeout.

Example

The example uses the `Match.Result` method to return the protocol followed by a colon followed by the port number.

```
using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string url = "http://www.contoso.com:8080/letters/readme.html";

        Regex r = new Regex(@"^(?<proto>\w+://[^/]+?(?<port>:\d+)?/",
                           RegexOptions.None, TimeSpan.FromMilliseconds(150));
        Match m = r.Match(url);
        if (m.Success)
            Console.WriteLine(m.Result("${proto}${port}"));
    }
}
// The example displays the following output:
//      http:8080
```

```
Imports System.Text.RegularExpressions

Module Example
    Public Sub Main()
        Dim url As String = "http://www.contoso.com:8080/letters/readme.html"
        Dim r As New Regex("^(?<proto>\w+://[^/]+?(?<port>:\d+)?/",
                           RegexOptions.None, TimeSpan.FromMilliseconds(150))

        Dim m As Match = r.Match(url)
        If m.Success Then
            Console.WriteLine(m.Result("${proto}${port}"))
        End If
    End Sub
End Module
' The example displays the following output:
'      http:8080
```

The regular expression pattern `^(?<proto>\w+://[^/]+?(?<port>:\d+)?/` can be interpreted as shown in the

following table.

PATTERN	DESCRIPTION
<code>^</code>	Begin the match at the start of the string.
<code>(?<proto>\w+)</code>	Match one or more word characters. Name this group <code>proto</code> .
<code>:://</code>	Match a colon followed by two slash marks.
<code>[^/]+?</code>	Match one or more occurrences (but as few as possible) of any character other than a slash mark.
<code>(?<port>:\d+)?</code>	Match zero or one occurrence of a colon followed by one or more digit characters. Name this group <code>port</code> .
<code>/</code>	Match a slash mark.

The [Match.Result](#) method expands the `${proto}${port}` replacement sequence, which concatenates the value of the two named groups captured in the regular expression pattern. It is a convenient alternative to explicitly concatenating the strings retrieved from the collection object returned by the [Match.Groups](#) property.

The example uses the [Match.Result](#) method with two substitutions, `${proto}` and `${port}`, to include the captured groups in the output string. You can retrieve the captured groups from the match's [GroupCollection](#) object instead, as the following code shows.

```
Console.WriteLine(m.Groups["proto"].Value + m.Groups["port"].Value);
```

```
Console.WriteLine(m.Groups("proto").Value + m.Groups("port").Value)
```

See also

- [.NET Regular Expressions](#)

How to: Strip Invalid Characters from a String

9/20/2022 • 2 minutes to read • [Edit Online](#)

The following example uses the static `Regex.Replace` method to strip invalid characters from a string.

WARNING

When using `System.Text.RegularExpressions` to process untrusted input, pass a timeout. A malicious user can provide input to `RegularExpressions` causing a Denial-of-Service attack. ASP.NET Core framework APIs that use `RegularExpressions` pass a timeout.

Example

You can use the `CleanInput` method defined in this example to strip potentially harmful characters that have been entered into a text field that accepts user input. In this case, `CleanInput` strips out all nonalphanumeric characters except periods (.), at symbols (@), and hyphens (-), and returns the remaining string. However, you can modify the regular expression pattern so that it strips out any characters that should not be included in an input string.

```
using System;
using System.Text.RegularExpressions;

public class Example
{
    static string CleanInput(string strIn)
    {
        // Replace invalid characters with empty strings.
        try {
            return Regex.Replace(strIn, @"[^w\.\@-]", "", RegexOptions.None, TimeSpan.FromSeconds(1.5));
        }
        // If we timeout when replacing invalid characters,
        // we should return Empty.
        catch (RegexMatchTimeoutException) {
            return String.Empty;
        }
    }
}
```

```
Imports System.Text.RegularExpressions

Module Example
    Function CleanInput(strIn As String) As String
        ' Replace invalid characters with empty strings.
        Try
            Return Regex.Replace(strIn, @"[^w\.\@-]", "")
        ' If we timeout when replacing invalid characters,
        ' we should return String.Empty.
        Catch e As RegexMatchTimeoutException
            Return String.Empty
        End Try
    End Function
End Module
```

The regular expression pattern `[^\w\.\@-]` matches any character that is not a word character, a period, an @ symbol, or a hyphen. A word character is any letter, decimal digit, or punctuation connector such as an underscore. Any character that matches this pattern is replaced by `String.Empty`, which is the string defined by the replacement pattern. To allow additional characters in user input, add those characters to the character class in the regular expression pattern. For example, the regular expression pattern `[^\w\.\@-\%\]` also allows a percentage symbol and a backslash in an input string.

See also

- [.NET Regular Expressions](#)

How to verify that strings are in valid email format

9/20/2022 • 4 minutes to read • [Edit Online](#)

The following example uses a regular expression to verify that a string is in valid email format.

This regular expression is comparatively simple to what can actually be used as an email. Using a regular expression to validate an email is useful to make sure the structure of an email is correct, but it isn't a substitution for verifying the email actually exists.

- ✓ DO use a small regular expression to check for the valid structure of an email.
- ✓ DO send a test email to the address provided by a user of your app.
- ✗ DON'T use a regular expression as the only way you validate an email.

If you try to create the *perfect* regular expression to validate that the structure of an email is correct, the expression becomes so complex that it's incredibly difficult to debug or improve. Regular expressions can't validate an email exists, even if it's structured correctly. The best way to validate an email is to send a test email to the address.

WARNING

When using `System.Text.RegularExpressions` to process untrusted input, pass a timeout. A malicious user can provide input to `RegularExpressions` causing a Denial-of-Service attack. ASP.NET Core framework APIs that use `RegularExpressions` pass a timeout.

Example

The example defines an `IsValidEmail` method, which returns `true` if the string contains a valid email address and `false` if it doesn't, but takes no other action.

To verify that the email address is valid, the `IsValidEmail` method calls the `Regex.Replace(String, String, MatchEvaluator)` method with the `(@)(.+)$` regular expression pattern to separate the domain name from the email address. The third parameter is a `MatchEvaluator` delegate that represents the method that processes and replaces the matched text. The regular expression pattern is interpreted as follows.

PATTERN	DESCRIPTION
<code>(@)</code>	Match the @ character. This part is the first capturing group.
<code>(.+)</code>	Match one or more occurrences of any character. This part is the second capturing group.
<code>\$</code>	End the match at the end of the string.

The domain name along with the @ character is passed to the `DomainMapper` method, which uses the `IIdnMapping` class to translate Unicode characters that are outside the US-ASCII character range to Punycode. The method also sets the `invalid` flag to `True` if the `IIdnMapping.GetAscii` method detects any invalid characters in the domain name. The method returns the Punycode domain name preceded by the @ symbol to the `IsValidEmail` method.

TIP

It's recommended that you use the simple `(@)(.+)$` regular expression pattern to normalize the domain and then return a value indicating that it passed or failed. However, the example in this article describes how to further use a regular expression to validate the email. Regardless of how you validate an email, you should always send a test email to the address to make sure it exists.

The `IsValidEmail` method then calls the `Regex.IsMatch(String, String)` method to verify that the address conforms to a regular expression pattern.

The `IsValidEmail` method merely determines whether the email format is valid for an email address, it doesn't validate that the email exists. Also, the `IsValidEmail` method doesn't verify that the top-level domain name is a valid domain name listed at the [IANA Root Zone Database](#), which would require a look-up operation.

```

using System;
using System.Globalization;
using System.Text.RegularExpressions;

namespace RegexExamples
{
    class RegexUtilities
    {
        public static bool IsValidEmail(string email)
        {
            if (string.IsNullOrWhiteSpace(email))
                return false;

            try
            {
                // Normalize the domain
                email = Regex.Replace(email, @"(@)(.+)$", DomainMapper,
                    RegexOptions.None, TimeSpan.FromMilliseconds(200));

                // Examines the domain part of the email and normalizes it.
                string DomainMapper(Match match)
                {
                    // Use IdnMapping class to convert Unicode domain names.
                    var idn = new IdnMapping();

                    // Pull out and process domain name (throws ArgumentException on invalid)
                    string domainName = idn.GetAscii(match.Groups[2].Value);

                    return match.Groups[1].Value + domainName;
                }
            }
            catch (RegexMatchTimeoutException e)
            {
                return false;
            }
            catch (ArgumentException e)
            {
                return false;
            }

            try
            {
                return Regex.IsMatch(email,
                    @"^[@\s]+@[^\s]+\.[^\s]+$",
                    RegexOptions.IgnoreCase, TimeSpan.FromMilliseconds(250));
            }
            catch (RegexMatchTimeoutException)
            {
                return false;
            }
        }
    }
}

```

```

Imports System.Globalization
Imports System.Text.RegularExpressions

Public Class RegexUtilities
    Public Shared Function IsValidEmail(email As String) As Boolean

        If String.IsNullOrWhiteSpace(email) Then Return False

        ' Use IdnMapping class to convert Unicode domain names.
        Try
            'Examines the domain part of the email and normalizes it.
            Dim DomainMapper =
                Function(match As Match) As String

                    'Use IdnMapping class to convert Unicode domain names.
                    Dim idn = New IdnMapping

                    'Pull out and process domain name (throws ArgumentException on invalid)
                    Dim domainName As String = idn.GetAscii(match.Groups(2).Value)

                    Return match.Groups(1).Value & domainName

                End Function

            'Normalize the domain
            email = Regex.Replace(email, "(@)(.+)$", DomainMapper,
                RegexOptions.None, TimeSpan.FromMilliseconds(200))

            Catch e As RegexMatchTimeoutException
                Return False

            Catch e As ArgumentException
                Return False

        End Try

        Try
            Return Regex.IsMatch(email,
                "^[^@\s]+@[^\s]+\.[^\s]+$",
                RegexOptions.IgnoreCase, TimeSpan.FromMilliseconds(250))

            Catch e As RegexMatchTimeoutException
                Return False

        End Try

    End Function
End Class

```

In this example, the regular expression pattern `^[^@\s]+@[^\s]+\.[^\s]+$` is interpreted as shown in the following table. The regular expression is compiled using the [RegexOptions.IgnoreCase](#) flag.

PATTERN	DESCRIPTION
<code>^</code>	Begin the match at the start of the string.
<code>[^@\s]+</code>	Match one or more occurrences of any character other than the @ character or whitespace.
<code>@</code>	Match the @ character.

PATTERN	DESCRIPTION
[^@\s]+	Match one or more occurrences of any character other than the @ character or whitespace.
\.	Match a single period character.
[^@\s]+	Match one or more occurrences of any character other than the @ character or whitespace.
\$	End the match at the end of the string.

IMPORTANT

This regular expression is not intended to cover every aspect of a valid email address. It's provided as an example for you to extend as needed.

See also

- [.NET Regular Expressions](#)
- [How far should one take e-mail address validation?](#)

Serialization in .NET

9/20/2022 • 2 minutes to read • [Edit Online](#)

Serialization is the process of converting the state of an object into a form that can be persisted or transported. The complement of serialization is deserialization, which converts a stream into an object. Together, these processes allow data to be stored and transferred.

.NET features the following serialization technologies:

- [Binary serialization](#) preserves type fidelity, which is useful for preserving the state of an object between different invocations of an application. For example, you can share an object between different applications by serializing it to the Clipboard. You can serialize an object to a stream, to a disk, to memory, over the network, and so forth. Remoting uses serialization to pass objects "by value" from one computer or application domain to another.
- [XML and SOAP serialization](#) serializes only public properties and fields and does not preserve type fidelity. This is useful when you want to provide or consume data without restricting the application that uses the data. Because XML is an open standard, it is an attractive choice for sharing data across the Web. SOAP is likewise an open standard, which makes it an attractive choice.
- [JSON serialization](#) serializes only public properties and does not preserve type fidelity. JSON is an open standard that is an attractive choice for sharing data across the web.

Reference

[System.Runtime.Serialization](#)

Contains classes that can be used for serializing and deserializing objects.

[System.Xml.Serialization](#)

Contains classes that can be used to serialize objects into XML format documents or streams.

[System.Text.Json](#)

Contains classes that can be used to serialize objects into JSON format documents or streams.

JSON serialization and deserialization (marshalling and unmarshalling) in .NET - overview

9/20/2022 • 2 minutes to read • [Edit Online](#)

The `System.Text.Json` namespace provides functionality for serializing to and deserializing from JavaScript Object Notation (JSON).

The library design emphasizes high performance and low memory allocation over an extensive feature set. Built-in UTF-8 support optimizes the process of reading and writing JSON text encoded as UTF-8, which is the most prevalent encoding for data on the web and files on disk.

The library also provides classes for working with an in-memory [document object model \(DOM\)](#). This feature enables random access to the elements in a JSON file or string.

There are some limitations on what parts of the library that you can use from Visual Basic code. For more information, see [Visual Basic support](#).

Run-time reflection vs. compile-time source generation

By default, `System.Text.Json` uses run-time [reflection](#) to gather the metadata it needs to access properties of objects for serialization and deserialization. As an alternative, `System.Text.Json` can use the C# [source generation](#) feature to improve performance, reduce private memory usage, and facilitate [assembly trimming](#), which reduces app size. For more information, see [How to choose reflection or source generation in System.Text.Json](#).

How to get the library

The library is built-in as part of the shared framework for .NET Core 3.0 and later versions. The source generation feature is built-in as part of the shared framework for .NET 6 and later versions. Use of source generation requires .NET 5 SDK or later.

For framework versions earlier than .NET Core 3.0, install the `System.Text.Json` NuGet package. The package supports:

- .NET Standard 2.0 and later
- .NET Framework 4.7.2 and later
- .NET Core 2.1 and later
- .NET 5 and later

Security information

For information about security threats that were considered when designing [JsonSerializer](#), and how they can be mitigated, see [System.Text.Json Threat Model](#).

Thread safety

- The `System.Text.Json` types are thread-safe, including:
 - `JsonSerializer`
 - `Utf8JsonReader`

- [Utf8JsonWriter](#)
- [JsonDocument](#)
- [JsonSerializer](#)
- [Utf8JsonReader](#)
- [Utf8JsonWriter](#)
- [JsonDocument](#)
- [JsonNode](#)

Additional resources

- [How to use the library](#)
- [Instantiate JsonSerializerOptions instances](#)
- [Enable case-insensitive matching](#)
- [Customize property names and values](#)
- [Ignore properties](#)
- [Allow invalid JSON](#)
- [Handle overflow JSON or use JsonElement or JsonNode](#)
- [Preserve references and handle circular references](#)
- [Deserialize to immutable types and non-public accessors](#)
- [Polymorphic serialization](#)
- [Migrate from Newtonsoft.Json to System.Text.Json](#)
- [Customize character encoding](#)
- [Use DOM, Utf8JsonReader, and Utf8JsonWriter](#)
- [Write custom converters for JSON serialization](#)
- [DateTime and DateTimeOffset support](#)
- [How to use source generation](#)
- [Supported collection types](#)
- [System.Text.Json API reference](#)
- [System.Text.Json.Serialization API reference](#)

How to choose reflection or source generation in System.Text.Json

9/20/2022 • 5 minutes to read • [Edit Online](#)

By default, `System.Text.Json` uses run-time reflection to gather the metadata it needs to access properties of objects for serialization and deserialization. As an alternative, `System.Text.Json` 6.0 can use the C# [source generation](#) feature to improve performance, reduce private memory usage, and facilitate [assembly trimming](#), which reduces app size.

You can use version 6.0 of `System.Text.Json` in projects that target earlier frameworks. For more information, see [How to get the library](#).

This article explains the options and provides guidance on how to choose the best approach for your scenario.

`System.Text.Json` version 6.0 and later can use the C# [source generation](#) feature to improve performance, reduce private memory usage, and improve [assembly trimming](#) accuracy. You can use version 6.0 of `System.Text.Json` in projects that target earlier frameworks. For more information, see:

- [How to get the library](#)
- [The .NET 6 version of this article](#).

Overview

Choose reflection or source generation modes based on the following benefits that each one offers:

BENEFIT	REFLECTION	SOURCE GENERATION: METADATA COLLECTION	SOURCE GENERATION: SERIALIZATION OPTIMIZATION
Simpler to code and debug.	✓	✗	✗
Supports non-public accessors.	✓	✗	✗
Supports init-only properties.	✓	✗	✗
Supports all available serialization customizations.	✓	✗	✗
Reduces start-up time.	✗	✓	✗
Reduces private memory usage.	✗	✓	✓
Eliminates run-time reflection.	✗	✓	✓
Facilitates trim-safe app size reduction.	✗	✓	✓

BENEFIT	REFLECTION	SOURCE GENERATION: METADATA COLLECTION	SOURCE GENERATION: SERIALIZATION OPTIMIZATION
Increases serialization throughput.	✗	✗	✓

The following sections explain these options and their relative benefits.

System.Text.Json metadata

To serialize or deserialize a type, [JsonSerializer](#) needs information about how to access the members of the type. [JsonSerializer](#) needs the following information:

- How to access property getters and fields for serialization.
- How to access a constructor, property setters, and fields for deserialization.
- Information about which attributes have been used to customize serialization or deserialization.
- Run-time configuration from [JsonSerializerOptions](#).

This information is referred to as *metadata*.

By default, [JsonSerializer](#) collects metadata at run time by using [reflection](#). Whenever [JsonSerializer](#) has to serialize or deserialize a type for the first time, it collects and caches this metadata. The metadata collection process takes time and uses memory.

Source generation - metadata collection mode

You can use source generation to move the metadata collection process from run time to compile time. During compilation, the metadata is collected and source code files are generated. The generated source code files are automatically compiled as an integral part of the application. This compile-time metadata collection eliminates run-time metadata collection, which improves performance of both serialization and deserialization.

The performance improvements provided by source generation can be substantial. For example, [test results](#) have shown up to 40% or more startup time reduction, private memory reduction, throughput speed increase (in serialization optimization mode), and app size reduction.

Source generation - known issues

Reflection mode supports the use of non-public accessors of public properties. For example, you can apply [\[JsonInclude\]](#) to a property that has a [private](#) setter or getter. Source generation mode supports only public or internal accessors of public properties. Use of [\[JsonInclude\]](#) on non-public accessors in source generation mode results in a [NotSupportedException](#) at run time.

Reflection mode also supports deserialization to init-only properties. Source generation doesn't support this, because the metadata-only mode required for deserialization can't express the required initialization statically in source code. The reflection serializer uses run-time reflection to set properties after construction.

In both reflection and source generation modes:

- Only public properties and public fields are supported.
- Only public constructors can be used for deserialization.

For information about the outstanding request to add support for non-public members, see GitHub issue [dotnet/runtime#31511](#). Even if that request is implemented, source generation mode will still be limited to support for public members.

For information about other known issues with source generation, see the [GitHub issues that are labeled "source-generator"](#) in the `dotnet/runtime` repository.

Serialization optimization mode

`JsonSerializer` has many features that customize the output of serialization, such as [camel-casing property names](#) and [preserving references](#). Support for all those features causes some performance overhead. Source generation can improve serialization performance by generating optimized code that uses `Utf8JsonWriter` directly.

The optimized code doesn't support all of the serialization features that `JsonSerializer` supports. The serializer detects whether the optimized code can be used and falls back to default serialization code if unsupported options are specified. For example, `JsonNumberHandling.AllowReadingFromString` is not applicable to writing, so specifying this option doesn't cause a fall-back to default code.

The following table shows which options in `JsonSerializerOptions` are supported by the optimized serialization code:

SERIALIZATION OPTION	SUPPORTED BY OPTIMIZED CODE
Converters	✗
DefaultIgnoreCondition	✓
DictionaryKeyPolicy	✗
Encoder	✗
IgnoreNullValues	✗
IgnoreReadOnlyFields	✓
IgnoreReadOnlyProperties	✓
IncludeFields	✓
NumberHandling	✗
PropertyNamingPolicy	✓
ReferenceHandler	✗
WriteIndented	✓

The following table shows which attributes are supported by the optimized serialization code:

ATTRIBUTE	SUPPORTED BY OPTIMIZED CODE
<code>JsonConverterAttribute</code>	✗
<code>JsonExtensionDataAttribute</code>	✗
<code>JsonIgnoreAttribute</code>	✓

ATTRIBUTE	SUPPORTED BY OPTIMIZED CODE
JsonIncludeAttribute	✓
JsonNumberHandlingAttribute	✗
JsonPropertyNameAttribute	✓

If a non-supported option or attribute is specified for a type, the serializer falls back to the default

`JsonSerializer` code. In that case, the optimized code isn't used when serializing that type but may be used for other types. Therefore it's important to do performance testing with your options and workloads to determine how much benefit you can actually get from serialization optimization mode. Also, the ability to fall back to `JsonSerializer` code requires metadata collection mode. If you select only serialization optimization mode, serialization might fail for types or options that need to fall back to `JsonSerializer` code.

How to use source generation modes

Most of the System.Text.Json documentation shows how to write code that uses reflection mode. For information about how to use source generation modes, see [How to use source generation in System.Text.Json](#).

See also

- [Try the new System.Text.Json source generator](#)
- [JSON serialization and deserialization in .NET - overview](#)
- [How to use the library](#)
- [Instantiate JsonSerializerOptions instances](#)
- [Enable case-insensitive matching](#)
- [Customize property names and values](#)
- [Ignore properties](#)
- [Allow invalid JSON](#)
- [Handle overflow JSON or use JsonElement or JsonNode](#)
- [Preserve references and handle circular references](#)
- [Deserialize to immutable types and non-public accessors](#)
- [Polymorphic serialization](#)
- [Migrate from Newtonsoft.Json to System.Text.Json](#)
- [Customize character encoding](#)
- [Use DOM, Utf8JsonReader, and Utf8JsonWriter](#)
- [Write custom converters for JSON serialization](#)
- [DateTime and DateTimeOffset support](#)
- [Supported collection types in System.Text.Json](#)
- [System.Text.Json API reference](#)
- [System.Text.Json.Serialization API reference](#)

How to serialize and deserialize (marshal and unmarshal) JSON in .NET

9/20/2022 • 17 minutes to read • [Edit Online](#)

This article shows how to use the `System.Text.Json` namespace to serialize to and deserialize from JavaScript Object Notation (JSON). If you're porting existing code from `Newtonsoft.Json`, see [How to migrate to `System.Text.Json`](#).

Code samples

The code samples in this article:

- Use the library directly, not through a framework such as [ASP.NET Core](#).
- Use the `JsonSerializer` class with custom types to serialize from and deserialize into.

For information about how to read and write JSON data without using `JsonSerializer`, see [How to use the JSON DOM, `Utf8JsonReader`, and `Utf8JsonWriter`](#).

- Use the `WriteIndented` option to format the JSON for human readability when that is helpful.

For production use, you would typically accept the default value of `false` for this setting, since adding unnecessary whitespace may incur a negative impact on performance and bandwidth usage.

- Refer to the following class and variants of it:

```
public class WeatherForecast
{
    public DateTimeOffset Date { get; set; }
    public int TemperatureCelsius { get; set; }
    public string? Summary { get; set; }
}
```

```
Public Class WeatherForecast
    Public Property [Date] As DateTimeOffset
    Public Property TemperatureCelsius As Integer
    Public Property Summary As String
End Class
```

Visual Basic support

Parts of `System.Text.Json` use [ref structs](#), which are not supported by Visual Basic. If you try to use `System.Text.Json` ref struct APIs with Visual Basic you get BC40000 compiler errors. The error message indicates that the problem is an obsolete API, but the actual issue is lack of ref struct support in the compiler. The following parts of `System.Text.Json` aren't usable from Visual Basic:

- The `Utf8JsonReader` class. Since the `JsonConverter<T>.Read` method takes a `Utf8JsonReader` parameter, this limitation means you can't use Visual Basic to write custom converters. A workaround for this is to implement custom converters in a C# library assembly, and reference that assembly from your VB project. This assumes that all you do in Visual Basic is register the converters into the serializer. You can't call the `Read` methods of the converters from Visual Basic code.

- Overloads of other APIs that include a `ReadOnlySpan<T>` type. Most methods include overloads that use `String` instead of `ReadOnlySpan`.

These restrictions are in place because ref structs cannot be used safely without language support, even when just "passing data through." Subverting this error will result in Visual Basic code that can corrupt memory and should not be done.

Namespaces

The `System.Text.Json` namespace contains all the entry points and the main types. The `System.Text.Json.Serialization` namespace contains attributes and APIs for advanced scenarios and customization specific to serialization and deserialization. The code examples shown in this article require `using` directives for one or both of these namespaces:

```
using System.Text.Json;  
using System.Text.Json.Serialization;
```

```
Imports System.Text.Json  
Imports System.Text.Json.Serialization
```

IMPORTANT

Attributes from the `System.Runtime.Serialization` namespace aren't supported in `System.Text.Json`.

How to write .NET objects as JSON (serialize)

To write JSON to a string or to a file, call the `JsonSerializer.Serialize` method.

The following example creates JSON as a string:

```
using System.Text.Json;

namespace SerializeBasic
{
    public class WeatherForecast
    {
        public DateTimeOffset Date { get; set; }
        public int TemperatureCelsius { get; set; }
        public string? Summary { get; set; }
    }

    public class Program
    {
        public static void Main()
        {
            var weatherForecast = new WeatherForecast
            {
                Date = DateTime.Parse("2019-08-01"),
                TemperatureCelsius = 25,
                Summary = "Hot"
            };

            string jsonString = JsonSerializer.Serialize(weatherForecast);

            Console.WriteLine(jsonString);
        }
    }
}

// output:
//>{"Date":"2019-08-01T00:00:00-07:00","TemperatureCelsius":25,"Summary":"Hot"}
```

```
Dim jsonString As String
```

The JSON output is minified (whitespace, indentation, and new-line characters are removed) by default.

The following example uses synchronous code to create a JSON file:

```
using System.Text.Json;

namespace SerializeToFile
{
    public class WeatherForecast
    {
        public DateTimeOffset Date { get; set; }
        public int TemperatureCelsius { get; set; }
        public string? Summary { get; set; }
    }

    public class Program
    {
        public static void Main()
        {
            var weatherForecast = new WeatherForecast
            {
                Date = DateTime.Parse("2019-08-01"),
                TemperatureCelsius = 25,
                Summary = "Hot"
            };

            string fileName = "WeatherForecast.json";
            string jsonString = JsonSerializer.Serialize(weatherForecast);
            File.WriteAllText(fileName, jsonString);

            Console.WriteLine(File.ReadAllText(fileName));
        }
    }
}

// output:
//{"Date":"2019-08-01T00:00:00-07:00","TemperatureCelsius":25,"Summary":"Hot"}
```

```
jsonString = JsonSerializer.Serialize(weatherForecast1)
File.WriteAllText(fileName, jsonString)
```

The following example uses asynchronous code to create a JSON file:

```

using System.Text.Json;

namespace SerializeToFileAsync
{
    public class WeatherForecast
    {
        public DateTimeOffset Date { get; set; }
        public int TemperatureCelsius { get; set; }
        public string? Summary { get; set; }
    }

    public class Program
    {
        public static async Task Main()
        {
            var weatherForecast = new WeatherForecast
            {
                Date = DateTime.Parse("2019-08-01"),
                TemperatureCelsius = 25,
                Summary = "Hot"
            };

            string fileName = "WeatherForecast.json";
            using FileStream createStream = File.Create(fileName);
            await JsonSerializer.SerializeAsync(createStream, weatherForecast);
            await createStream.DisposeAsync();

            Console.WriteLine(File.ReadAllText(fileName));
        }
    }
}
// output:
//{"Date":"2019-08-01T00:00:00-07:00","TemperatureCelsius":25,"Summary":"Hot"}

```

```

Dim createStream As FileStream = File.Create(fileName)
Await JsonSerializer.SerializeAsync(createStream, weatherForecast1)

```

The preceding examples use type inference for the type being serialized. An overload of `Serialize()` takes a generic type parameter:

```

using System.Text.Json;

namespace SerializeWithGenericParameter
{
    public class WeatherForecast
    {
        public DateTimeOffset Date { get; set; }
        public int TemperatureCelsius { get; set; }
        public string? Summary { get; set; }
    }

    public class Program
    {
        public static void Main()
        {
            var weatherForecast = new WeatherForecast
            {
                Date = DateTime.Parse("2019-08-01"),
                TemperatureCelsius = 25,
                Summary = "Hot"
            };

            string jsonString = JsonSerializer.Serialize<WeatherForecast>(weatherForecast);

            Console.WriteLine(jsonString);
        }
    }
}

// output:
//{"Date":"2019-08-01T00:00:00-07:00","TemperatureCelsius":25,"Summary":"Hot"}

```

```
jsonString = JsonSerializer.Serialize(Of WeatherForecastWithPOCOs)(weatherForecast)
```

Serialization example

Here's an example showing how a class that contains collection properties and a user-defined type is serialized:

```

using System.Text.Json;

namespace SerializeExtra
{
    public class WeatherForecast
    {
        public DateTimeOffset Date { get; set; }
        public int TemperatureCelsius { get; set; }
        public string? Summary { get; set; }
        public string? SummaryField;
        public IList<DateTimeOffset>? DatesAvailable { get; set; }
        public Dictionary<string, HighLowTemps>? TemperatureRanges { get; set; }
        public string[]? SummaryWords { get; set; }
    }

    public class HighLowTemps
    {
        public int High { get; set; }
        public int Low { get; set; }
    }

    public class Program
    {
        public static void Main()
        {
            var weatherForecast = new WeatherForecast
            {
                Date = DateTimeOffset.Parse("2019-08-01")
            }
        }
    }
}

```

```
        Date = DateTime.Parse("2019-08-01"),
        TemperatureCelsius = 25,
        Summary = "Hot",
        SummaryField = "Hot",
        DatesAvailable = new List<DateTimeOffset>()
            { DateTime.Parse("2019-08-01"), DateTime.Parse("2019-08-02") },
        TemperatureRanges = new Dictionary<string, HighLowTemps>
        {
            ["Cold"] = new HighLowTemps { High = 20, Low = -10 },
            ["Hot"] = new HighLowTemps { High = 60, Low = 20 }
        },
        SummaryWords = new[] { "Cool", "Windy", "Humid" }
    };

    var options = new JsonSerializerOptions { WriteIndented = true };
    string jsonString = JsonSerializer.Serialize(weatherForecast, options);

    Console.WriteLine(jsonString);
}
}
}

// output:
//{
//    "Date": "2019-08-01T00:00:00-07:00",
//    "TemperatureCelsius": 25,
//    "Summary": "Hot",
//    "DatesAvailable": [
//        "2019-08-01T00:00:00-07:00",
//        "2019-08-02T00:00:00-07:00"
//    ],
//    "TemperatureRanges": {
//        "Cold": {
//            "High": 20,
//            "Low": -10
//        },
//        "Hot": {
//            "High": 60,
//            "Low": 20
//        }
//    },
//    "SummaryWords": [
//        "Cool",
//        "Windy",
//        "Humid"
//    ]
//}
```

```

Public Class WeatherForecastWithPOCOs
    Public Property [Date] As DateTimeOffset
    Public Property TemperatureCelsius As Integer
    Public Property Summary As String
    Public SummaryField As String
    Public Property DatesAvailable As IList(Of DateTimeOffset)
    Public Property TemperatureRanges As Dictionary(Of String, HighLowTemps)
    Public Property SummaryWords As String()
End Class

Public Class HighLowTemps
    Public Property High As Integer
    Public Property Low As Integer
End Class

' serialization output formatted (pretty-printed with whitespace and indentation):
' {
'     "Date": "2019-08-01T00:00:00-07:00",
'     "TemperatureCelsius": 25,
'     "Summary": "Hot",
'     "DatesAvailable": [
'         "2019-08-01T00:00:00-07:00",
'         "2019-08-02T00:00:00-07:00"
'     ],
'     "TemperatureRanges": {
'         "Cold": {
'             "High": 20,
'             "Low": -10
'         },
'         "Hot": {
'             "High": 60,
'             "Low": 20
'         }
'     },
'     "SummaryWords": [
'         "Cool",
'         "Windy",
'         "Humid"
'     ]
' }

```

Serialize to UTF-8

Serializing to a UTF-8 byte array is about 5-10% faster than using the string-based methods. The difference is because the bytes (as UTF-8) don't need to be converted to strings (UTF-16).

To serialize to a UTF-8 byte array, call the [JsonSerializer.SerializeToUtf8Bytes](#) method:

```
byte[] jsonUtf8Bytes = JsonSerializer.SerializeToUtf8Bytes(weatherForecast);
```

```

Dim jsonUtf8Bytes As Byte()
Dim options As JsonSerializerOptions = New JsonSerializerOptions With {
    .WriteIndented = True
}
jsonUtf8Bytes = JsonSerializer.SerializeToUtf8Bytes(weatherForecast1, options)

```

A [Serialize](#) overload that takes a [Utf8JsonWriter](#) is also available.

Serialization behavior

- By default, all public properties are serialized. You can [specify properties to ignore](#).
- The [default encoder](#) escapes non-ASCII characters, HTML-sensitive characters within the ASCII-range, and characters that must be escaped according to [the RFC 8259 JSON spec](#).
- By default, JSON is minified. You can [pretty-print the JSON](#).
- By default, casing of JSON names matches the .NET names. You can [customize JSON name casing](#).
- By default, circular references are detected and exceptions thrown. You can [preserve references and handle circular references](#).
- By default, [fields](#) are ignored. You can [include fields](#).

When you use `System.Text.Json` indirectly in an ASP.NET Core app, some default behaviors are different. For more information, see [Web defaults for `JsonSerializerOptions`](#).

- By default, all public properties are serialized. You can [specify properties to ignore](#).
- The [default encoder](#) escapes non-ASCII characters, HTML-sensitive characters within the ASCII-range, and characters that must be escaped according to [the RFC 8259 JSON spec](#).
- By default, JSON is minified. You can [pretty-print the JSON](#).
- By default, casing of JSON names matches the .NET names. You can [customize JSON name casing](#).
- Circular references are detected and exceptions thrown.
- [Fields](#) are ignored.

Supported types include:

- .NET primitives that map to JavaScript primitives, such as numeric types, strings, and Boolean.
- User-defined [plain old CLR objects \(POCOs\)](#).
- One-dimensional and jagged arrays (`T[][]`).
- Collections and dictionaries from the following namespaces.
 - [System.Collections](#)
 - [System.Collections.Generic](#)
 - [System.Collections.Immutable](#)
 - [System.Collections.Concurrent](#)
 - [System.Collections.Specialized](#)
 - [System.Collections.ObjectModel](#)
- .NET primitives that map to JavaScript primitives, such as numeric types, strings, and Boolean.
- User-defined [plain old CLR objects \(POCOs\)](#).
- One-dimensional and jagged arrays (`ArrayName[][]`).
- `Dictionary<string, TValue>` where `TValue` is `object`, `JsonElement`, or a POCO.
- Collections from the following namespaces.
 - [System.Collections](#)
 - [System.Collections.Generic](#)
 - [System.Collections.Immutable](#)
 - [System.Collections.Concurrent](#)
 - [System.Collections.Specialized](#)
 - [System.Collections.ObjectModel](#)

For more information, see [Supported collection types in `System.Text.Json`](#).

You can [implement custom converters](#) to handle additional types or to provide functionality that isn't supported by the built-in converters.

How to read JSON as .NET objects (deserialize)

A common way to deserialize JSON is to first create a class with properties and fields that represent one or more of the JSON properties. Then, to deserialize from a string or a file, call the [JsonSerializer.Deserialize](#) method. For the generic overloads, you pass the type of the class you created as the generic type parameter. For the non-generic overloads, you pass the type of the class you created as a method parameter. You can deserialize either synchronously or asynchronously. Any JSON properties that aren't represented in your class are ignored.

The following example shows how to deserialize a JSON string:

```

using System.Text.Json;

namespace DeserializeExtra
{
    public class WeatherForecast
    {
        public DateTimeOffset Date { get; set; }
        public int TemperatureCelsius { get; set; }
        public string? Summary { get; set; }
        public string? SummaryField;
        public IList<DateTimeOffset>? DatesAvailable { get; set; }
        public Dictionary<string, HighLowTemps>? TemperatureRanges { get; set; }
        public string[]? SummaryWords { get; set; }
    }

    public class HighLowTemps
    {
        public int High { get; set; }
        public int Low { get; set; }
    }

    public class Program
    {
        public static void Main()
        {
            string jsonString =
@"
{
    ""Date"": ""2019-08-01T00:00:00-07:00"",
    ""TemperatureCelsius"": 25,
    ""Summary"": ""Hot"",
    ""DatesAvailable"": [
        ""2019-08-01T00:00:00-07:00"",
        ""2019-08-02T00:00:00-07:00""
    ],
    ""TemperatureRanges"": {
        ""Cold"": {
            ""High"": 20,
            ""Low"": -10
        },
        ""Hot"": {
            ""High"": 60,
            ""Low"": 20
        }
    },
    ""SummaryWords"": [
        ""Cool"",
        ""Windy"",
        ""Humid""
    ]
}
";
        }

        WeatherForecast? weatherForecast =
            JsonSerializer.Deserialize<WeatherForecast>(jsonString);

        Console.WriteLine($"Date: {weatherForecast?.Date}");
        Console.WriteLine($"TemperatureCelsius: {weatherForecast?.TemperatureCelsius}");
        Console.WriteLine($"Summary: {weatherForecast?.Summary}");
    }
}

// output:
//Date: 8/1/2019 12:00:00 AM -07:00
//TemperatureCelsius: 25
//Summary: Hot

```

```
weatherForecast = JsonSerializer.Deserialize(Of WeatherForecastWithPOCOs)(jsonString)
```

To deserialize from a file by using synchronous code, read the file into a string, as shown in the following example:

```
using System.Text.Json;

namespace DeserializeFromFile
{
    public class WeatherForecast
    {
        public DateTimeOffset Date { get; set; }
        public int TemperatureCelsius { get; set; }
        public string? Summary { get; set; }
    }

    public class Program
    {
        public static void Main()
        {
            string fileName = "WeatherForecast.json";
            string jsonString = File.ReadAllText(fileName);
            WeatherForecast weatherForecast = JsonSerializer.Deserialize<WeatherForecast>(jsonString)!

            Console.WriteLine($"Date: {weatherForecast.Date}");
            Console.WriteLine($"TemperatureCelsius: {weatherForecast.TemperatureCelsius}");
            Console.WriteLine($"Summary: {weatherForecast.Summary}");
        }
    }
}

// output:
//Date: 8/1/2019 12:00:00 AM -07:00
//TemperatureCelsius: 25
//Summary: Hot
```

```
jsonString = File.ReadAllText(fileName)
weatherForecast1 = JsonSerializer.Deserialize(Of WeatherForecast)(jsonString)
```

To deserialize from a file by using asynchronous code, call the [DeserializeAsync](#) method:

```

using System.Text.Json;

namespace DeserializeFromFileAsync
{
    public class WeatherForecast
    {
        public DateTimeOffset Date { get; set; }
        public int TemperatureCelsius { get; set; }
        public string? Summary { get; set; }
    }

    public class Program
    {
        public static async Task Main()
        {
            string fileName = "WeatherForecast.json";
            using FileStream openStream = File.OpenRead(fileName);
            WeatherForecast? weatherForecast =
                await JsonSerializer.DeserializeAsync<WeatherForecast>(openStream);

            Console.WriteLine($"Date: {weatherForecast?.Date}");
            Console.WriteLine($"TemperatureCelsius: {weatherForecast?.TemperatureCelsius}");
            Console.WriteLine($"Summary: {weatherForecast?.Summary}");
        }
    }
}

// output:
//Date: 8/1/2019 12:00:00 AM -07:00
//TemperatureCelsius: 25
//Summary: Hot

```

```

Dim openStream As FileStream = File.OpenRead(fileName)
weatherForecast1 = Await JsonSerializer.Deserialize(Of WeatherForecast)(openStream)

```

TIP

If you have JSON that you want to deserialize, and you don't have the class to deserialize it into, you have options other than manually creating the class that you need:

- Deserialize into a [JSON DOM \(document object model\)](#) and extract what you need from the DOM.
The DOM lets you navigate to a subsection of a JSON payload and deserialize a single value, a custom type, or an array. For information about the [JsonNode](#) DOM in .NET 6, see [Deserialize subsections of a JSON payload](#). For information about the [JsonDocument](#) DOM, see [How to search a JsonDocument and JsonElement for sub-elements](#).
- Use the [Utf8JsonReader](#) directly.
- Use Visual Studio 2022 to automatically generate the class you need:
 - Copy the JSON that you need to deserialize.
 - Create a class file and delete the template code.
 - Choose **Edit > Paste Special > Paste JSON as Classes**. The result is a class that you can use for your deserialization target.

Deserialize from UTF-8

To deserialize from UTF-8, call a [JsonSerializer.Deserialize](#) overload that takes a `ReadOnlySpan<byte>` or a `Utf8JsonReader`, as shown in the following examples. The examples assume the JSON is in a byte array named `jsonUtf8Bytes`.

```
var readOnlySpan = new ReadOnlySpan<byte>(jsonUtf8Bytes);
WeatherForecast deserializedWeatherForecast =
    JsonSerializer.Deserialize<WeatherForecast>(readOnlySpan)!;
```

```
Dim jsonString = Encoding.UTF8.GetString(jsonUtf8Bytes)
weatherForecast1 = JsonSerializer.Deserialize(Of WeatherForecast)(jsonString)
```

```
var utf8Reader = new Utf8JsonReader(jsonUtf8Bytes);
WeatherForecast deserializedWeatherForecast =
    JsonSerializer.Deserialize<WeatherForecast>(ref utf8Reader)!;
```

' This code example doesn't apply to Visual Basic. For more information, go to the following URL:
' <https://docs.microsoft.com/dotnet/standard/serialization/system-text-json-how-to#visual-basic-support>

Deserialization behavior

The following behaviors apply when deserializing JSON:

- By default, property name matching is case-sensitive. You can [specify case-insensitivity](#).
- If the JSON contains a value for a read-only property, the value is ignored and no exception is thrown.
- Non-public constructors are ignored by the serializer.
- Deserialization to immutable objects or properties that don't have public `set` accessors is supported. See [Immutable types and Records](#).
- By default, enums are supported as numbers. You can [serialize enum names as strings](#).
- By default, fields are ignored. You can [include fields](#).
- By default, comments or trailing commas in the JSON throw exceptions. You can [allow comments and trailing commas](#).
- The [default maximum depth](#) is 64.

When you use `System.Text.Json` indirectly in an ASP.NET Core app, some default behaviors are different. For more information, see [Web defaults for `JsonSerializerOptions`](#).

- By default, property name matching is case-sensitive. You can [specify case-insensitivity](#). ASP.NET Core apps [specify case-insensitivity by default](#).
- If the JSON contains a value for a read-only property, the value is ignored and no exception is thrown.
- A parameterless constructor, which can be public, internal, or private, is used for deserialization.
- Deserialization to immutable objects or properties that don't have public `set` accessors isn't supported.
- By default, enums are supported as numbers. You can [serialize enum names as strings](#).
- Fields aren't supported.
- By default, comments or trailing commas in the JSON throw exceptions. You can [allow comments and trailing commas](#).
- The [default maximum depth](#) is 64.

When you use `System.Text.Json` indirectly in an ASP.NET Core app, some default behaviors are different. For more information, see [Web defaults for `JsonSerializerOptions`](#).

You can [implement custom converters](#) to provide functionality that isn't supported by the built-in converters.

Serialize to formatted JSON

To pretty-print the JSON output, set `JsonSerializerOptions.WriteIndented` to `true`:

```
using System.Text.Json;

namespace SerializeWriteIndented
{
    public class WeatherForecast
    {
        public DateTimeOffset Date { get; set; }
        public int TemperatureCelsius { get; set; }
        public string? Summary { get; set; }
    }

    public class Program
    {
        public static void Main()
        {
            var weatherForecast = new WeatherForecast
            {
                Date = DateTime.Parse("2019-08-01"),
                TemperatureCelsius = 25,
                Summary = "Hot"
            };

            var options = new JsonSerializerOptions { WriteIndented = true };
            string jsonString = JsonSerializer.Serialize(weatherForecast, options);

            Console.WriteLine(jsonString);
        }
    }
}

// output:
//{
//    "Date": "2019-08-01T00:00:00-07:00",
//    "TemperatureCelsius": 25,
//    "Summary": "Hot"
//}
```

```
Dim options As JsonSerializerOptions = New JsonSerializerOptions With {
    .WriteIndented = True
}
jsonString = JsonSerializer.Serialize(weatherForecast, options)
```

If you use `JsonSerializerOptions` repeatedly with the same options, don't create a new `JsonSerializerOptions` instance each time you use it. Reuse the same instance for every call. For more information, see [Reuse `JsonSerializerOptions` instances](#).

Include fields

Use the `JsonSerializerOptions.IncludeFields` global setting or the `[JsonInclude]` attribute to include fields when serializing or deserializing, as shown in the following example:

```
using System.Text.Json;
using System.Text.Json.Serialization;

namespace Fields
{
    public class Forecast
    {
        public DateTime Date;
        public int TemperatureC;
        public string? Summary;
    }
}
```

```

}

public class Forecast2
{
    [JsonProperty]
    public DateTime Date;
    [JsonProperty]
    public int TemperatureC;
    [JsonProperty]
    public string? Summary;
}

public class Program
{
    public static void Main()
    {
        var json =
            @"""Date"":""2020-09-06T11:31:01.923395""",""TemperatureC"":-1,""Summary"":""Cold"""} ";
        Console.WriteLine($"Input JSON: {json}");

        var options = new JsonSerializerOptions
        {
            IncludeFields = true,
        };
        var forecast = JsonSerializer.Deserialize<Forecast>(json, options)!;

        Console.WriteLine($"forecast.Date: {forecast.Date}");
        Console.WriteLine($"forecast.TemperatureC: {forecast.TemperatureC}");
        Console.WriteLine($"forecast.Summary: {forecast.Summary}");

        var roundTrippedJson =
            JsonSerializer.Serialize<Forecast>(forecast, options);

        Console.WriteLine($"Output JSON: {roundTrippedJson}");

        var forecast2 = JsonSerializer.Deserialize<Forecast2>(json)!;

        Console.WriteLine($"forecast2.Date: {forecast2.Date}");
        Console.WriteLine($"forecast2.TemperatureC: {forecast2.TemperatureC}");
        Console.WriteLine($"forecast2.Summary: {forecast2.Summary}");

        roundTrippedJson = JsonSerializer.Serialize<Forecast2>(forecast2);

        Console.WriteLine($"Output JSON: {roundTrippedJson}");
    }
}
}

// Produces output like the following example:
// 
//Input JSON: { "Date":"2020-09-06T11:31:01.923395","TemperatureC":-1,"Summary":"Cold"}
//forecast.Date: 9/6/2020 11:31:01 AM
//forecast.TemperatureC: -1
//forecast.Summary: Cold
//Output JSON: { "Date":"2020-09-06T11:31:01.923395","TemperatureC":-1,"Summary":"Cold"}
//forecast2.Date: 9/6/2020 11:31:01 AM
//forecast2.TemperatureC: -1
//forecast2.Summary: Cold
//Output JSON: { "Date":"2020-09-06T11:31:01.923395","TemperatureC":-1,"Summary":"Cold"} 
```

```

Imports System.Text.Json
Imports System.Text.Json.Serialization

Namespace Fields

    Public Class Forecast
        Public [Date] As Date
        ... 
```

```

    Public TemperatureC As Integer
    Public Summary As String
End Class

Public Class Forecast2

    <JsonPropertyName>
    Public [Date] As Date

    <JsonPropertyName>
    Public TemperatureC As Integer

    <JsonPropertyName>
    Public Summary As String

End Class

Public NotInheritable Class Program

    Public Shared Sub Main()
        Dim json As String = "{\"Date\":\"2020-09-06T11:31:01.923395\",\"TemperatureC\":-1,\"Summary\":\"Cold\"}"
        Console.WriteLine($"Input JSON: {json}")

        Dim options As New JsonSerializerOptions With {
            .IncludeFields = True
        }
        Dim forecast1 As Forecast = JsonSerializer.Deserialize(Of Forecast)(json, options)

        Console.WriteLine($"forecast.Date: {forecast1.[Date]}")
        Console.WriteLine($"forecast.TemperatureC: {forecast1.TemperatureC}")
        Console.WriteLine($"forecast.Summary: {forecast1.Summary}")

        Dim roundTrippedJson As String = JsonSerializer.Serialize(forecast1, options)

        Console.WriteLine($"Output JSON: {roundTrippedJson}")

        Dim forecast21 As Forecast2 = JsonSerializer.Deserialize(Of Forecast2)(json)

        Console.WriteLine($"forecast2.Date: {forecast21.[Date]}")
        Console.WriteLine($"forecast2.TemperatureC: {forecast21.TemperatureC}")
        Console.WriteLine($"forecast2.Summary: {forecast21.Summary}")

        roundTrippedJson = JsonSerializer.Serialize(forecast21)

        Console.WriteLine($"Output JSON: {roundTrippedJson}")
    End Sub

End Class

End Namespace

' Produces output like the following example:
'
'Input JSON: { "Date":"2020-09-06T11:31:01.923395","TemperatureC":-1,"Summary":"Cold"}
'forecast.Date: 9/6/2020 11:31:01 AM
'forecast.TemperatureC: -1
'forecast.Summary: Cold
'Output JSON: { "Date":"2020-09-06T11:31:01.923395","TemperatureC":-1,"Summary":"Cold"}
'forecast2.Date: 9/6/2020 11:31:01 AM
'forecast2.TemperatureC: -1
'forecast2.Summary: Cold
'Output JSON: { "Date":"2020-09-06T11:31:01.923395","TemperatureC":-1,"Summary":"Cold"}
```

To ignore read-only fields, use the [JsonSerializerOptions.IgnoreReadOnlyFields](#) global setting.

Fields are not supported in System.Text.Json in .NET Core 3.1. [Custom converters](#) can provide this functionality.

HttpClient and HttpContent extension methods

Serializing and deserializing JSON payloads from the network are common operations. Extension methods on [HttpClient](#) and [HttpContent](#) let you do these operations in a single line of code. These extension methods use [web defaults for JsonSerializerOptions](#).

The following example illustrates use of [HttpClientJsonExtensions.GetFromJsonAsync](#) and [HttpClientJsonExtensions.PostAsJsonAsync](#):

```
using System.Net.Http.Json;

namespace HttpClientExtensionMethods
{
    public class User
    {
        public int Id { get; set; }
        public string? Name { get; set; }
        public string? Username { get; set; }
        public string? Email { get; set; }
    }

    public class Program
    {
        public static async Task Main()
        {
            using HttpClient client = new()
            {
                BaseAddress = new Uri("https://jsonplaceholder.typicode.com")
            };

            // Get the user information.
            User? user = await client.GetFromJsonAsync<User>("users/1");
            Console.WriteLine($"Id: {user?.Id}");
            Console.WriteLine($"Name: {user?.Name}");
            Console.WriteLine($"Username: {user?.Username}");
            Console.WriteLine($"Email: {user?.Email}");

            // Post a new user.
            HttpResponseMessage response = await client.PostAsJsonAsync("users", user);
            Console.WriteLine(
                $"{{(response.IsSuccessStatusCode ? "Success" : "Error")}} - {response.StatusCode}");
        }
    }
}

// Produces output like the following example but with different names:
// 
//Id: 1
//Name: Tyler King
//Username: Tyler
//Email: Tyler @contoso.com
//Success - Created
```

```

Imports System.Net.Http
Imports System.Net.Http.Json

Namespace HttpClientExtensionMethods

    Public Class User
        Public Property Id As Integer
        Public Property Name As String
        Public Property Username As String
        Public Property Email As String
    End Class

    Public Class Program

        Public Shared Async Function Main() As Task
            Using client As New HttpClient With {
                .BaseAddress = New Uri("https://jsonplaceholder.typicode.com")
            }

                ' Get the user information.
                Dim user1 As User = Await client.GetFromJsonAsync(Of User)("users/1")
                Console.WriteLine($"Id: {user1.Id}")
                Console.WriteLine($"Name: {user1.Name}")
                Console.WriteLine($"Username: {user1.Username}")
                Console.WriteLine($"Email: {user1.Email}")

                ' Post a new user.
                Dim response As HttpResponseMessage = Await client.PostAsJsonAsync("users", user1)
                Console.WriteLine(
                    $"{{(If(response.IsSuccessStatusCode, "Success", "Error"))} - {response.StatusCode}}")
            End Using
        End Function

    End Class

End Namespace

' Produces output like the following example but with different names:
'
'Id: 1
'Name: Tyler King
'Username: Tyler
'Email: Tyler @contoso.com
'Success - Created

```

There are also extension methods for `System.Text.Json` on `HttpClient`.

Extension methods on `HttpClient` and `HttpContent` are not available in `System.Text.Json` in .NET Core 3.1.

See also

- [System.Text.Json overview](#)
- [Instantiate JsonSerializerOptions instances](#)
- [Enable case-insensitive matching](#)
- [Customize property names and values](#)
- [Ignore properties](#)
- [Allow invalid JSON](#)
- [Handle overflow JSON or use JsonElement or JsonNode](#)
- [Preserve references and handle circular references](#)
- [Deserialize to immutable types and non-public accessors](#)
- [Polymorphic serialization](#)

- [Migrate from Newtonsoft.Json to System.Text.Json](#)
- [Customize character encoding](#)
- [Use DOM, Utf8JsonReader, and Utf8JsonWriter](#)
- [Write custom converters for JSON serialization](#)
- [DateTime and DateTimeOffset support](#)
- [How to use source generation](#)
- [Supported collection types](#)
- [System.Text.Json API reference](#)
- [System.Text.Json.Serialization API reference](#)

How to instantiate JsonSerializerOptions instances with System.Text.Json

9/20/2022 • 5 minutes to read • [Edit Online](#)

This article explains how to avoid performance problems when you use [JsonSerializerOptions](#). It also shows how to use the parameterized constructors that are available.

Reuse JsonSerializerOptions instances

If you use `JsonSerializerOptions` repeatedly with the same options, don't create a new `JsonSerializerOptions` instance each time you use it. Reuse the same instance for every call. This guidance applies to code you write for custom converters and when you call [JsonSerializer.Serialize](#) or [JsonSerializer.Deserialize](#). It's safe to use the same instance across multiple threads. The metadata caches on the options instance are thread-safe, and the instance is immutable after the first serialization or deserialization.

The following code demonstrates the performance penalty for using new options instances.

```

using System.Diagnostics;
using System.Text.Json;

namespace OptionsPerfDemo
{
    public record Forecast(DateTime Date, int TemperatureC, string Summary);

    public class Program
    {
        public static void Main()
        {
            Forecast forecast = new(DateTime.Now, 40, "Hot");
            JsonSerializerOptions options = new() { WriteIndented = true };
            int iterations = 100000;

            var watch = Stopwatch.StartNew();
            for (int i = 0; i < iterations; i++)
            {
                Serialize(forecast, options);
            }
            watch.Stop();
            Console.WriteLine($"Elapsed time using one options instance: {watch.ElapsedMilliseconds}");

            watch = Stopwatch.StartNew();
            for (int i = 0; i < iterations; i++)
            {
                Serialize(forecast);
            }
            watch.Stop();
            Console.WriteLine($"Elapsed time creating new options instances: {watch.ElapsedMilliseconds}");
        }

        private static void Serialize(Forecast forecast, JsonSerializerOptions? options = null)
        {
            _ = JsonSerializer.Serialize<Forecast>(forecast,
                options ?? new JsonSerializerOptions() { WriteIndented = true });
        }
    }
}

// Produces output like the following example:
// 
//Elapsed time using one options instance: 190
//Elapsed time creating new options instances: 40140

```

The preceding code serializes a small object 100,000 times using the same options instance. Then it serializes the same object the same number of times and creates a new options instance each time. A typical run time difference is 190 compared to 40,140 milliseconds. The difference is even greater if you increase the number of iterations.

The serializer undergoes a warm-up phase during the first serialization of each type in the object graph when a new options instance is passed to it. This warm-up includes creating a cache of metadata that is needed for serialization. The metadata includes delegates to property getters, setters, constructor arguments, specified attributes, and so forth. This metadata cache is stored in the options instance. The same warm-up process and cache applies to deserialization.

The size of the metadata cache in a `JsonSerializerOptions` instance depends on the number of types to be serialized. If you pass numerous types—for example, dynamically generated types—to the serializer, the cache size will continue to grow and can end up causing an `OutOfMemoryException`.

The `JsonSerializerOptions.Default` property

If the instance of `JsonSerializerOptions` that you need to use is the default instance (has all of the default settings and the default converters), use the `JsonSerializerOptions.Default` property rather than creating an options instance. For more information, see [Use default system converter](#).

Copy JsonSerializerOptions

There is a `JsonSerializerOptions constructor` that lets you create a new instance with the same options as an existing instance, as shown in the following example:

```
using System.Text.Json;

namespace CopyOptions
{
    public class Forecast
    {
        public DateTime Date { get; init; }
        public int TemperatureC { get; set; }
        public string? Summary { get; set; }
    };

    public class Program
    {
        public static void Main()
        {
            Forecast forecast = new()
            {
                Date = DateTime.Now,
                TemperatureC = 40,
                Summary = "Hot"
            };

            JsonSerializerOptions options = new()
            {
                WriteIndented = true
            };

            JsonSerializerOptions optionsCopy = new(options);
            string forecastJson =
                JsonSerializer.Serialize<Forecast>(forecast, optionsCopy);

            Console.WriteLine($"Output JSON:\n{forecastJson}");
        }
    }
}

// Produces output like the following example:
// 
//Output JSON:
//{
//    "Date": "2020-10-21T15:40:06.8998502-07:00",
//    "TemperatureC": 40,
//    "Summary": "Hot"
//}
```

```

Imports System.Text.Json
Imports System.Text.Json.Serialization

Namespace CopyOptions

    Public Class Forecast
        Public Property [Date] As Date
        Public Property TemperatureC As Integer
        Public Property Summary As String
    End Class

    Public NotInheritable Class Program

        Public Shared Sub Main()
            Dim forecast1 As New Forecast() With {
                .[Date] = Date.Now,
                .Summary = Nothing,
                .TemperatureC = CType(Nothing, Integer)
            }

            Dim options As New JsonSerializerOptions() With {
                .DefaultIgnoreCondition = JsonIgnoreCondition.WhenWritingDefault
            }

            Dim optionsCopy As New JsonSerializerOptions
            Dim forecastJson As String = JsonSerializer.Serialize(forecast1, optionsCopy)

            Console.WriteLine($"Output JSON:{forecastJson}")
        End Sub

    End Class

End Namespace

' Produces output like the following example:
'
'Output JSON:
'{ 
    "Date": "2020-10-21T15:40:06.8998502-07:00",
    "TemperatureC": 40,
    "Summary": "Hot"
}'

```

The metadata cache of the existing `JsonSerializerOptions` instance isn't copied to the new instance. So using this constructor is not the same as reusing an existing instance of `JsonSerializerOptions`.

A `JsonSerializerOptions` constructor that takes an existing instance is not available in .NET Core 3.1.

Web defaults for JsonSerializerOptions

Here are the options that have different defaults for web apps:

- `PropertyNameCaseInsensitive` = `true`
- `JsonNamingPolicy` = `CamelCase`
- `NumberHandling` = `AllowReadingFromString`

There's a [JsonSerializerOptions constructor](#) that lets you create a new instance with the default options that ASP.NET Core uses for web apps, as shown in the following example:

```

using System.Text.Json;

namespace OptionsDefaults
{
    public class Forecast
    {
        public DateTime? Date { get; init; }
        public int TemperatureC { get; set; }
        public string? Summary { get; set; }
    };

    public class Program
    {
        public static void Main()
        {
            Forecast forecast = new()
            {
                Date = DateTime.Now,
                TemperatureC = 40,
                Summary = "Hot"
            };

            JsonSerializerOptions options = new(JsonSerializerDefaults.Web)
            {
                WriteIndented = true
            };

            Console.WriteLine(
                $"PropertyNameCaseInsensitive: {options.PropertyNameCaseInsensitive}");
            Console.WriteLine(
                $"JsonNamingPolicy: {options.JsonNamingPolicy}");
            Console.WriteLine(
                $"NumberHandling: {options.NumberHandling}");

            string forecastJson = JsonSerializer.Serialize<Forecast>(forecast, options);
            Console.WriteLine($"Output JSON:\n{forecastJson}");

            Forecast? forecastDeserialized =
                JsonSerializer.Deserialize<Forecast>(forecastJson, options);

            Console.WriteLine($"Date: {forecastDeserialized?.Date}");
            Console.WriteLine($"TemperatureC: {forecastDeserialized?.TemperatureC}");
            Console.WriteLine($"Summary: {forecastDeserialized?.Summary}");
        }
    }
}

// Produces output like the following example:
// 
//PropertyNameCaseInsensitive: True
//JsonNamingPolicy: System.Text.Json.JsonCamelCaseNamingPolicy
//NumberHandling: AllowReadingFromString
//Output JSON:
//{
//    "date": "2020-10-21T15:40:06.9040831-07:00",
//    "temperatureC": 40,
//    "summary": "Hot"
//}
//Date: 10 / 21 / 2020 3:40:06 PM
//TemperatureC: 40
//Summary: Hot

```

```

Imports System.Text.Json

Namespace OptionsDefaults

    Public Class Forecast
        Public Property [Date] As Date
        Public Property TemperatureC As Integer
        Public Property Summary As String
    End Class

    Public NotInheritable Class Program

        Public Shared Sub Main()
            Dim forecast1 As New Forecast() With {
                .[Date] = Date.Now,
                .TemperatureC = 40,
                .Summary = "Hot"
            }

            Dim options As New JsonSerializerOptions(JsonSerializerDefaults.Web) With {
                .WriteIndented = True
            }

            Console.WriteLine(
                $"PropertyNameCaseInsensitive: {options.PropertyNameCaseInsensitive}")
            Console.WriteLine(
                $"JsonNamingPolicy: {options.JsonNamingPolicy}")
            Console.WriteLine(
                $"NumberHandling: {options.NumberHandling}")

            Dim forecastJson As String = JsonSerializer.Serialize(forecast1, options)
            Console.WriteLine($"Output JSON:{forecastJson}")

            Dim forecastDeserialized As Forecast = JsonSerializer.Deserialize(Of Forecast)(forecastJson,
options)

            Console.WriteLine($"Date: {forecastDeserialized.[Date]}")
            Console.WriteLine($"TemperatureC: {forecastDeserialized.TemperatureC}")
            Console.WriteLine($"Summary: {forecastDeserialized.Summary}")
        End Sub

    End Class
End Namespace

' Produces output like the following example:
'
'PropertyNameCaseInsensitive: True
'JsonNamingPolicy: System.Text.Json.JsonCamelCaseNamingPolicy
'NumberHandling: AllowReadingFromString
'Output JSON:
'{
'    "date": "2020-10-21T15:40:06.9040831-07:00",
'    "temperatureC": 40,
'    "summary": "Hot"
'}
'Date: 10 / 21 / 2020 3:40:06 PM
'TemperatureC: 40
'Summary: Hot

```

Here are the options that have different defaults for web apps:

- `PropertyNameCaseInsensitive` = `true`
- `JsonNamingPolicy` = `CamelCase`

A `JsonSerializerOptions` constructor that specifies a set of defaults is not available in .NET Core 3.1.

See also

- [System.Text.Json overview](#)
- [How to serialize and deserialize JSON](#)
- [Enable case-insensitive matching](#)
- [Customize property names and values](#)
- [Ignore properties](#)
- [Allow invalid JSON](#)
- [Handle overflow JSON or use JsonElement or JsonNode](#)
- [Preserve references and handle circular references](#)
- [Deserialize to immutable types and non-public accessors](#)
- [Polymorphic serialization](#)
- [Migrate from Newtonsoft.Json to System.Text.Json](#)
- [Customize character encoding](#)
- [Use DOM, Utf8JsonReader, and Utf8JsonWriter](#)
- [Write custom converters for JSON serialization](#)
- [DateTime and DateTimeOffset support](#)
- [How to use source generation](#)
- [Supported collection types](#)
- [System.Text.Json API reference](#)
- [System.Text.Json.Serialization API reference](#)

How to enable case-insensitive property name matching with System.Text.Json

9/20/2022 • 2 minutes to read • [Edit Online](#)

In this article, you will learn how to enable case-insensitive property name matching with the `System.Text.Json` namespace.

Case-insensitive property matching

By default, deserialization looks for case-sensitive property name matches between JSON and the target object properties. To change that behavior, set `JsonSerializerOptions.PropertyNameCaseInsensitive` to `true`:

NOTE

The [web default](#) is case-insensitive.

```
var options = new JsonSerializerOptions
{
    PropertyNameCaseInsensitive = true
};
var weatherForecast = JsonSerializer.Deserialize<WeatherForecast>(jsonString, options);
```

```
Dim options As JsonSerializerOptions = New JsonSerializerOptions With {
    .PropertyNameCaseInsensitive = True
}
Dim weatherForecast1 = JsonSerializer.Deserialize(Of WeatherForecast)(jsonString, options)
```

Here's example JSON with camel case property names. It can be deserialized into the following type that has Pascal case property names.

```
{
    "date": "2019-08-01T00:00:00-07:00",
    "temperatureCelsius": 25,
    "summary": "Hot",
}
```

```
public class WeatherForecast
{
    public DateTimeOffset Date { get; set; }
    public int TemperatureCelsius { get; set; }
    public string? Summary { get; set; }
}
```

```
Public Class WeatherForecast
    Public Property [Date] As DateTimeOffset
    Public Property TemperatureCelsius As Integer
    Public Property Summary As String
End Class
```

See also

- [System.Text.Json overview](#)
- [How to serialize and deserialize JSON](#)
- [Instantiate JsonSerializerOptions instances](#)
- [Customize property names and values](#)
- [Ignore properties](#)
- [Allow invalid JSON](#)
- [Handle overflow JSON or use JsonElement or JsonNode](#)
- [Preserve references and handle circular references](#)
- [Deserialize to immutable types and non-public accessors](#)
- [Polymorphic serialization](#)
- [Migrate from Newtonsoft.Json to System.Text.Json](#)
- [Customize character encoding](#)
- [Use DOM, Utf8JsonReader, and Utf8JsonWriter](#)
- [Write custom converters for JSON serialization](#)
- [DateTime and DateTimeOffset support](#)
- [How to use source generation](#)
- [Supported collection types](#)
- [System.Text.Json API reference](#)
- [System.Text.Json.Serialization API reference](#)

How to customize property names and values with System.Text.Json

9/20/2022 • 6 minutes to read • [Edit Online](#)

By default, property names and dictionary keys are unchanged in the JSON output, including case. Enum values are represented as numbers. In this article, you'll learn how to:

- [Customize individual property names](#)
- [Convert all property names to camel case](#)
- [Implement a custom property naming policy](#)
- [Convert dictionary keys to camel case](#)
- [Convert enums to strings and camel case](#)
- [Configure the order of serialized properties](#)

NOTE

The [web default](#) is camel case.

For other scenarios that require special handling of JSON property names and values, you can [implement custom converters](#).

Customize individual property names

To set the name of individual properties, use the [\[JsonPropertyName\]](#) attribute.

Here's an example type to serialize and resulting JSON:

```
public class WeatherForecastWithPropertyNameAttribute
{
    public DateTimeOffset Date { get; set; }
    public int TemperatureCelsius { get; set; }
    public string? Summary { get; set; }
    [JsonPropertyName("Wind")]
    public int WindSpeed { get; set; }
}
```

```
Public Class WeatherForecastWithPropertyNameAttribute
    Public Property [Date] As DateTimeOffset
    Public Property TemperatureCelsius As Integer
    Public Property Summary As String

    <JsonPropertyName("Wind")>
    Public Property WindSpeed As Integer

End Class
```

```
{
    "Date": "2019-08-01T00:00:00-07:00",
    "TemperatureCelsius": 25,
    "Summary": "Hot",
    "Wind": 35
}
```

The property name set by this attribute:

- Applies in both directions, for serialization and deserialization.
- Takes precedence over property naming policies.
- [Doesn't affect parameter name matching for parameterized constructors.](#)

Use camel case for all JSON property names

To use camel case for all JSON property names, set `JsonSerializerOptions.PropertyNamingPolicy` to `JsonNamingPolicy.CamelCase`, as shown in the following example:

```
var serializeOptions = new JsonSerializerOptions
{
    PropertyNamingPolicy = JsonNamingPolicy.CamelCase,
    WriteIndented = true
};
jsonString = JsonSerializer.Serialize(weatherForecast, serializeOptions);
```

```
Dim serializeOptions As JsonSerializerOptions = New JsonSerializerOptions With {
    .PropertyNamingPolicy = JsonNamingPolicy.CamelCase,
    .WriteIndented = True
}
jsonString = JsonSerializer.Serialize(weatherForecast, serializeOptions)
```

Here's an example class to serialize and JSON output:

```
public class WeatherForecastWithPropertyNameAttribute
{
    public DateTimeOffset Date { get; set; }
    public int TemperatureCelsius { get; set; }
    public string? Summary { get; set; }
    [JsonPropertyName("Wind")]
    public int WindSpeed { get; set; }
}
```

```
Public Class WeatherForecastWithPropertyNameAttribute
    Public Property [Date] As DateTimeOffset
    Public Property TemperatureCelsius As Integer
    Public Property Summary As String

    <JsonPropertyName("Wind")>
    Public Property WindSpeed As Integer

End Class
```

```
{
    "date": "2019-08-01T00:00:00-07:00",
    "temperatureCelsius": 25,
    "summary": "Hot",
    "Wind": 35
}
```

The camel case property naming policy:

- Applies to serialization and deserialization.
- Is overridden by `[JsonPropertyName]` attributes. This is why the JSON property name `Wind` in the example is not camel case.

Use a custom JSON property naming policy

To use a custom JSON property naming policy, create a class that derives from `JsonNamingPolicy` and override the `ConvertName` method, as shown in the following example:

```
using System.Text.Json;

namespace SystemTextJsonSamples
{
    public class UpperCaseNamingPolicy : JsonNamingPolicy
    {
        public override string ConvertName(string name) =>
            name.ToUpper();
    }
}
```

```
Imports System.Text.Json

Namespace SystemTextJsonSamples

    Public Class UpperCaseNamingPolicy
        Inherits JsonNamingPolicy

        Public Overrides Function ConvertName(name As String) As String
            Return name.ToUpper()
        End Function

    End Class

End Namespace
```

Then set the `JsonSerializerOptions.PropertyNamingPolicy` property to an instance of your naming policy class:

```
var options = new JsonSerializerOptions
{
    PropertyNamingPolicy = new UpperCaseNamingPolicy(),
    WriteIndented = true
};
jsonString = JsonSerializer.Serialize(weatherForecast, options);
```

```

Dim options As JsonSerializerOptions = New JsonSerializerOptions With {
    .PropertyNamingPolicy = New UpperCaseNamingPolicy,
    .WriteIndented = True
}
jsonString = JsonSerializer.Serialize(weatherForecast1, options)

```

Here's an example class to serialize and JSON output:

```

public class WeatherForecastWithPropertyNameAttribute
{
    public DateTimeOffset Date { get; set; }
    public int TemperatureCelsius { get; set; }
    public string? Summary { get; set; }
    [JsonPropertyName("Wind")]
    public int WindSpeed { get; set; }
}

```

```

Public Class WeatherForecastWithPropertyNameAttribute
    Public Property [Date] As DateTimeOffset
    Public Property TemperatureCelsius As Integer
    Public Property Summary As String

    <JsonPropertyName("Wind")>
    Public Property WindSpeed As Integer

End Class

```

```
{
    "DATE": "2019-08-01T00:00:00-07:00",
    "TEMPERATURECELSIUS": 25,
    "SUMMARY": "Hot",
    "Wind": 35
}
```

The JSON property naming policy:

- Applies to serialization and deserialization.
- Is overridden by `[JsonPropertyName]` attributes. This is why the JSON property name `Wind` in the example is not upper case.

Camel case dictionary keys

If a property of an object to be serialized is of type `Dictionary<string,TValue>`, the `string` keys can be converted to camel case. To do that, set `DictionaryKeyPolicy` to `JsonNamingPolicy.CamelCase`, as shown in the following example:

```

var options = new JsonSerializerOptions
{
    DictionaryKeyPolicy = JsonNamingPolicy.CamelCase,
    WriteIndented = true
};
jsonString = JsonSerializer.Serialize(weatherForecast, options);

```

```

Dim options As JsonSerializerOptions = New JsonSerializerOptions With {
    .DictionaryKeyPolicy = JsonNamingPolicy.CamelCase,
    .WriteIndented = True
}
jsonString = JsonSerializer.Serialize(weatherForecast, options)

```

Serializing an object with a dictionary named `TemperatureRanges` that has key-value pairs `"ColdMinTemp", 20` and `"HotMinTemp", 40` would result in JSON output like the following example:

```
{
    "Date": "2019-08-01T00:00:00-07:00",
    "TemperatureCelsius": 25,
    "Summary": "Hot",
    "TemperatureRanges": {
        "coldMinTemp": 20,
        "hotMinTemp": 40
    }
}
```

The camel case naming policy for dictionary keys applies to serialization only. If you deserialize a dictionary, the keys will match the JSON file even if you specify `JsonNamingPolicy.CamelCase` for the `DictionaryKeyPolicy`.

Enums as strings

By default, enums are serialized as numbers. To serialize enum names as strings, use the [JsonStringEnumConverter](#).

For example, suppose you need to serialize the following class that has an enum:

```

public class WeatherForecastWithEnum
{
    public DateTimeOffset Date { get; set; }
    public int TemperatureCelsius { get; set; }
    public Summary? Summary { get; set; }
}

public enum Summary
{
    Cold, Cool, Warm, Hot
}

```

```

Public Class WeatherForecastWithEnum
    Public Property [Date] As DateTimeOffset
    Public Property TemperatureCelsius As Integer
    Public Property Summary As Summary
End Class

Public Enum Summary
    Cold
    Cool
    Warm
    Hot
End Enum

```

If the Summary is `Hot`, by default the serialized JSON has the numeric value 3:

```
{  
    "Date": "2019-08-01T00:00:00-07:00",  
    "TemperatureCelsius": 25,  
    "Summary": 3  
}
```

The following sample code serializes the enum names instead of the numeric values, and converts the names to camel case:

```
options = new JsonSerializerOptions  
{  
    WriteIndented = true,  
    Converters =  
    {  
        new JsonStringEnumConverter(JsonNamingPolicy.CamelCase)  
    }  
};  
jsonString = JsonSerializer.Serialize(weatherForecast, options);
```

```
options = New JsonSerializerOptions With {  
    .WriteIndented = True  
}  
options.Converters.Add(New JsonStringEnumConverter(JsonNamingPolicy.CamelCase))  
jsonString = JsonSerializer.Serialize(weatherForecast, options)
```

The resulting JSON looks like the following example:

```
{  
    "Date": "2019-08-01T00:00:00-07:00",  
    "TemperatureCelsius": 25,  
    "Summary": "hot"  
}
```

The built-in [JsonStringEnumConverter](#) can deserialize string values as well. It works without a specified naming policy or with the [CamelCase](#) naming policy. It doesn't support other naming policies, such as snake case. The following example shows deserialization using [CamelCase](#):

```
options = new JsonSerializerOptions  
{  
    Converters =  
    {  
        new JsonStringEnumConverter(JsonNamingPolicy.CamelCase)  
    }  
};  
weatherForecast = JsonSerializer.Deserialize<WeatherForecastWithEnum>(jsonString, options)!
```

```
options = New JsonSerializerOptions  
options.Converters.Add(New JsonStringEnumConverter(JsonNamingPolicy.CamelCase))  
weatherForecast = JsonSerializer.Deserialize(Of WeatherForecastWithEnum)(jsonString, options)
```

For information about custom converter code that supports deserialization while using a snake case naming policy, see [Support enum string value serialization](#).

Configure the order of serialized properties

The `[JsonPropertyOrder]` attribute lets you specify the order of properties in the JSON output from serialization.

The default value of the `Order` property is zero. Set `Order` to a positive number to position a property after those that have the default value. A negative `Order` positions a property before those that have the default value. Properties are written in order from the lowest `Order` value to the highest. Here's an example:

```
using System.Text.Json;
using System.Text.Json.Serialization;

namespace PropertyOrder
{
    public class WeatherForecast
    {
        [JsonPropertyOrder(-5)]
        public DateTime Date { get; set; }
        public int TemperatureC { get; set; }
        [JsonPropertyOrder(-2)]
        public int TemperatureF { get; set; }
        [JsonPropertyOrder(5)]
        public string? Summary { get; set; }
        [JsonPropertyOrder(2)]
        public int WindSpeed { get; set; }
    }

    public class Program
    {
        public static void Main()
        {
            var weatherForecast = new WeatherForecast
            {
                Date = DateTime.Parse("2019-08-01"),
                TemperatureC = 25,
                TemperatureF = 25,
                Summary = "Hot",
                WindSpeed = 10
            };

            var options = new JsonSerializerOptions { WriteIndented = true };
            string jsonString = JsonSerializer.Serialize(weatherForecast, options);
            Console.WriteLine(jsonString);
        }
    }
}

// output:
//{
//    "Date": "2019-08-01T00:00:00",
//    "TemperatureF": 25,
//    "TemperatureC": 25,
//    "WindSpeed": 10,
//    "Summary": "Hot"
//}
```

See also

- [System.Text.Json overview](#)
- [How to serialize and deserialize JSON](#)
- [Instantiate JsonSerializerOptions instances](#)
- [Enable case-insensitive matching](#)
- [Ignore properties](#)
- [Allow invalid JSON](#)
- [Handle overflow JSON or use JsonElement or JsonNode](#)
- [Preserve references and handle circular references](#)

- Deserialize to immutable types and non-public accessors
- Polymorphic serialization
- Migrate from Newtonsoft.Json to System.Text.Json
- Customize character encoding
- Use DOM, Utf8JsonReader, and Utf8JsonWriter
- Write custom converters for JSON serialization
- DateTime and DateTimeOffset support
- How to use source generation
- Supported collection types
- [System.Text.Json API reference](#)
- [System.Text.Json.Serialization API reference](#)

How to ignore properties with System.Text.Json

9/20/2022 • 6 minutes to read • [Edit Online](#)

When serializing C# objects to JavaScript Object Notation (JSON), by default, all public properties are serialized. If you don't want some of them to appear in the resulting JSON, you have several options. In this article you learn how to ignore properties based on various criteria:

- [Individual properties](#)
- [All read-only properties](#)
- [All null-value properties](#)
- [All default-value properties](#)
- [Individual properties](#)
- [All read-only properties](#)
- [All null-value properties](#)

Ignore individual properties

To ignore individual properties, use the `[JsonIgnore]` attribute.

Here's an example type to serialize and JSON output:

```
public class WeatherForecastWithIgnoreAttribute
{
    public DateTimeOffset Date { get; set; }
    public int TemperatureCelsius { get; set; }
    [JsonIgnore]
    public string? Summary { get; set; }
}
```

```
Public Class WeatherForecastWithIgnoreAttribute
    Public Property [Date] As DateTimeOffset
    Public Property TemperatureCelsius As Integer

    <JsonIgnore>
    Public Property Summary As String

End Class
```

```
{
    "Date": "2019-08-01T00:00:00-07:00",
    "TemperatureCelsius": 25,
}
```

You can specify conditional exclusion by setting the `[JsonIgnore]` attribute's `Condition` property. The `JsonIgnoreCondition` enum provides the following options:

- `Always` - The property is always ignored. If no `Condition` is specified, this option is assumed.
- `Never` - The property is always serialized and deserialized, regardless of the `DefaultIgnoreCondition`, `IgnoreReadOnlyProperties`, and `IgnoreReadOnlyFields` global settings.
- `WhenWritingDefault` - The property is ignored on serialization if it's a reference type `null`, a nullable value

type `null`, or a value type `default`.

- `WhenWritingNull` - The property is ignored on serialization if it's a reference type `null`, or a nullable value type `null`.

The following example illustrates use of the `[JsonIgnore]` attribute's `Condition` property:

```
using System.Text.Json;
using System.Text.Json.Serialization;

namespace JsonIgnoreAttributeExample
{
    public class Forecast
    {
        [JsonIgnore(Condition = JsonIgnoreCondition.WhenWritingDefault)]
        public DateTime Date { get; set; }

        [JsonIgnore(Condition = JsonIgnoreCondition.Never)]
        public int TemperatureC { get; set; }

        [JsonIgnore(Condition = JsonIgnoreCondition.WhenWritingNull)]
        public string? Summary { get; set; }
    };

    public class Program
    {
        public static void Main()
        {
            Forecast forecast = new()
            {
                Date = default,
                Summary = null,
                TemperatureC = default
            };

            JsonSerializerOptions options = new()
            {
                DefaultIgnoreCondition = JsonIgnoreCondition.WhenWritingDefault
            };

            string forecastJson =
                JsonSerializer.Serialize<Forecast>(forecast,options);

            Console.WriteLine(forecastJson);
        }
    }
}

// Produces output like the following example:
//>{"TemperatureC":0}
```

```

Imports System.Text.Json
Imports System.Text.Json.Serialization

Namespace JsonIgnoreAttributeExample

    Public Class Forecast

        <JsonIgnore(Condition:=JsonIgnoreCondition.WhenWritingDefault)>
        Public Property [Date] As Date

        <JsonIgnore(Condition:=JsonIgnoreCondition.Never)>
        Public Property TemperatureC As Integer

        <JsonIgnore(Condition:=JsonIgnoreCondition.WhenWritingNull)>
        Public Property Summary As String

    End Class

    Public NotInheritable Class Program

        Public Shared Sub Main()
            Dim forecast1 As New Forecast() With {
                .[Date] = CType(Nothing, Date),
                .Summary = Nothing,
                .TemperatureC = CType(Nothing, Integer)
            }

            Dim options As New JsonSerializerOptions() With {
                .DefaultIgnoreCondition = JsonIgnoreCondition.WhenWritingDefault
            }

            Dim forecastJson As String = JsonSerializer.Serialize(forecast1, options)

            Console.WriteLine(forecastJson)
        End Sub

    End Class

End Namespace

' Produces output like the following example:
'
'{"TemperatureC":0}

```

Ignore all read-only properties

A property is read-only if it contains a public getter but not a public setter. To ignore all read-only properties when serializing, set the `JsonSerializerOptions.IgnoreReadOnlyProperties` to `true`, as shown in the following example:

```

var options = new JsonSerializerOptions
{
    IgnoreReadOnlyProperties = true,
    WriteIndented = true
};
jsonString = JsonSerializer.Serialize(weatherForecast, options);

```

```

Dim options As JsonSerializerOptions = New JsonSerializerOptions With {
    .IgnoreReadOnlyProperties = True,
    .WriteIndented = True
}
jsonString = JsonSerializer.Serialize(weatherForecast, options)

```

Here's an example type to serialize and JSON output:

```

public class WeatherForecastWithROProperty
{
    public DateTimeOffset Date { get; set; }
    public int TemperatureCelsius { get; set; }
    public string? Summary { get; set; }
    public int WindSpeedReadOnly { get; private set; } = 35;
}

```

```

Public Class WeatherForecastWithROProperty
    Public Property [Date] As DateTimeOffset
    Public Property TemperatureCelsius As Integer
    Public Property Summary As String
    Private _windSpeedReadOnly As Integer

    Public Property WindSpeedReadOnly As Integer
        Get
            Return _windSpeedReadOnly
        End Get
        Private Set(Value As Integer)
            _windSpeedReadOnly = Value
        End Set
    End Property

End Class

```

```
{
    "Date": "2019-08-01T00:00:00-07:00",
    "TemperatureCelsius": 25,
    "Summary": "Hot",
}
```

This option applies only to serialization. During deserialization, read-only properties are ignored by default.

This option applies only to properties. To ignore read-only fields when [serializing fields](#), use the `JsonSerializerOptions.IgnoreReadOnlyFields` global setting.

Ignore all null-value properties

To ignore all null-value properties, set the `DefaultIgnoreCondition` property to `WhenWritingNull`, as shown in the following example:

```
using System.Text.Json;
using System.Text.Json.Serialization;

namespace IgnoreNullOnSerialize
{
    public class Forecast
    {
        public DateTime Date { get; set; }
        public int TemperatureC { get; set; }
        public string? Summary { get; set; }
    };

    public class Program
    {
        public static void Main()
        {
            Forecast forecast = new()
            {
                Date = DateTime.Now,
                Summary = null,
                TemperatureC = default
           };

            JsonSerializerOptions options = new()
            {
                DefaultIgnoreCondition = JsonIgnoreCondition.WhenWritingNull
            };

            string forecastJson =
                JsonSerializer.Serialize<Forecast>(forecast, options);

            Console.WriteLine(forecastJson);
        }
    }
}

// Produces output like the following example:
// [{"Date": "2020-10-30T10:11:40.2359135-07:00", "TemperatureC": 0}]
```

```

Imports System.Text.Json

Namespace IgnoreNullOnSerialize

    Public Class Forecast
        Public Property [Date] As Date
        Public Property TemperatureC As Integer
        Public Property Summary As String
    End Class

    Public NotInheritable Class Program

        Public Shared Sub Main()
            Dim forecast1 As New Forecast() With
            {
                .[Date] = Date.Now,
                .Summary = Nothing,
                .TemperatureC = CType(Nothing, Integer)
            }

            Dim options As New JsonSerializerOptions

            Dim forecastJson As String = JsonSerializer.Serialize(forecast1, options)

            Console.WriteLine(forecastJson)
        End Sub

    End Class

End Namespace

' Produces output like the following example:
'
'{"Date":"2020-10-30T10:11:40.2359135-07:00","TemperatureC":0}

```

To ignore all null-value properties when serializing or deserializing, set the `IgnoreNullValues` property to `true`. The following example shows this option used for serialization:

```

var options = new JsonSerializerOptions
{
    IgnoreNullValues = true,
    WriteIndented = true
};
jsonString = JsonSerializer.Serialize(weatherForecast, options);

```

```

Dim options As JsonSerializerOptions = New JsonSerializerOptions With {
    .IgnoreNullValues = True,
    .WriteIndented = True
}
jsonString = JsonSerializer.Serialize(weatherForecast1, options)

```

Here's an example object to serialize and JSON output:

PROPERTY	VALUE
Date	8/1/2019 12:00:00 AM -07:00
TemperatureCelsius	25

PROPERTY	VALUE
Summary	null
{ "Date": "2019-08-01T00:00:00-07:00", "TemperatureCelsius": 25 }	

NOTE

The [IgnoreNullValues](#) property is deprecated in .NET 5 and later versions. For the current way to ignore null values, see [how to ignore all null-value properties in .NET 5 and later](#).

Ignore all default-value properties

To prevent serialization of default values in value type properties, set the [DefaultIgnoreCondition](#) property to [WhenWritingDefault](#), as shown in the following example:

```
using System.Text.Json;
using System.Text.Json.Serialization;

namespace IgnoreValueDefaultOnSerialize
{
    public class Forecast
    {
        public DateTime Date { get; set; }
        public int TemperatureC { get; set; }
        public string? Summary { get; set; }
    };

    public class Program
    {
        public static void Main()
        {
            Forecast forecast = new()
            {
                Date = DateTime.Now,
                Summary = null,
                TemperatureC = default
           };

            JsonSerializerOptions options = new()
            {
                DefaultIgnoreCondition = JsonIgnoreCondition.WhenWritingDefault
            };

            string forecastJson =
                JsonSerializer.Serialize<Forecast>(forecast, options);

            Console.WriteLine(forecastJson);
        }
    }
}

// Produces output like the following example:
// {
//     "Date": "2020-10-21T15:40:06.8920138-07:00"
// }
```

```

Imports System.Text.Json
Imports System.Text.Json.Serialization

Namespace IgnoreValueDefaultOnSerialize

    Public Class Forecast
        Public Property [Date] As Date
        Public Property TemperatureC As Integer
        Public Property Summary As String
    End Class

    Public NotInheritable Class Program

        Public Shared Sub Main()
            Dim forecast1 As New Forecast() With
            {[.][Date] = Date.Now,
             .Summary = Nothing,
             .TemperatureC = CType(Nothing, Integer)}
        }

        Dim options As New JsonSerializerOptions() With {
            .DefaultIgnoreCondition = JsonIgnoreCondition.WhenWritingDefault
        }

        Dim forecastJson As String = JsonSerializer.Serialize(forecast1, options)

        Console.WriteLine(forecastJson)
    End Sub

    End Class
End Namespace

' Produces output like the following example:
'
'{ "Date":"2020-10-21T15:40:06.8920138-07:00"}
```

The [WhenWritingDefault](#) setting also prevents serialization of null-value reference type and nullable value type properties.

There is no built-in way to prevent serialization of properties with value type defaults in `System.Text.Json` in .NET Core 3.1.

See also

- [System.Text.Json overview](#)
- [How to serialize and deserialize JSON](#)
- [Instantiate JsonSerializerOptions instances](#)
- [Enable case-insensitive matching](#)
- [Customize property names and values](#)
- [Allow invalid JSON](#)
- [Handle overflow JSON or use JsonElement or JsonNode](#)
- [Preserve references and handle circular references](#)
- [Deserialize to immutable types and non-public accessors](#)
- [Polymorphic serialization](#)
- [Migrate from Newtonsoft.Json to System.Text.Json](#)
- [Customize character encoding](#)
- [Use DOM, Utf8JsonReader, and Utf8JsonWriter](#)

- [Write custom converters for JSON serialization](#)
- [DateTime and DateTimeOffset support](#)
- [How to use source generation](#)
- [Supported collection types](#)
- [System.Text.Json API reference](#)
- [System.Text.Json.Serialization API reference](#)

How to allow some kinds of invalid JSON with System.Text.Json

9/20/2022 • 2 minutes to read • [Edit Online](#)

In this article, you will learn how to allow comments, trailing commas, and quoted numbers in JSON, and how to write numbers as strings.

Allow comments and trailing commas

By default, comments and trailing commas are not allowed in JSON. To allow comments in the JSON, set the `JsonSerializerOptions.ReadCommentHandling` property to `JsonCommentHandling.Skip`. And to allow trailing commas, set the `JsonSerializerOptions.AllowTrailingCommas` property to `true`. The following example shows how to allow both:

```
var options = new JsonSerializerOptions
{
    ReadCommentHandling = JsonCommentHandling.Skip,
    AllowTrailingCommas = true,
};
var weatherForecast = JsonSerializer.Deserialize<WeatherForecast>(jsonString, options);
```

```
Dim options As JsonSerializerOptions = New JsonSerializerOptions With {
    .ReadCommentHandling = JsonCommentHandling.Skip,
    .AllowTrailingCommas = True
}
Dim weatherForecast1 = JsonSerializer.Deserialize(Of WeatherForecast)(jsonString, options)
```

Here's example JSON with comments and a trailing comma:

```
{
    "Date": "2019-08-01T00:00:00-07:00",
    "TemperatureCelsius": 25, // Fahrenheit 77
    "Summary": "Hot", /* Zharko */
    // Comments on
    /* separate lines */
}
```

Allow or write numbers in quotes

Some serializers encode numbers as JSON strings (surrounded by quotes).

For example:

```
{
    "DegreesCelsius": "23"
}
```

Instead of:

```
{  
    "DegreesCelsius": 23  
}
```

To serialize numbers in quotes or accept numbers in quotes across the entire input object graph, set [JsonSerializerOptions.NumberHandling](#) as shown in the following example:

```
using System.Text.Json;  
using System.Text.Json.Serialization;  
  
namespace QuotedNumbers  
{  
    public class Forecast  
    {  
        public DateTime Date { get; init; }  
        public int TemperatureC { get; set; }  
        public string? Summary { get; set; }  
    };  
  
    public class Program  
    {  
        public static void Main()  
        {  
            Forecast forecast = new()  
            {  
                Date = DateTime.Now,  
                TemperatureC = 40,  
                Summary = "Hot"  
           };  
  
            JsonSerializerOptions options = new()  
            {  
                NumberHandling =  
                    JsonNumberHandling.AllowReadingFromString |  
                    JsonNumberHandling.WriteString,  
                WriteIndented = true  
           };  
  
            string forecastJson =  
                JsonSerializer.Serialize<Forecast>(forecast, options);  
  
            Console.WriteLine($"Output JSON:\n{forecastJson}");  
  
            Forecast forecastDeserialized =  
                JsonSerializer.Deserialize<Forecast>(forecastJson, options)!;  
  
            Console.WriteLine($"Date: {forecastDeserialized.Date}");  
            Console.WriteLine($"TemperatureC: {forecastDeserialized.TemperatureC}");  
            Console.WriteLine($"Summary: {forecastDeserialized.Summary}");  
        }  
    }  
}  
  
// Produces output like the following example:  
//  
//Output JSON:  
//{  
//    "Date": "2020-10-23T12:27:06.4017385-07:00",  
//    "TemperatureC": "40",  
//    "Summary": "Hot"  
//}  
//Date: 10/23/2020 12:27:06 PM  
//TemperatureC: 40  
//Summary: Hot
```

```

Imports System.Text.Json
Imports System.Text.Json.Serialization

Namespace QuotedNumbers

    Public Class Forecast
        Public Property [Date] As Date
        Public Property TemperatureC As Integer
        Public Property Summary As String
    End Class

    Public NotInheritable Class Program

        Public Shared Sub Main()
            Dim forecast1 As New Forecast() With {
                .[Date] = Date.Now,
                .TemperatureC = 40,
                .Summary = "Hot"
            }

            Dim options As New JsonSerializerOptions() With {
                .NumberHandling = JsonNumberHandling.AllowReadingFromString Or
                    JsonNumberHandling.WriteAsString,
                .WriteIndented = True
            }

            Dim forecastJson As String = JsonSerializer.Serialize(forecast1, options)

            Console.WriteLine($"Output JSON:{forecastJson}")

            Dim forecastDeserialized As Forecast = JsonSerializer.Deserialize(Of Forecast)(forecastJson,
options)

            Console.WriteLine($"Date: {forecastDeserialized.[Date]}")
            Console.WriteLine($"TemperatureC: {forecastDeserialized.TemperatureC}")
            Console.WriteLine($"Summary: {forecastDeserialized.Summary}")
        End Sub

    End Class
End Namespace

' Produces output like the following example:
'
'Output JSON:
'{{
'    "Date": "2020-10-23T12:27:06.4017385-07:00",
'    "TemperatureC": "40",
'    "Summary": "Hot"
'}}
'Date: 10/23/2020 12:27:06 PM
'TemperatureC: 40
'Summary: Hot

```

When you use `System.Text.Json` indirectly through ASP.NET Core, quoted numbers are allowed when deserializing because ASP.NET Core specifies [web default options](#).

To allow or write quoted numbers for specific properties, fields, or types, use the [\[JsonNumberHandling\]](#) attribute.

`System.Text.Json` in .NET Core 3.1 doesn't support serializing or deserializing numbers surrounded by quotation marks. For more information, see [Allow or write numbers in quotes](#).

See also

- [System.Text.Json overview](#)
- [How to serialize and deserialize JSON](#)
- [Instantiate JsonSerializerOptions instances](#)
- [Enable case-insensitive matching](#)
- [Customize property names and values](#)
- [Ignore properties](#)
- [Handle overflow JSON or use JsonElement or JsonNode](#)
- [Preserve references and handle circular references](#)
- [Deserialize to immutable types and non-public accessors](#)
- [Polymorphic serialization](#)
- [Migrate from Newtonsoft.Json to System.Text.Json](#)
- [Customize character encoding](#)
- [Use DOM, Utf8JsonReader, and Utf8JsonWriter](#)
- [Write custom converters for JSON serialization](#)
- [DateTime and DateTimeOffset support](#)
- [How to use source generation](#)
- [Supported collection types](#)
- [System.Text.Json API reference](#)
- [System.Text.Json.Serialization API reference](#)

How to handle overflow JSON or use JsonElement or JsonNode in System.Text.Json

9/20/2022 • 5 minutes to read • [Edit Online](#)

This article shows how to handle overflow JSON with the `System.Text.Json` namespace. It also shows how to deserialize into `JsonElement` or `JsonNode`, as an alternative for other scenarios where the target type might not perfectly match all of the JSON being deserialized.

Handle overflow JSON

While deserializing, you might receive data in the JSON that is not represented by properties of the target type. For example, suppose your target type is this:

```
public class WeatherForecast
{
    public DateTimeOffset Date { get; set; }
    public int TemperatureCelsius { get; set; }
    public string? Summary { get; set; }
}
```

```
Public Class WeatherForecast
    Public Property [Date] As DateTimeOffset
    Public Property TemperatureCelsius As Integer
    Public Property Summary As String
End Class
```

And the JSON to be deserialized is this:

```
{
    "Date": "2019-08-01T00:00:00-07:00",
    "temperatureCelsius": 25,
    "Summary": "Hot",
    "DatesAvailable": [
        "2019-08-01T00:00:00-07:00",
        "2019-08-02T00:00:00-07:00"
    ],
    "SummaryWords": [
        "Cool",
        "Windy",
        "Humid"
    ]
}
```

If you deserialize the JSON shown into the type shown, the `DatesAvailable` and `SummaryWords` properties have nowhere to go and are lost. To capture extra data such as these properties, apply the `[JsonExtensionData]` attribute to a property of type `Dictionary<string,object>` or `Dictionary<string,JsonElement>`:

```

public class WeatherForecastWithExtensionData
{
    public DateTimeOffset Date { get; set; }
    public int TemperatureCelsius { get; set; }
    public string? Summary { get; set; }
    [JsonExtensionData]
    public Dictionary<string, JsonElement>? ExtensionData { get; set; }
}

```

```

Public Class WeatherForecastWithExtensionData
    Public Property [Date] As DateTimeOffset
    Public Property TemperatureCelsius As Integer
    Public Property Summary As String

    <JsonExtensionData>
    Public Property ExtensionData As Dictionary(Of String, Object)

End Class

```

The following table shows the result of deserializing the JSON shown earlier into this sample type. The extra data becomes key-value pairs of the `ExtensionData` property:

PROPERTY	VALUE	NOTES
<code>Date</code>	"8/1/2019 12:00:00 AM -07:00"	
<code>TemperatureCelsius</code>	0	Case-sensitive mismatch (<code>temperatureCelsius</code> in the JSON), so the property isn't set.
<code>Summary</code>	"Hot"	
<code>ExtensionData</code>	<pre> "temperatureCelsius": 25, "DatesAvailable": ["2019-08-01T00:00:00-07:00", "2019-08-02T00:00:00-07:00"], "SummaryWords": ["Cool", "Windy", "Humid"] </pre>	<p>Since the case didn't match, <code>temperatureCelsius</code> is an extra and becomes a key-value pair in the dictionary.</p> <p>Each extra array from the JSON becomes a key-value pair, with an array as the value object.</p>

When the target object is serialized, the extension data key value pairs become JSON properties just as they were in the incoming JSON:

```

{
    "Date": "2019-08-01T00:00:00-07:00",
    "TemperatureCelsius": 0,
    "Summary": "Hot",
    "temperatureCelsius": 25,
    "DatesAvailable": [
        "2019-08-01T00:00:00-07:00",
        "2019-08-02T00:00:00-07:00"
    ],
    "SummaryWords": [
        "Cool",
        "Windy",
        "Humid"
    ]
}

```

Notice that the `[ExtensionData]` property name doesn't appear in the JSON. This behavior lets the JSON make a round trip without losing any extra data that otherwise wouldn't be serialized.

The following example shows a round trip from JSON to a deserialized object and back to JSON:

```
using System.Text.Json;
using System.Text.Json.Serialization;

namespace RoundtripExtensionData
{
    public class WeatherForecast
    {
        public DateTimeOffset Date { get; set; }
        public int TemperatureCelsius { get; set; }
        public string? Summary { get; set; }
        [JsonExtensionData]
        public Dictionary<string, JsonElement>? ExtensionData { get; set; }
    }

    public class Program
    {
        public static void Main()
        {
            string jsonString =
@"
{
    ""Date"": ""2019-08-01T00:00:00-07:00"",
    ""temperatureCelsius"": 25,
    ""Summary"": ""Hot"",
    ""SummaryField"": ""Hot"",
    ""DatesAvailable"": [
        ""2019-08-01T00:00:00-07:00"",
        ""2019-08-02T00:00:00-07:00""
    ],
    ""SummaryWords"": [
        ""Cool"",
        ""Windy"",
        ""Humid""
    ]
}";
            WeatherForecast weatherForecast =
                JsonSerializer.Deserialize<WeatherForecast>(jsonString);

            var serializeOptions = new JsonSerializerOptions { WriteIndented = true };
            jsonString = JsonSerializer.Serialize(weatherForecast, serializeOptions);
            Console.WriteLine($"JSON output:\n{jsonString}\n");
        }
    }
}

// output:
//JSON output:
//{
//    "Date": "2019-08-01T00:00:00-07:00",
//    "TemperatureCelsius": 0,
//    "Summary": "Hot",
//    "temperatureCelsius": 25,
//    "SummaryField": "Hot",
//    "DatesAvailable": [
//        "2019-08-01T00:00:00-07:00",
//        "2019-08-02T00:00:00-07:00"
//    ],
//    "SummaryWords": [
//        "Cool",
//        "Windy",
//        "Humid"
//    ]
//}
```

Deserialize into JsonElement or JsonNode

If you just want to be flexible about what JSON to accept for a particular property, an alternative is to deserialize into [JsonElement](#) or [JsonNode](#). Any valid JSON property can be deserialized into [JsonElement](#) or [JsonNode](#).

Choose [JsonElement](#) to create an immutable object or [JsonNode](#) to create a mutable object.

The following example shows a round trip from JSON and back to JSON for a class that includes properties of type [JsonElement](#) and [JsonNode](#).

```
using System.Text.Json;
using System.Text.Json.Nodes;

namespace RoundtripJsonElementAndNode
{
    public class WeatherForecast
    {
        public DateTimeOffset Date { get; set; }
        public int TemperatureCelsius { get; set; }
        public string? Summary { get; set; }
        public JsonElement DatesAvailable { get; set; }
        public JsonNode? SummaryWords { get; set; }
    }

    public class Program
    {
        public static void Main()
        {
            string jsonString =
@"
{
    "Date": "2019-08-01T00:00:00-07:00",
    "TemperatureCelsius": 25,
    "Summary": "Hot",
    "DatesAvailable": [
        "2019-08-01T00:00:00-07:00",
        "2019-08-02T00:00:00-07:00"
    ],
    "SummaryWords": [
        "Cool",
        "Windy",
        "Humid"
    ]
}";
            WeatherForecast? weatherForecast =
                JsonSerializer.Deserialize<WeatherForecast>(jsonString);

            var serializeOptions = new JsonSerializerOptions { WriteIndented = true };
            jsonString = JsonSerializer.Serialize(weatherForecast, serializeOptions);
            Console.WriteLine(jsonString);
        }
    }
}

// output:
//{
//    "Date": "2019-08-01T00:00:00-07:00",
//    "TemperatureCelsius": 25,
//    "Summary": "Hot",
//    "DatesAvailable": [
//        "2019-08-01T00:00:00-07:00",
//        "2019-08-02T00:00:00-07:00"
//    ],
//    "SummaryWords": [
//        "Cool",
//        "Windy",
//        "Humid"
//    ]
//}
```

If you just want to be flexible about what JSON to accept for a particular property, an alternative is to deserialize into [JsonElement](#). Any valid JSON property can be deserialized into [JsonElement](#). [JsonNode](#) is not supported in .NET 5 and earlier versions.

The following example shows a round trip from JSON and back to JSON for a class that includes properties of type `JsonElement`.

```
using System;
using System.Text.Json;

namespace RoundtripJsonElement
{
    public class WeatherForecast
    {
        public DateTimeOffset Date { get; set; }
        public int TemperatureCelsius { get; set; }
        public string? Summary { get; set; }
        public JsonElement DatesAvailable { get; set; }
        public JsonElement SummaryWords { get; set; }
    }

    public class Program
    {
        public static void Main()
        {
            string jsonString =
@"
{
    ""Date"": ""2019-08-01T00:00:00-07:00"",
    ""TemperatureCelsius"": 25,
    ""Summary"": ""Hot"",
    ""DatesAvailable"": [
        ""2019-08-01T00:00:00-07:00"",
        ""2019-08-02T00:00:00-07:00""
    ],
    ""SummaryWords"": [
        ""Cool"",
        ""Windy"",
        ""Humid""
    ]
}";
            WeatherForecast weatherForecast =
                JsonSerializer.Deserialize<WeatherForecast>(jsonString)!;

            var serializeOptions = new JsonSerializerOptions { WriteIndented = true };
            jsonString = JsonSerializer.Serialize(weatherForecast, serializeOptions);
            Console.WriteLine(jsonString);
        }
    }
}

// output:
//{
//    "Date": "2019-08-01T00:00:00-07:00",
//    "TemperatureCelsius": 25,
//    "Summary": "Hot",
//    "DatesAvailable": [
//        "2019-08-01T00:00:00-07:00",
//        "2019-08-02T00:00:00-07:00"
//    ],
//    "SummaryWords": [
//        "Cool",
//        "Windy",
//        "Humid"
//    ]
//}
```

See also

- [System.Text.Json overview](#)
- [How to serialize and deserialize JSON](#)
- [Instantiate JsonSerializerOptions instances](#)
- [Enable case-insensitive matching](#)
- [Customize property names and values](#)
- [Ignore properties](#)
- [Allow invalid JSON](#)
- [Preserve references and handle circular references](#)
- [Deserialize to immutable types and non-public accessors](#)
- [Polymorphic serialization](#)
- [Migrate from Newtonsoft.Json to System.Text.Json](#)
- [Customize character encoding](#)
- [Use DOM, Utf8JsonReader, and Utf8JsonWriter](#)
- [Write custom converters for JSON serialization](#)
- [DateTime and DateTimeOffset support](#)
- [How to use source generation](#)
- [Supported collection types](#)
- [System.Text.Json API reference](#)
- [System.Text.Json.Serialization API reference](#)

How to preserve references and handle or ignore circular references in System.Text.Json

9/20/2022 • 6 minutes to read • [Edit Online](#)

This article shows how to preserve references and handle or ignore circular references while using `System.Text.Json` to serialize and deserialize JSON in .NET

Preserve references and handle circular references

To preserve references and handle circular references, set `ReferenceHandler` to `Preserve`. This setting causes the following behavior:

- On serialize:

When writing complex types, the serializer also writes metadata properties (`$id`, `$values`, and `$ref`).

- On deserialize:

Metadata is expected (although not mandatory), and the deserializer tries to understand it.

The following code illustrates use of the `Preserve` setting.

```
using System.Text.Json;
using System.Text.Json.Serialization;

namespace PreserveReferences
{
    public class Employee
    {
        public string? Name { get; set; }
        public Employee? Manager { get; set; }
        public List<Employee>? DirectReports { get; set; }
    }

    public class Program
    {
        public static void Main()
        {
            Employee tyler = new()
            {
                Name = "Tyler Stein"
            };

            Employee adrian = new()
            {
                Name = "Adrian King"
            };

            tyler.DirectReports = new List<Employee> { adrian };
            adrian.Manager = tyler;

            JsonSerializerOptions options = new()
            {
                ReferenceHandler = ReferenceHandler.Preserve,
                WriteIndented = true
            };

            string tylerJson = JsonSerializer.Serialize(tyler, options);
            Console.WriteLine($"Tyler serialized:\n{tylerJson}");
        }
    }
}
```

```
Employee? tylerDeserialized =
    JsonSerializer.Deserialize<Employee>(tylerJson, options);

Console.WriteLine(
    "Tyler is manager of Tyler's first direct report: ");
Console.WriteLine(
    tylerDeserialized?.DirectReports?[0].Manager == tylerDeserialized);
}

}

// Produces output like the following example:
//  

//Tyler serialized:  

//{  

//  "$id": "1",  

//  "Name": "Tyler Stein",  

//  "Manager": null,  

//  "DirectReports": {  

//    "$id": "2",  

//    "$values": [  

//      {  

//        "$id": "3",  

//        "Name": "Adrian King",  

//        "Manager": {  

//          "$ref": "1"  

//        },  

//        "DirectReports": null  

//      }  

//    ]  

//  }  

//}  

//Tyler is manager of Tyler's first direct report:  

//True
```

```

Imports System.Text.Json
Imports System.Text.Json.Serialization

Namespace PreserveReferences

    Public Class Employee
        Public Property Name As String
        Public Property Manager As Employee
        Public Property DirectReports As List(Of Employee)
    End Class

    Public NotInheritable Class Program

        Public Shared Sub Main()
            Dim tyler As New Employee

            Dim adrian As New Employee

            tyler.DirectReports = New List(Of Employee) From {
                adrian}
            adrian.Manager = tyler

            Dim options As New JsonSerializerOptions With {
                .ReferenceHandler = ReferenceHandler.Preserve,
                .WriteIndented = True
            }

            Dim tylerJson As String = JsonSerializer.Serialize(tyler, options)
            Console.WriteLine($"Tyler serialized:{tylerJson}")

            Dim tylerDeserialized As Employee = JsonSerializer.Deserialize(Of Employee)(tylerJson, options)

            Console.WriteLine(
                "Tyler is manager of Tyler's first direct report: ")
            Console.WriteLine(
                tylerDeserialized.DirectReports(0).Manager Is tylerDeserialized)
        End Sub

    End Class

End Namespace

' Produces output like the following example:
'
'Tyler serialized:
'{ 
'  "$id": "1",
'  "Name": "Tyler Stein",
'  "Manager": null,
'  "DirectReports": {
'    "$id": "2",
'    "$values": [
'      {
'        "$id": "3",
'        "Name": "Adrian King",
'        "Manager": {
'          "$ref": "1"
'        },
'        "DirectReports": null
'      }
'    ]
'  }
'}

'Tyler is manager of Tyler's first direct report:
'True

```

This feature can't be used to preserve value types or immutable types. On deserialization, the instance of an immutable type is created after the entire payload is read. So it would be impossible to deserialize the same instance if a reference to it appears within the JSON payload.

For value types, immutable types, and arrays, no reference metadata is serialized. On deserialization, an exception is thrown if `$ref` or `$id` is found. However, value types ignore `$id` (and `$values` in the case of collections) to make it possible to deserialize payloads that were serialized by using Newtonsoft.Json. Newtonsoft.Json does serialize metadata for such types.

To determine if objects are equal, `System.Text.Json` uses `ReferenceEqualityComparer.Instance`, which uses reference equality (`Object.ReferenceEquals(Object, Object)`) instead of value equality (`Object.Equals(Object)`) when comparing two object instances.

For more information about how references are serialized and deserialized, see [ReferenceHandler.Preserve](#).

The `ReferenceResolver` class defines the behavior of preserving references on serialization and deserialization. Create a derived class to specify custom behavior. For an example, see [GuidReferenceResolver](#).

Persist reference metadata across multiple serialization and deserialization calls

By default, reference data is only cached for each call to `Serialize` or `Deserialize`. To persist references from one `Serialize` / `Deserialize` call to another one, root the `ReferenceResolver` instance in the call site of `Serialize` / `Deserialize`. The following code shows an example for this scenario:

- You have a list of `Employee`s and you have to serialize each one individually.
- you want to take advantage of the references saved in the `ReferenceHandler`'s resolver.

Here is the `Employee` class:

```
public class Employee
{
    public string? Name { get; set; }
    public Employee? Manager { get; set; }
    public List<Employee>? DirectReports { get; set; }
}
```

A class that derives from `ReferenceResolver` stores the references in a dictionary:

```

class MyReferenceResolver : ReferenceResolver
{
    private uint _referenceCount;
    private readonly Dictionary<string, object> _referenceIdToObjectMap = new ();
    private readonly Dictionary<object, string> _objectToReferenceIdMap = new
(ReferenceEqualityComparer.Instance);

    public override void AddReference(string referenceId, object value)
    {
        if (!_referenceIdToObjectMap.TryAdd(referenceId, value))
        {
            throw new JsonException();
        }
    }

    public override string GetReference(object value, out bool alreadyExists)
    {
        if (_objectToReferenceIdMap.TryGetValue(value, out string? referenceId))
        {
            alreadyExists = true;
        }
        else
        {
            _referenceCount++;
            referenceId = _referenceCount.ToString();
            _objectToReferenceIdMap.Add(value, referenceId);
            alreadyExists = false;
        }

        return referenceId;
    }

    public override object ResolveReference(string referenceId)
    {
        if (!_referenceIdToObjectMap.TryGetValue(referenceId, out object? value))
        {
            throw new JsonException();
        }

        return value;
    }
}

```

A class that derives from [ReferenceHandler](#) holds an instance of `MyReferenceResolver` and creates a new instance only when needed (in a method named `Reset` in this example):

```

class MyReferenceHandler : ReferenceHandler
{
    public MyReferenceHandler() => Reset();
    private ReferenceResolver? _rootedResolver;
    public override ReferenceResolver CreateResolver() => _rootedResolver!;
    public void Reset() => _rootedResolver = new MyReferenceResolver();
}

```

When the sample code calls the serializer, it uses a [JsonSerializerOptions](#) instance in which the [ReferenceHandler](#) property is set to an instance of `MyReferenceHandler`. When you follow this pattern, be sure to reset the `ReferenceResolver` dictionary when you're finished serializing, to keep it from growing forever.

```
var options = new JsonSerializerOptions();
options.WriteIndented = true;
var myReferenceHandler = new MyReferenceHandler();
options.ReferenceHandler = myReferenceHandler;

string json;
foreach (Employee emp in employees)
{
    json = JsonSerializer.Serialize(emp, options);
    DoSomething(json);
}

// Reset after serializing to avoid out of bounds memory growth in the resolver.
myReferenceHandler.Reset();
```

System.Text.Json in .NET Core 3.1 only supports serialization by value and throws an exception for circular references.

Ignore circular references

Instead of handling circular references, you can ignore them. To ignore circular references, set [ReferenceHandler](#) to [IgnoreCycles](#). The serializer sets circular reference properties to `null`, as shown in the following example:

```

using System.Text.Json;
using System.Text.Json.Serialization;

namespace SerializeIgnoreCycles
{
    public class Employee
    {
        public string? Name { get; set; }
        public Employee? Manager { get; set; }
        public List<Employee>? DirectReports { get; set; }
    }

    public class Program
    {
        public static void Main()
        {
            Employee tyler = new()
            {
                Name = "Tyler Stein"
            };

            Employee adrian = new()
            {
                Name = "Adrian King"
            };

            tyler.DirectReports = new List<Employee> { adrian };
            adrian.Manager = tyler;

            JsonSerializerOptions options = new()
            {
                ReferenceHandler = ReferenceHandler.IgnoreCycles,
                WriteIndented = true
            };

            string tylerJson = JsonSerializer.Serialize(tyler, options);
            Console.WriteLine($"Tyler serialized:\n{tylerJson}");

            Employee? tylerDeserialized =
                JsonSerializer.Deserialize<Employee>(tylerJson, options);

            Console.WriteLine(
                "Tyler is manager of Tyler's first direct report: ");
            Console.WriteLine(
                tylerDeserialized?.DirectReports?[0]?.Manager == tylerDeserialized);
        }
    }
}

// Produces output like the following example:
// 
//Tyler serialized:
//{
//  "Name": "Tyler Stein",
//  "Manager": null,
//  "DirectReports": [
//    {
//      "Name": "Adrian King",
//      "Manager": null,
//      "DirectReports": null
//    }
//  ]
//}
//Tyler is manager of Tyler's first direct report:
//False

```

In the preceding example, `Manager` under `Adrian King` is serialized as `null` to avoid the circular reference. This behavior has the following advantages over [ReferenceHandler.Preserve](#):

- It decreases payload size.
- It creates JSON that is comprehensible for serializers other than `System.Text.Json` and `Newtonsoft.Json`.

This behavior has the following disadvantages:

- Silent loss of data.
- Data can't make a round trip from JSON back to the source object.

`System.Text.Json` in .NET 5 and earlier doesn't support [ReferenceHandler.IgnoreCycles](#).

See also

- [System.Text.Json overview](#)
- [How to serialize and deserialize JSON](#)
- [Instantiate JsonSerializerOptions instances](#)
- [Enable case-insensitive matching](#)
- [Customize property names and values](#)
- [Ignore properties](#)
- [Allow invalid JSON](#)
- [Handle overflow JSON or use JsonElement or JsonNode](#)
- [Deserialize to immutable types and non-public accessors](#)
- [Polymorphic serialization](#)
- [Migrate from Newtonsoft.Json to System.Text.Json](#)
- [Customize character encoding](#)
- [Use DOM, Utf8JsonReader, and Utf8JsonWriter](#)
- [Write custom converters for JSON serialization](#)
- [DateTime and DateTimeOffset support](#)
- [How to use source generation](#)
- [Supported collection types](#)
- [System.Text.Json API reference](#)
- [System.Text.Json.Serialization API reference](#)

How to use immutable types and non-public accessors with System.Text.Json

9/20/2022 • 5 minutes to read • [Edit Online](#)

This article shows how to use immutable types, public parameterized constructors, and non-public accessors with the `System.Text.Json` namespace.

Immutable types and Records

`System.Text.Json` can use a public parameterized constructor, which makes it possible to deserialize an immutable class or struct. For a class, if the only constructor is a parameterized one, that constructor will be used. For a struct, or a class with multiple constructors, specify the one to use by applying the `[JsonConstructor]` attribute. When the attribute is not used, a public parameterless constructor is always used if present. The attribute can only be used with public constructors. The following example uses the `[JsonConstructor]` attribute:

```
using System.Text.Json;
using System.Text.Json.Serialization;

namespace ImmutableTypes
{
    public struct Forecast
    {
        public DateTime Date { get; }
        public int TemperatureC { get; }
        public string Summary { get; }

        [JsonConstructor]
        public Forecast(DateTime date, int temperatureC, string summary) =>
            (Date, TemperatureC, Summary) = (date, temperatureC, summary);
    }

    public class Program
    {
        public static void Main()
        {
            var json = @"{""date"":""2020-09-06T11:31:01.923395-07:00"",""temperatureC"": -1,""summary"":""Cold""} ";
            Console.WriteLine($"Input JSON: {json}");

            var options = new JsonSerializerOptions(JsonSerializerDefaults.Web);

            var forecast = JsonSerializer.Deserialize<Forecast>(json, options);

            Console.WriteLine($"forecast.Date: {forecast.Date}");
            Console.WriteLine($"forecast.TemperatureC: {forecast.TemperatureC}");
            Console.WriteLine($"forecast.Summary: {forecast.Summary}");

            var roundTrippedJson =
                JsonSerializer.Serialize<Forecast>(forecast, options);

            Console.WriteLine($"Output JSON: {roundTrippedJson}");
        }
    }
}

// Produces output like the following example:
//
//Input JSON: { "date":"2020-09-06T11:31:01.923395-07:00","temperatureC": -1,"summary": "Cold"}
//forecast.Date: 9 / 6 / 2020 11:31:01 AM
//forecast.TemperatureC: -1
//forecast.Summary: Cold
//Output JSON: { "date":"2020-09-06T11:31:01.923395-07:00","temperatureC": -1,"summary": "Cold"}
```

```

Imports System.Text.Json
Imports System.Text.Json.Serialization

Namespace ImmutableTypes

    Public Structure Forecast
        Public ReadOnly Property [Date] As Date
        Public ReadOnly Property TemperatureC As Integer
        Public ReadOnly Property Summary As String

        <>JsonConstructor>
        Public Sub New([Date] As Date, TemperatureC As Integer, Summary As String)
            Me.Date = [Date]
            Me.TemperatureC = TemperatureC
            Me.Summary = Summary
        End Sub

    End Structure

    Public NotInheritable Class Program

        Public Shared Sub Main()
            Dim json As String = "{\"date\":\"2020-09-06T11:31:01.923395-07:00\",\"temperatureC\":-1,\"summary\":\"Cold\"}"
            Console.WriteLine($"Input JSON: {json}")

            Dim options As New JsonSerializerOptions(JsonSerializerDefaults.Web)

            Dim forecast1 As Forecast = JsonSerializer.Deserialize(Of Forecast)(json, options)

            Console.WriteLine($"forecast.Date: {forecast1.[Date]}")
            Console.WriteLine($"forecast.TemperatureC: {forecast1.TemperatureC}")
            Console.WriteLine($"forecast.Summary: {forecast1.Summary}")

            Dim roundTrippedJson As String = JsonSerializer.Serialize(forecast1, options)

            Console.WriteLine($"Output JSON: {roundTrippedJson}")
        End Sub

    End Class

End Namespace

' Produces output like the following example:
'
'Input JSON: { "date":"2020-09-06T11:31:01.923395-07:00","temperatureC":-1,"summary":"Cold"}
'forecast.Date: 9 / 6 / 2020 11:31:01 AM
'forecast.TemperatureC: -1
'forecast.Summary: Cold
'Output JSON: { "date":"2020-09-06T11:31:01.923395-07:00","temperatureC":-1,"summary":"Cold"}
```

The parameter names of a parameterized constructor must match the property names. Matching is case-insensitive, and the constructor parameter must match the actual property name even if you use [\[JsonPropertyName\]](#) to rename a property. In the following example, the name for the `TemperatureC` property is changed to `celsius` in the JSON, but the constructor parameter is still named `temperatureC`:

```

using System.Text.Json;
using System.Text.Json.Serialization;

namespace ImmutableTypesCtorParams
{
    public struct Forecast
    {
        public DateTime Date { get; }
        [JsonPropertyName("celsius")]
        public int TemperatureC { get; }
        public string Summary { get; }

        [JsonConstructor]
        public Forecast(DateTime date, int temperatureC, string summary) =>
            (Date, TemperatureC, Summary) = (date, temperatureC, summary);
    }

    public class Program
    {
        public static void Main()
        {
            var json = @"{""date"":""2020-09-06T11:31:01.923395-07:00"",""celsius"":-1,""summary"":""Cold""}";
            Console.WriteLine($"Input JSON: {json}");

            var options = new JsonSerializerOptions(JsonSerializerDefaults.Web);

            var forecast = JsonSerializer.Deserialize<Forecast>(json, options);

            Console.WriteLine($"forecast.Date: {forecast.Date}");
            Console.WriteLine($"forecast.TemperatureC: {forecast.TemperatureC}");
            Console.WriteLine($"forecast.Summary: {forecast.Summary}");

            var roundTrippedJson =
                JsonSerializer.Serialize<Forecast>(forecast, options);

            Console.WriteLine($"Output JSON: {roundTrippedJson}");
        }
    }
}

// Produces output like the following example:
// 
//Input JSON: { "date":"2020-09-06T11:31:01.923395-07:00","celsius":-1,"summary":"Cold"}
//forecast.Date: 9 / 6 / 2020 11:31:01 AM
//forecast.TemperatureC: -1
//forecast.Summary: Cold
//Output JSON: { "date":"2020-09-06T11:31:01.923395-07:00","celsius":-1,"summary":"Cold"}
```

Besides `[JsonPropertyName]` the following attributes support deserialization with parameterized constructors:

- `[JsonConverter]`
- `[JsonIgnore]`
- `[JsonInclude]`
- `[JsonNumberHandling]`

Records in C# 9 are also supported, as shown in the following example:

```

using System.Text.Json;

namespace Records
{
    public record Forecast(DateTime Date, int TemperatureC)
    {
        public string? Summary { get; init; }
    };

    public class Program
    {
        public static void Main()
        {
            Forecast forecast = new(DateTime.Now, 40)
            {
                Summary = "Hot!"
            };

            string forecastJson = JsonSerializer.Serialize<Forecast>(forecast);
            Console.WriteLine(forecastJson);
            Forecast? forecastObj = JsonSerializer.Deserialize<Forecast>(forecastJson);
            Console.WriteLine(forecastObj);
        }
    }
}

// Produces output like the following example:
// 
//{ "Date":"2020-10-21T15:26:10.5044594-07:00","TemperatureC":40,"Summary":"Hot!"}
//Forecast { Date = 10 / 21 / 2020 3:26:10 PM, TemperatureC = 40, Summary = Hot! }

```

You can apply any of the attributes to the property names, using the `property:` target on the attribute. For more information on positional records, see the article on [records](#) in the C# language reference.

For types that are immutable because all their property setters are non-public, see the following section.

`JsonConstructorAttribute` and C# 9 Record support aren't available in .NET Core 3.1.

Non-public property accessors

To enable use of a non-public property accessor, use the [\[JsonInclude\]](#) attribute, as shown in the following example:

```
using System.Text.Json;
using System.Text.Json.Serialization;

namespace NonPublicAccessors
{
    public class Forecast
    {
        public DateTime Date { get; init; }

        [JsonProperty]
        public int TemperatureC { get; private set; }

        [JsonProperty]
        public string? Summary { private get; set; }
    };

    public class Program
    {
        public static void Main()
        {
            string json = @"""Date"":""2020-10-23T09:51:03.8702889-07:00""",""TemperatureC"":40,""Summary"":""Hot""}";
            Console.WriteLine($"Input JSON: {json}");

            Forecast forecastDeserialized = JsonSerializer.Deserialize<Forecast>(json)!;
            Console.WriteLine($"Date: {forecastDeserialized.Date}");
            Console.WriteLine($"TemperatureC: {forecastDeserialized.TemperatureC}");

            json = JsonSerializer.Serialize<Forecast>(forecastDeserialized);
            Console.WriteLine($"Output JSON: {json}");
        }
    }
}

// Produces output like the following example:
// 
//Input JSON: { "Date":"2020-10-23T09:51:03.8702889-07:00", "TemperatureC":40, "Summary":"Hot" }
//Date: 10 / 23 / 2020 9:51:03 AM
//TemperatureC: 40
//Output JSON: { "Date":"2020-10-23T09:51:03.8702889-07:00", "TemperatureC":40, "Summary":"Hot" }
```

```

Imports System.Text.Json
Imports System.Text.Json.Serialization

Namespace NonPublicAccessors

    Public Class Forecast
        Public Property [Date] As Date

        Private _temperatureC As Integer

        <JsonPropertyName>
        Public Property TemperatureC As Integer
            Get
                Return _temperatureC
            End Get
            Private Set(Value As Integer)
                _temperatureC = Value
            End Set
        End Property

        Private _summary As String

        <JsonPropertyName>
        Public Property Summary As String
            Private Get
                Return _summary
            End Get
            Set(Value As String)
                _summary = Value
            End Set
        End Property

    End Class

    Public NotInheritable Class Program

        Public Shared Sub Main()
            Dim json As String = "{\"Date\":\"2020-10-23T09:51:03.8702889-07:00\",\"TemperatureC\":40,\"Summary\":\"Hot\"}"
            Console.WriteLine($"Input JSON: {json}")

            Dim forecastDeserialized As Forecast = JsonSerializer.Deserialize(Of Forecast)(json)
            Console.WriteLine($"Date: {forecastDeserialized.[Date]}")
            Console.WriteLine($"TemperatureC: {forecastDeserialized.TemperatureC}")

            json = JsonSerializer.Serialize(forecastDeserialized)
            Console.WriteLine($"Output JSON: {json}")
        End Sub

    End Class

End Namespace

' Produces output like the following example:
'
'Input JSON: { "Date":"2020-10-23T09:51:03.8702889-07:00", "TemperatureC":40, "Summary":"Hot" }
'Date: 10 / 23 / 2020 9:51:03 AM
'TemperatureC: 40
'Output JSON: { "Date":"2020-10-23T09:51:03.8702889-07:00", "TemperatureC":40, "Summary":"Hot" }

```

Non-public property accessors are not supported in .NET Core 3.1. For more information, see [the Migrate from Newtonsoft.Json article](#).

See also

- [System.Text.Json overview](#)
- [How to serialize and deserialize JSON](#)
- [Instantiate JsonSerializerOptions instances](#)
- [Enable case-insensitive matching](#)
- [Customize property names and values](#)
- [Ignore properties](#)
- [Allow invalid JSON](#)
- [Handle overflow JSON or use JsonElement or JsonNode](#)
- [Preserve references and handle circular references](#)
- [Polymorphic serialization](#)
- [Migrate from Newtonsoft.Json to System.Text.Json](#)
- [Customize character encoding](#)
- [Use DOM, Utf8JsonReader, and Utf8JsonWriter](#)
- [Write custom converters for JSON serialization](#)
- [DateTime and DateTimeOffset support](#)
- [How to use source generation](#)
- [Supported collection types](#)
- [System.Text.Json API reference](#)
- [System.Text.Json.Serialization API reference](#)

How to serialize properties of derived classes with System.Text.Json

9/20/2022 • 5 minutes to read • [Edit Online](#)

In this article, you will learn how to serialize properties of derived classes with the `System.Text.Json` namespace.

Serialize properties of derived classes

Serialization of a polymorphic type hierarchy is *not* supported. For example, if a property is defined as an interface or an abstract class, only the properties defined on the interface or abstract class are serialized, even if the runtime type has additional properties. The exceptions to this behavior are explained in this section.

For example, suppose you have a `WeatherForecast` class and a derived class `WeatherForecastDerived`:

```
public class WeatherForecast
{
    public DateTimeOffset Date { get; set; }
    public int TemperatureCelsius { get; set; }
    public string? Summary { get; set; }
}
```

```
Public Class WeatherForecast
    Public Property [Date] As DateTimeOffset
    Public Property TemperatureCelsius As Integer
    Public Property Summary As String
End Class
```

```
public class WeatherForecastDerived : WeatherForecast
{
    public int WindSpeed { get; set; }
}
```

```
Public Class WeatherForecastDerived
    Inherits WeatherForecast
    Public Property WindSpeed As Integer
End Class
```

And suppose the type argument of the `Serialize` method at compile time is `WeatherForecast`:

```
var options = new JsonSerializerOptions
{
    WriteIndented = true
};
jsonString = JsonSerializer.Serialize<WeatherForecast>(weatherForecast, options);
```

```

Dim options As JsonSerializerOptions = New JsonSerializerOptions With {
    .WriteIndented = True
}
jsonString = JsonSerializer.Serialize(weatherForecast1, options)

```

In this scenario, the `WindSpeed` property is not serialized even if the `weatherForecast` object is actually a `WeatherForecastDerived` object. Only the base class properties are serialized:

```

{
    "Date": "2019-08-01T00:00:00-07:00",
    "TemperatureCelsius": 25,
    "Summary": "Hot"
}

```

This behavior is intended to help prevent accidental exposure of data in a derived runtime-created type.

To serialize the properties of the derived type in the preceding example, use one of the following approaches:

- Call an overload of `Serialize` that lets you specify the type at run time:

```

options = new JsonSerializerOptions
{
    WriteIndented = true
};
jsonString = JsonSerializer.Serialize(weatherForecast, weatherForecast.GetType(), options);

```

```

options = New JsonSerializerOptions With {
    .WriteIndented = True
}
jsonString = JsonSerializer.Serialize(weatherForecast1, weatherForecast1.[GetType](), options)

```

- Declare the object to be serialized as `object`.

```

options = new JsonSerializerOptions
{
    WriteIndented = true
};
jsonString = JsonSerializer.Serialize<object>(weatherForecast, options);

```

```

options = New JsonSerializerOptions With {
    .WriteIndented = True
}
jsonString = JsonSerializer.Serialize(Of Object)(weatherForecast1, options)

```

In the preceding example scenario, both approaches cause the `WindSpeed` property to be included in the JSON output:

```

{
    "WindSpeed": 35,
    "Date": "2019-08-01T00:00:00-07:00",
    "TemperatureCelsius": 25,
    "Summary": "Hot"
}

```

IMPORTANT

These approaches provide polymorphic serialization only for the root object to be serialized, not for properties of that root object.

You can get polymorphic serialization for lower-level objects if you define them as type `object`. For example, suppose your `WeatherForecast` class has a property named `PreviousForecast` that can be defined as type `WeatherForecast` or `object`:

```
public class WeatherForecastWithPrevious
{
    public DateTimeOffset Date { get; set; }
    public int TemperatureCelsius { get; set; }
    public string? Summary { get; set; }
    public WeatherForecast? PreviousForecast { get; set; }
}
```

```
Public Class WeatherForecastWithPrevious
    Public Property [Date] As DateTimeOffset
    Public Property TemperatureCelsius As Integer
    Public Property Summary As String
    Public Property PreviousForecast As WeatherForecast
End Class
```

```
public class WeatherForecastWithPreviousAsObject
{
    public DateTimeOffset Date { get; set; }
    public int TemperatureCelsius { get; set; }
    public string? Summary { get; set; }
    public object? PreviousForecast { get; set; }
}
```

```
Public Class WeatherForecastWithPreviousAsObject
    Public Property [Date] As DateTimeOffset
    Public Property TemperatureCelsius As Integer
    Public Property Summary As String
    Public Property PreviousForecast As Object
End Class
```

If the `PreviousForecast` property contains an instance of `WeatherForecastDerived`:

- The JSON output from serializing `WeatherForecastWithPrevious` **doesn't include** `WindSpeed`.
- The JSON output from serializing `WeatherForecastWithPreviousAsObject` **includes** `WindSpeed`.

To serialize `WeatherForecastWithPreviousAsObject`, it isn't necessary to call `Serialize<object>` or `GetType` because the root object isn't the one that may be of a derived type. The following code example doesn't call `Serialize<object>` or `GetType`:

```
options = new JsonSerializerOptions
{
    WriteIndented = true
};
jsonString = JsonSerializer.Serialize(weatherForecastWithPreviousAsObject, options);
```

```
options = New JsonSerializerOptions With {
    .WriteIndented = True
}
jsonString = JsonSerializer.Serialize(weatherForecastWithPreviousAsObject1, options)
```

The preceding code correctly serializes `WeatherForecastWithPreviousAsObject`:

```
{
    "Date": "2019-08-01T00:00:00-07:00",
    "TemperatureCelsius": 25,
    "Summary": "Hot",
    "PreviousForecast": {
        "WindSpeed": 35,
        "Date": "2019-08-01T00:00:00-07:00",
        "TemperatureCelsius": 25,
        "Summary": "Hot"
    }
}
```

The same approach of defining properties as `object` works with interfaces. Suppose you have the following interface and implementation, and you want to serialize a class with properties that contain implementation instances:

```
namespace SystemTextJsonSamples
{
    public interface IForecast
    {
        public DateTimeOffset Date { get; set; }
        public int TemperatureCelsius { get; set; }
        public string? Summary { get; set; }
    }

    public class Forecast : IForecast
    {
        public DateTimeOffset Date { get; set; }
        public int TemperatureCelsius { get; set; }
        public string? Summary { get; set; }
        public int WindSpeed { get; set; }
    }

    public class Forecasts
    {
        public IForecast? Monday { get; set; }
        public object? Tuesday { get; set; }
    }
}
```

```

Namespace SystemTextJsonSamples

    Public Interface IForecast
        Property [Date] As DateTimeOffset
        Property TemperatureCelsius As Integer
        Property Summary As String
    End Interface

    Public Class Forecast
        Implements IForecast
        Public Property [Date] As DateTimeOffset Implements IForecast.[Date]
        Public Property TemperatureCelsius As Integer Implements IForecast.TemperatureCelsius
        Public Property Summary As String Implements IForecast.Summary
        Public Property WindSpeed As Integer
    End Class

    Public Class Forecasts
        Public Property Monday As IForecast
        Public Property Tuesday As Object
    End Class

End Namespace

```

When you serialize an instance of `Forecasts`, only `Tuesday` shows the `WindSpeed` property, because `Tuesday` is defined as `object`:

```

var forecasts = new Forecasts
{
    Monday = new Forecast
    {
        Date = DateTime.Parse("2020-01-06"),
        TemperatureCelsius = 10,
        Summary = "Cool",
        WindSpeed = 8
    },
    Tuesday = new Forecast
    {
        Date = DateTime.Parse("2020-01-07"),
        TemperatureCelsius = 11,
        Summary = "Rainy",
        WindSpeed = 10
    }
};

options = new JsonSerializerOptions
{
    WriteIndented = true
};
jsonString = JsonSerializer.Serialize(forecasts, options);

```

```

Dim forecasts1 As New Forecasts With {
    .Monday = New Forecast With {
        .[Date] = Date.Parse("2020-01-06"),
        .TemperatureCelsius = 10,
        .Summary = "Cool",
        .WindSpeed = 8
    },
    .Tuesday = New Forecast With {
        .[Date] = Date.Parse("2020-01-07"),
        .TemperatureCelsius = 11,
        .Summary = "Rainy",
        .WindSpeed = 10
    }
}

options = New JsonSerializerOptions With {
    .WriteIndented = True
}
jsonString = JsonSerializer.Serialize(forecasts1, options)

```

The following example shows the JSON that results from the preceding code:

```
{
    "Monday": {
        "Date": "2020-01-06T00:00:00-08:00",
        "TemperatureCelsius": 10,
        "Summary": "Cool"
    },
    "Tuesday": {
        "Date": "2020-01-07T00:00:00-08:00",
        "TemperatureCelsius": 11,
        "Summary": "Rainy",
        "WindSpeed": 10
    }
}
```

NOTE

This article is about serialization, not deserialization. Polymorphic deserialization is not supported, but as a workaround you can write a custom converter, such as the example in [Support polymorphic deserialization](#).

See also

- [System.Text.Json overview](#)
- [How to serialize and deserialize JSON](#)
- [Instantiate JsonSerializerOptions instances](#)
- [Enable case-insensitive matching](#)
- [Customize property names and values](#)
- [Ignore properties](#)
- [Allow invalid JSON](#)
- [Handle overflow JSON or use JsonElement or JsonNode](#)
- [Preserve references and handle circular references](#)
- [Deserialize to immutable types and non-public accessors](#)
- [Migrate from Newtonsoft.Json to System.Text.Json](#)
- [Customize character encoding](#)
- [Use DOM, Utf8JsonReader, and Utf8JsonWriter](#)

- [Write custom converters for JSON serialization](#)
- [DateTime and DateTimeOffset support](#)
- [How to use source generation](#)
- [Supported collection types](#)
- [System.Text.Json API reference](#)
- [System.Text.Json.Serialization API reference](#)

How to use a JSON document, Utf8JsonReader, and Utf8JsonWriter in System.Text.Json

9/20/2022 • 26 minutes to read • [Edit Online](#)

This article shows how to use:

- A [JSON Document Object Model \(DOM\)](#) for random access to data in a JSON payload.
- The [Utf8JsonWriter](#) type for building custom serializers.
- The [Utf8JsonReader](#) type for building custom parsers and deserializers.

JSON DOM choices

Working with a DOM is an alternative to deserialization with [JsonSerializer](#):

- When you don't have a type to deserialize into.
- When the JSON you receive doesn't have a fixed schema and must be inspected to know what it contains.

`System.Text.Json` provides two ways to build a JSON DOM:

- [JsonDocument](#) provides the ability to build a read-only DOM by using [Utf8JsonReader](#). The JSON elements that compose the payload can be accessed via the [JsonElement](#) type. The [JsonElement](#) type provides array and object enumerators along with APIs to convert JSON text to common .NET types. [JsonDocument](#) exposes a [RootElement](#) property. For more information, see [Use JsonDocument](#) later in this article.
- [JsonNode](#) and the classes that derive from it in the [System.Text.Json.Nodes](#) namespace provide the ability to create a mutable DOM. The JSON elements that compose the payload can be accessed via the [JsonNode](#), [JsonObject](#), [JsonArray](#), [JsonValue](#), and [JsonElement](#) types. For more information, see [Use JsonNode](#) later in this article.

Consider the following factors when choosing between [JsonDocument](#) and [JsonNode](#):

- The [JsonNode](#) DOM can be changed after it's created. The [JsonDocument](#) DOM is immutable.
- The [JsonDocument](#) DOM provides faster access to its data.
- Starting in .NET 6, [JsonNode](#) and the classes that derive from it in the [System.Text.Json.Nodes](#) namespace provide the ability to create a mutable DOM. For more information, see the [.NET 6 version of this article](#).

Use [JsonNode](#)

The following example shows how to use [JsonNode](#) and the other types in the [System.Text.Json.Nodes](#) namespace to:

- Create a DOM from a JSON string
- Write JSON from a DOM.
- Get a value, object, or array from a DOM.

```
using System.Text.Json;
using System.Text.Json.Nodes;

namespace JsonNodeFromStringExample;

public class Program
```

```

public class Program
{
    public static void Main()
    {
        string jsonString =
@"{
    ""Date"": ""2019-08-01T00:00:00"",
    ""Temperature"": 25,
    ""Summary"": ""Hot"",
    ""DatesAvailable"": [
        ""2019-08-01T00:00:00"",
        ""2019-08-02T00:00:00""
    ],
    ""TemperatureRanges"": {
        ""Cold"": {
            ""High"": 20,
            ""Low"": -10
        },
        ""Hot"": {
            ""High"": 60,
            ""Low"": 20
        }
    }
}
";
        // Create a JsonNode DOM from a JSON string.
        JsonNode forecastNode = JsonNode.Parse(jsonString);

        // Write JSON from a JsonNode
        var options = new JsonSerializerOptions { WriteIndented = true };
        Console.WriteLine(forecastNode!.ToJsonString(options));
        // output:
        //{
        //    "Date": "2019-08-01T00:00:00",
        //    "Temperature": 25,
        //    "Summary": "Hot",
        //    "DatesAvailable": [
        //        "2019-08-01T00:00:00",
        //        "2019-08-02T00:00:00"
        //    ],
        //    "TemperatureRanges": {
        //        "Cold": {
        //            "High": 20,
        //            "Low": -10
        //        },
        //        "Hot": {
        //            "High": 60,
        //            "Low": 20
        //        }
        //    }
        //}

        // Get value from a JsonNode.
        JsonNode temperatureNode = forecastNode!["Temperature"]!;
        Console.WriteLine($"Type={temperatureNode.GetType()}");
        Console.WriteLine($"JSON={temperatureNode.ToString()}");
        //output:
        //Type = System.Text.Json.Nodes.JsonValue`1[System.Text.Json.JsonElement]
        //JSON = 25

        // Get a typed value from a JsonNode.
        int temperatureInt = (int)forecastNode!["Temperature"]!;
        Console.WriteLine($"Value={temperatureInt}");
        //output:
        //Value=25

        // Get a typed value from a JsonNode by using GetValue<T>.
        temperatureInt = forecastNode!["Temperature"]!.GetValue<int>();
        Console.WriteLine($"TemperatureInt={temperatureInt}");
        //output:
        //TemperatureInt=25
    }
}

```

```

//output:
//Value=25

// Get a JSON object from a JsonNode.
JsonNode temperatureRanges = forecastNode!["TemperatureRanges"]!;
Console.WriteLine($"Type={temperatureRanges.GetType()}");
Console.WriteLine($"JSON={temperatureRanges.ToString()}");
//output:
//Type = System.Text.Json.Nodes.JsonObject
//JSON = { "Cold":{ "High":20,"Low":-10}, "Hot":{ "High":60,"Low":20} }

// Get a JSON array from a JsonNode.
JsonNode datesAvailable = forecastNode!["DatesAvailable"]!;
Console.WriteLine($"Type={datesAvailable.GetType()}");
Console.WriteLine($"JSON={datesAvailable.ToString()}");
//output:
//datesAvailable Type = System.Text.Json.Nodes.JsonArray
//datesAvailable JSON =["2019-08-01T00:00:00", "2019-08-02T00:00:00"]

// Get an array element value from a JsonArray.
JsonNode firstDateAvailable = datesAvailable[0]!;
Console.WriteLine($"Type={firstDateAvailable.GetType()}");
Console.WriteLine($"JSON={firstDateAvailable.ToString()}");
//output:
//Type = System.Text.Json.Nodes.JsonValue`1[System.Text.Json.JsonElement]
//JSON = "2019-08-01T00:00:00"

// Get a typed value by chaining references.
int coldHighTemperature = (int)forecastNode["TemperatureRanges"]!["Cold"]!["High"]!;
Console.WriteLine($"TemperatureRanges.Cold.High={coldHighTemperature}");
//output:
//TemperatureRanges.Cold.High = 20

// Parse a JSON array
var datesNode = JsonNode.Parse(@"[\"2019-08-01T00:00:00\", \"2019-08-02T00:00:00\"]");
JsonNode firstDate = datesNode![0]!.GetValue<DateTime>();
Console.WriteLine($"firstDate={ firstDate}");
//output:
//firstDate = "2019-08-01T00:00:00"
}

}

```

Create a JsonNode DOM with object initializers and make changes

The following example shows how to:

- Create a DOM by using object initializers.
- Make changes to a DOM.

```

using System.Text.Json;
using System.Text.Json.Nodes;

namespace JsonNodeFromObjectExample;

public class Program
{
    public static void Main()
    {
        // Create a new JsonObject using object initializers.
        var forecastObject = new JsonObject
        {
            ["Date"] = new DateTime(2019, 8, 1),
            ["Temperature"] = 25,
            ["Summary"] = "Hot",
            ["DatesAvailable"] = new JsonArray(
                new DateTime(2019, 8, 1), new DateTime(2019, 8, 2)),
            ["TemperatureRanges"] = new JsonObject
            {
                ["Cold"] = new JsonObject
                {
                    ["High"] = 20,
                    ["Low"] = -10
                },
                ["SummaryWords"] = new JsonArray("Cool", "Windy", "Humid")
            };
        };

        // Add an object.
        forecastObject!["TemperatureRanges"]!["Hot"] =
            new JsonObject { ["High"] = 60, ["Low"] = 20 };

        // Remove a property.
        forecastObject.Remove("SummaryWords");

        // Change the value of a property.
        forecastObject["Date"] = new DateTime(2019, 8, 3);

        var options = new JsonSerializerOptions { WriteIndented = true };
        Console.WriteLine(forecastObject.ToString(options));
        //output:
        //{
        //    "Date": "2019-08-03T00:00:00",
        //    "Temperature": 25,
        //    "Summary": "Hot",
        //    "DatesAvailable": [
        //        "2019-08-01T00:00:00",
        //        "2019-08-02T00:00:00"
        //    ],
        //    "TemperatureRanges": {
        //        "Cold": {
        //            "High": 20,
        //            "Low": -10
        //        },
        //        "Hot": {
        //            "High": 60,
        //            "Low": 20
        //        }
        //    }
        //}
    }
}

```

Deserialize subsections of a JSON payload

The following example shows how to use [JsonNode](#) to navigate to a subsection of a JSON tree and deserialize a

single value, a custom type, or an array from that subsection.

```
using System.Text.Json;
using System.Text.Json.Nodes;

namespace JsonNodePOCOExample;

public class TemperatureRanges : Dictionary<string, HighLowTemps>
{
}

public class HighLowTemps
{
    public int High { get; set; }
    public int Low { get; set; }
}

public class Program
{
    public static DateTime[]? DatesAvailable { get; set; }

    public static void Main()
    {
        string jsonString =
@"
{
    ""Date"": ""2019-08-01T00:00:00"",
    ""Temperature"": 25,
    ""Summary"": ""Hot"",
    ""DatesAvailable"": [
        ""2019-08-01T00:00:00"",
        ""2019-08-02T00:00:00""
    ],
    ""TemperatureRanges"": {
        ""Cold"": {
            ""High"": 20,
            ""Low"": -10
        },
        ""Hot"": {
            ""High"": 60,
            ""Low"": 20
        }
    }
}
";
        // Parse all of the JSON.
        JsonNode forecastNode = JsonNode.Parse(jsonString);

        // Get a single value
        int hotHigh = forecastNode["TemperatureRanges"]!["Hot"]!["High"]!.GetValue<int>();
        Console.WriteLine($"Hot.High={hotHigh}");
        // output:
        //Hot.High=60

        // Get a subsection and deserialize it into a custom type.
        JsonObject temperatureRangesObject = forecastNode!["TemperatureRanges"]!.AsObject();
        using var stream = new MemoryStream();
        using var writer = new Utf8JsonWriter(stream);
        temperatureRangesObject.WriteTo(writer);
        writer.Flush();
        TemperatureRanges? temperatureRanges =
            JsonSerializer.Deserialize<TemperatureRanges>(stream.ToArray());
        Console.WriteLine($"Cold.Low={temperatureRanges!["Cold"].Low}, Hot.High={temperatureRanges["Hot"].High}");
        // output:
        //Cold.Low=-10, Hot.High=60

        // Get a subsection and deserialize it into an array.
        JsonArray datesAvailable = forecastNode!["DatesAvailable"]!.AsArray()!;
}
```

```
        Console.WriteLine($"DatesAvailable[0]={datesAvailable[0]}");
        // output:
        //DatesAvailable[0]=8/1/2019 12:00:00 AM
    }
}
```

JsonNode average grade example

The following example selects a JSON array that has integer values and calculates an average value:

```

using System.Text.Json.Nodes;

namespace JsonNodeAverageGradeExample;

public class Program
{
    public static void Main()
    {
        string jsonString =
@"{
    ""Class Name"": ""Science"",
    ""Teacher\u0027s Name"": ""Jane"",
    ""Semester"": ""2019-01-01"",
    ""Students"": [
        {
            ""Name"": ""John"",
            ""Grade"": 94.3
        },
        {
            ""Name"": ""James"",
            ""Grade"": 81.0
        },
        {
            ""Name"": ""Julia"",
            ""Grade"": 91.9
        },
        {
            ""Name"": ""Jessica"",
            ""Grade"": 72.4
        },
        {
            ""Name"": ""Johnathan""
        }
    ],
    ""Final"": true
}
";
        double sum = 0;
        int count = 0;

        JsonNode document = JsonNode.Parse(jsonString)!

        JsonNode root = document.Root;
        JsonArray studentsArray = root["Students"]!.AsArray();

        count = studentsArray.Count;

        foreach (JsonNode? student in studentsArray)
        {
            if (student?["Grade"] is JsonNode gradeNode)
            {
                sum += (double)gradeNode;
            }
            else
            {
                sum += 70;
            }
        }

        double average = sum / count;
        Console.WriteLine($"Average grade : {average}");
    }
}

// output:
//Average grade : 81.92

```

The preceding code:

- Calculates an average grade for objects in a `Students` array that have a `Grade` property.
- Assigns a default grade of 70 for students who don't have a grade.
- Gets the number of students from the `Count` property of `JSONArray`.

```
JsonNode with JsonSerializerOptions
```

You can use `JsonSerializer` to serialize and deserialize an instance of `JsonNode`. However, if you use an overload that takes `JsonSerializerOptions`, the options instance is only used to get custom converters. Other features of the options instance are not used. For example, if you set `JsonSerializerOptions.DefaultIgnoreCondition` to `WhenWritingNull` and call `JsonSerializer` with an overload that takes `JsonSerializerOptions`, null properties won't be ignored.

The same limitation applies to the `JsonNode` methods that take a `JsonSerializerOptions` parameter: `WriteTo(Utf8JsonWriter, JsonSerializerOptions)` and `ToJsonObject(JsonSerializerOptions)`. These APIs use `JsonSerializerOptions` only to get custom converters.

The following example illustrates the result of using methods that take a `JsonSerializerOptions` parameter and serialize a `JsonNode` instance:

```

using System.Text;
using System.Text.Json;
using System.Text.Json.Nodes;
using System.Text.Json.Serialization;

namespace JsonNodeWithJsonSerializerOptions;

public class Program
{
    public static void Main()
    {
        Person person = new Person { Name = "Nancy" };

        // Default serialization - Address property included with null token.
        // Output: {"Name":"Nancy","Address":null}
        string personJsonWithNull = JsonSerializer.Serialize(person);
        Console.WriteLine(personJsonWithNull);

        // Serialize and ignore null properties - null Address property is omitted
        // Output: {"Name":"Nancy"}
        JsonSerializerOptions options = new()
        {
            DefaultIgnoreCondition = JsonIgnoreCondition.WhenWritingNull
        };
        string personJsonWithoutNull = JsonSerializer.Serialize(person, options);
        Console.WriteLine(personJsonWithoutNull);

        // Ignore null properties doesn't work when serializing JsonNode instance
        // by using JsonSerializer.
        // Output: {"Name":"Nancy","Address":null}
        var personJsonNode = JsonSerializer.Deserialize<JsonNode>(personJsonWithNull);
        personJsonWithNull = JsonSerializer.Serialize(personJsonNode, options);
        Console.WriteLine(personJsonWithNull);

        // Ignore null properties doesn't work when serializing JsonNode instance
        // by using JsonNode.ToString method.
        // Output: {"Name":"Nancy","Address":null}
        personJsonWithNull = personJsonNode!.ToString(options);
        Console.WriteLine(personJsonWithNull);

        // Ignore null properties doesn't work when serializing JsonNode instance
        // by using JsonNode.WriteTo method.
        // Output: {"Name":"Nancy","Address":null}
        using var stream = new MemoryStream();
        using var writer = new Utf8JsonWriter(stream);
        personJsonNode!.WriteTo(writer, options);
        writer.Flush();
        personJsonWithNull = Encoding.UTF8.GetString(stream.ToArray());
        Console.WriteLine(personJsonWithNull);
    }
}

public class Person
{
    public string? Name { get; set; }
    public string? Address { get; set; }
}

```

If you need features of `JsonSerializerOptions` other than custom converters, use `JsonSerializer` with strongly typed targets (such as the `Person` class in this example) rather than `JsonNode`.

Use `JsonDocument`

The following example shows how to use the `JsonDocument` class for random access to data in a JSON string:

```

double sum = 0;
int count = 0;

using (JsonDocument document = JsonDocument.Parse(jsonString))
{
    JsonElement root = document.RootElement;
    JsonElement studentsElement = root.GetProperty("Students");
    foreach (JsonElement student in studentsElement.EnumerateArray())
    {
        if (student.TryGetProperty("Grade", out JsonElement gradeElement))
        {
            sum += gradeElement.GetDouble();
        }
        else
        {
            sum += 70;
        }
        count++;
    }
}

double average = sum / count;
Console.WriteLine($"Average grade : {average}");

```

```

Dim sum As Double = 0
Dim count As Integer = 0
Using document As JsonDocument = JsonDocument.Parse(jsonString)
    Dim root As JsonElement = document.RootElement
    Dim studentsElement As JsonElement = root.GetProperty("Students")
    For Each student As JsonElement In studentsElement.EnumerateArray()
        Dim gradeElement As JsonElement = Nothing
        If student.TryGetProperty("Grade", gradeElement) Then
            sum += gradeElement.GetDouble()
        Else
            sum += 70
        End If
        count += 1
    Next
End Using

Dim average As Double = sum / count
Console.WriteLine($"Average grade : {average}")

```

The preceding code:

- Assumes the JSON to analyze is in a string named `jsonString`.
- Calculates an average grade for objects in a `Students` array that have a `Grade` property.
- Assigns a default grade of 70 for students who don't have a grade.
- Creates the `JsonDocument` instance in a `using statement` because `JsonDocument` implements `IDisposable`. After a `JsonDocument` instance is disposed, you lose access to all of its `JsonElement` instances also. To retain access to a `JsonElement` instance, make a copy of it before the parent `JsonDocument` instance is disposed. To make a copy, call `JsonElement.Clone`. For more information, see [JsonDocument is IDisposable](#).

The preceding example code counts students by incrementing a `count` variable with each iteration. An alternative is to call `GetArrayLength`, as shown in the following example:

```

double sum = 0;
int count = 0;

using (JsonDocument document = JsonDocument.Parse(jsonString))
{
    JsonElement root = document.RootElement;
    JsonElement studentsElement = root.GetProperty("Students");

    count = studentsElement.GetArrayLength();

    foreach (JsonElement student in studentsElement.EnumerateArray())
    {
        if (student.TryGetProperty("Grade", out JsonElement gradeElement))
        {
            sum += gradeElement.GetDouble();
        }
        else
        {
            sum += 70;
        }
    }
}

double average = sum / count;
Console.WriteLine($"Average grade : {average}");

```

```

Dim sum As Double = 0
Dim count As Integer = 0
Using document As JsonDocument = JsonDocument.Parse(jsonString)
    Dim root As JsonElement = document.RootElement
    Dim studentsElement As JsonElement = root.GetProperty("Students")

    count = studentsElement.GetArrayLength()

    For Each student As JsonElement In studentsElement.EnumerateArray()
        Dim gradeElement As JsonElement = Nothing
        If student.TryGetProperty("Grade", gradeElement) Then
            sum += gradeElement.GetDouble()
        Else
            sum += 70
        End If
    Next
End Using

Dim average As Double = sum / count
Console.WriteLine($"Average grade : {average}")

```

Here's an example of the JSON that this code processes:

```
{
    "Class Name": "Science",
    "Teacher\u0027s Name": "Jane",
    "Semester": "2019-01-01",
    "Students": [
        {
            "Name": "John",
            "Grade": 94.3
        },
        {
            "Name": "James",
            "Grade": 81.0
        },
        {
            "Name": "Julia",
            "Grade": 91.9
        },
        {
            "Name": "Jessica",
            "Grade": 72.4
        },
        {
            "Name": "Johnathan"
        }
    ],
    "Final": true
}
```

For a similar example that uses `JsonNode` instead of `JsonDocument`, see [JsonNode average grade example](#).

How to search a `JsonDocument` and `JsonElement` for sub-elements

Searches on `JsonElement` require a sequential search of the properties and hence are relatively slow (for example when using `TryGetProperty`). `System.Text.Json` is designed to minimize initial parse time rather than lookup time. Therefore, use the following approaches to optimize performance when searching through a `JsonDocument` object:

- Use the built-in enumerators (`EnumerateArray` and `EnumerateObject`) rather than doing your own indexing or loops.
- Don't do a sequential search on the whole `JsonDocument` through every property by using `RootElement`. Instead, search on nested JSON objects based on the known structure of the JSON data. For example, the preceding code examples look for a `Grade` property in `Student` objects by looping through the `Student` objects and getting the value of `Grade` for each, rather than searching through all `JsonElement` objects looking for `Grade` properties. Doing the latter would result in unnecessary passes over the same data.

Use `JsonDocument` to write JSON

The following example shows how to write JSON from a `JsonDocument`:

```

string jsonString = File.ReadAllText(inputFileName);

var writerOptions = new JsonWriterOptions
{
    Indented = true
};

var documentOptions = new JsonDocumentOptions
{
    CommentHandling = JsonCommentHandling.Skip
};

using FileStream fs = File.Create(outputFileName);
using var writer = new Utf8JsonWriter(fs, options: writerOptions);
using JsonDocument document = JsonDocument.Parse(jsonString, documentOptions);

JsonElement root = document.RootElement;

if (root.ValueKind == JsonValueKind.Object)
{
    writer.WriteStartObject();
}
else
{
    return;
}

foreach (JsonProperty property in root.EnumerateObject())
{
    property.WriteTo(writer);
}

writer.WriteEndObject();

writer.Flush();

```

```

Dim jsonString As String = File.ReadAllText(inputFileName)

Dim writerOptions As JsonWriterOptions = New JsonWriterOptions With {
    .Indented = True
}

Dim documentOptions As JsonDocumentOptions = New JsonDocumentOptions With {
    .CommentHandling = JsonCommentHandling.Skip
}

Dim fs As FileStream = File.Create(outputFileName)
Dim writer As Utf8JsonWriter = New Utf8JsonWriter(fs, options:=writerOptions)
Dim document As JsonDocument = JsonDocument.Parse(jsonString, documentOptions)

Dim root As JsonElement = document.RootElement

If root.ValueKind = JsonValueKind.[Object] Then
    writer.WriteStartObject()
Else
    Return
End If

For Each [property] As JsonProperty In root.EnumerateObject()
    [property].WriteTo(writer)
Next

writer.WriteEndObject()

writer.Flush()

```

The preceding code:

- Reads a JSON file, loads the data into a `JsonDocument`, and writes formatted (pretty-printed) JSON to a file.
- Uses `JsonDocumentOptions` to specify that comments in the input JSON are allowed but ignored.
- When finished, calls `Flush` on the writer. An alternative is to let the writer auto-flush when it's disposed.

Here's an example of JSON input to be processed by the example code:

```
{"Class Name": "Science", "Teacher's Name": "Jane", "Semester": "2019-01-01", "Students": [{"Name": "John", "Grade": 94.3}, {"Name": "James", "Grade": 81.0}, {"Name": "Julia", "Grade": 91.9}, {"Name": "Jessica", "Grade": 72.4}, {"Name": "Johnathan"}], "Final": true}
```

The result is the following pretty-printed JSON output:

```
{
  "Class Name": "Science",
  "Teacher\u0027s Name": "Jane",
  "Semester": "2019-01-01",
  "Students": [
    {
      "Name": "John",
      "Grade": 94.3
    },
    {
      "Name": "James",
      "Grade": 81.0
    },
    {
      "Name": "Julia",
      "Grade": 91.9
    },
    {
      "Name": "Jessica",
      "Grade": 72.4
    },
    {
      "Name": "Johnathan"
    }
  ],
  "Final": true
}
```

JsonDocument is IDisposable

`JsonDocument` builds an in-memory view of the data into a pooled buffer. Therefore the `JsonDocument` type implements `IDisposable` and needs to be used inside a `using` block.

Only return a `JsonDocument` from your API if you want to transfer lifetime ownership and dispose responsibility to the caller. In most scenarios, that isn't necessary. If the caller needs to work with the entire JSON document, return the `Clone` of the `RootElement`, which is a `JsonElement`. If the caller needs to work with a particular element within the JSON document, return the `Clone` of that `JsonElement`. If you return the `RootElement` or a sub-element directly without making a `Clone`, the caller won't be able to access the returned `JsonElement` after the `JsonDocument` that owns it is disposed.

Here's an example that requires you to make a `Clone`:

```
public JsonElement LookAndFeel(JsonElement source)
{
    string json = File.ReadAllText(source.GetProperty("fileName").GetString());

    using (JsonDocument doc = JsonDocument.Parse(json))
    {
        return doc.RootElement.Clone();
    }
}
```

The preceding code expects a `JsonElement` that contains a `fileName` property. It opens the JSON file and creates a `JsonDocument`. The method assumes that the caller wants to work with the entire document, so it returns the `Clone` of the `RootElement`.

If you receive a `JsonElement` and are returning a sub-element, it's not necessary to return a `Clone` of the sub-element. The caller is responsible for keeping alive the `JsonDocument` that the passed-in `JsonElement` belongs to. For example:

```
public JsonElement ReturnFileName(JsonElement source)
{
    return source.GetProperty("fileName");
}
```

`JsonDocument` **with** `JsonSerializerOptions`

You can use `JsonSerializer` to serialize and deserialize an instance of `JsonDocument`. However, the implementation for reading and writing `JsonDocument` instances by using `JsonSerializer` is a wrapper over the `JsonDocument.ParseValue(Utf8JsonReader)` and `JsonDocument.WriteTo(Utf8JsonWriter)`. This wrapper does not forward any `JsonSerializerOptions` (serializer features) to `Utf8JsonReader` or `Utf8JsonWriter`. For example, if you set `JsonSerializerOptions.DefaultIgnoreCondition` to `WhenWritingNull` and call `JsonSerializer` with an overload that takes `JsonSerializerOptions`, null properties won't be ignored.

The following example illustrates the result of using methods that take a `JsonSerializerOptions` parameter and serialize a `JsonDocument` instance:

```

using System.Text;
using System.Text.Json;
using System.Text.Json.Nodes;
using System.Text.Json.Serialization;

namespace JsonDocumentWithJsonSerializerOptions;

public class Program
{
    public static void Main()
    {
        Person person = new Person { Name = "Nancy" };

        // Default serialization - Address property included with null token.
        // Output: {"Name":"Nancy","Address":null}
        string personJsonWithNull = JsonSerializer.Serialize(person);
        Console.WriteLine(personJsonWithNull);

        // Serialize and ignore null properties - null Address property is omitted
        // Output: {"Name":"Nancy"}
        JsonSerializerOptions options = new()
        {
            DefaultIgnoreCondition = JsonIgnoreCondition.WhenWritingNull
        };
        string personJsonWithoutNull = JsonSerializer.Serialize(person, options);
        Console.WriteLine(personJsonWithoutNull);

        // Ignore null properties doesn't work when serializing JsonDocument instance
        // by using JsonSerializer.
        // Output: {"Name":"Nancy","Address":null}
        var personJsonDocument = JsonSerializer.Deserialize<JsonDocument>(personJsonWithNull);
        personJsonWithNull = JsonSerializer.Serialize(personJsonDocument, options);
        Console.WriteLine(personJsonWithNull);
    }
}

public class Person
{
    public string? Name { get; set; }

    public string? Address { get; set; }
}

```

If you need features of `JsonSerializerOptions`, use `JsonSerializer` with strongly typed targets (such as the `Person` class in this example) rather than `JsonDocument`.

Use `Utf8JsonWriter`

`Utf8JsonWriter` is a high-performance way to write UTF-8 encoded JSON text from common .NET types like `String`, `Int32`, and `DateTime`. The writer is a low-level type that can be used to build custom serializers. The `JsonSerializer.Serialize` method uses `Utf8JsonWriter` under the covers.

The following example shows how to use the `Utf8JsonWriter` class:

```

var options = new JsonWriterOptions
{
    Indented = true
};

using var stream = new MemoryStream();
using var writer = new Utf8JsonWriter(stream, options);

writer.WriteStartObject();
writer.WriteString("date", DateTimeOffset.UtcNow);
writer.WriteNumber("temp", 42);
writer.WriteEndObject();
writer.Flush();

string json = Encoding.UTF8.GetString(stream.ToArray());
Console.WriteLine(json);

```

```

Dim options As JsonWriterOptions = New JsonWriterOptions With {
    .Indented = True
}

Dim stream As MemoryStream = New MemoryStream
Dim writer As Utf8JsonWriter = New Utf8JsonWriter(stream, options)

writer.WriteStartObject()
writer.WriteString("date", DateTimeOffset.UtcNow)
writer.WriteNumber("temp", 42)
writer.WriteEndObject()
writer.Flush()

Dim json As String = Encoding.UTF8.GetString(stream.ToArray())
Console.WriteLine(json)

```

Write with UTF-8 text

To achieve the best possible performance while using the `Utf8JsonWriter`, write JSON payloads already encoded as UTF-8 text rather than as UTF-16 strings. Use `JsonEncodedText` to cache and pre-encode known string property names and values as statics, and pass those to the writer, rather than using UTF-16 string literals. This is faster than caching and using UTF-8 byte arrays.

This approach also works if you need to do custom escaping. `System.Text.Json` doesn't let you disable escaping while writing a string. However, you could pass in your own custom `JavaScriptEncoder` as an option to the writer, or create your own `JsonEncodedText` that uses your `JavascriptEncoder` to do the escaping, and then write the `JsonEncodedText` instead of the string. For more information, see [Customize character encoding](#).

Write raw JSON

In some scenarios, you might want to write "raw" JSON to a JSON payload that you're creating with `Utf8JsonWriter`. You can use `Utf8JsonWriter.WriteRawValue` to do that. Here are typical scenarios:

- You have an existing JSON payload that you want to enclose in new JSON.
- You want to format values differently from the default `Utf8JsonWriter` formatting.

For example, you might want to customize number formatting. By default, `System.Text.Json` omits the decimal point for whole numbers, writing `1` rather than `1.0`, for example. The rationale is that writing fewer bytes is good for performance. But suppose the consumer of your JSON treats numbers with decimals as doubles, and numbers without decimals as integers. You might want to ensure that the numbers in an array are all recognized as doubles, by writing a decimal point and zero for whole numbers. The following example shows how to do that:

```
using System.Text;
using System.Text.Json;

namespace WriteRawJson;

public class Program
{
    public static void Main()
    {
        JsonWriterOptions writerOptions = new() { Indented = true, };

        using MemoryStream stream = new();
        using Utf8JsonWriter writer = new(stream, writerOptions);

        writer.WriteStartObject();

        writer.WriteStartArray("defaultJsonFormatting");
        foreach (double number in new double[] { 50.4, 51 })
        {
            writer.WriteStartObject();
            writer.WritePropertyName("value");
            writer.WriteNumberValue(number);
            writer.WriteEndObject();
        }
        writer.WriteEndArray();

        writer.WriteStartArray("customJsonFormatting");
        foreach (double result in new double[] { 50.4, 51 })
        {
            writer.WriteStartObject();
            writer.WritePropertyName("value");
            writer.WriteRawValue(
                FormatNumberValue(result), skipInputValidation: true);
            writer.WriteEndObject();
        }
        writer.WriteEndArray();

        writer.WriteEndObject();
        writer.Flush();

        string json = Encoding.UTF8.GetString(stream.ToArray());
        Console.WriteLine(json);
    }

    static string FormatNumberValue(double numberValue)
    {
        return numberValue == Convert.ToInt32(numberValue) ?
            numberValue.ToString() + ".0" : numberValue.ToString();
    }
}

// output:
//{
//  "defaultJsonFormatting": [
//    {
//      "value": 50.4
//    },
//    {
//      "value": 51
//    }
//  ],
//  "customJsonFormatting": [
//    {
//      "value": 50.4
//    },
//    {
//      "value": 51.0
//    }
//  ]
//}
```

Customize character escaping

The `StringEscapeHandling` setting of `JsonTextWriter` offers options to escape all non-ASCII characters or HTML characters. By default, `Utf8JsonWriter` escapes all non-ASCII and HTML characters. This escaping is done for defense-in-depth security reasons. To specify a different escaping policy, create a `JavaScriptEncoder` and set `JsonWriterOptions.Encoder`. For more information, see [Customize character encoding](#).

Write null values

To write null values by using `Utf8JsonWriter`, call:

- `WriteNull` to write a key-value pair with null as the value.
- `WriteNullValue` to write null as an element of a JSON array.

For a string property, if the string is null, `WriteString` and `WriteStringValue` are equivalent to `WriteNull` and `WriteNullValue`.

Write Timespan, Uri, or char values

To write `Timespan`, `Uri`, or `char` values, format them as strings (by calling `ToString()`, for example) and call `WriteStringValue`.

Use `Utf8JsonReader`

`Utf8JsonReader` is a high-performance, low allocation, forward-only reader for UTF-8 encoded JSON text, read from a `ReadOnlySpan<byte>` or `ReadOnlySequence<byte>`. The `Utf8JsonReader` is a low-level type that can be used to build custom parsers and deserializers. The `JsonSerializer.Deserialize` method uses `Utf8JsonReader` under the covers. The `Utf8JsonReader` can't be used directly from Visual Basic code. For more information, see [Visual Basic support](#).

The following example shows how to use the `Utf8JsonReader` class:

```

var options = new JsonReaderOptions
{
    AllowTrailingCommas = true,
    CommentHandling = JsonCommentHandling.Skip
};
var reader = new Utf8JsonReader(jsonUtf8Bytes, options);

while (reader.Read())
{
    Console.Write(reader.TokenType);

    switch (reader.TokenType)
    {
        case JsonTokenType.PropertyName:
        case JsonTokenType.String:
            {
                string? text = reader.GetString();
                Console.Write(" ");
                Console.WriteLine(text);
                break;
            }

        case JsonTokenType.Number:
            {
                int intValue = reader.GetInt32();
                Console.Write(" ");
                Console.WriteLine(intValue);
                break;
            }

        // Other token types elided for brevity
    }
    Console.WriteLine();
}

```

' This code example doesn't apply to Visual Basic. For more information, go to the following URL:
' <https://docs.microsoft.com/dotnet/standard/serialization/system-text-json-how-to#visual-basic-support>

The preceding code assumes that the `jsonUtf8` variable is a byte array that contains valid JSON, encoded as UTF-8.

Filter data using `Utf8JsonReader`

The following example shows how to synchronously read a file, and search for a value.

```

using System.Text;
using System.Text.Json;

namespace SystemTextJsonSamples
{
    public class Utf8ReaderFromFile
    {
        private static readonly byte[] s_nameUtf8 = Encoding.UTF8.GetBytes("name");
        private static ReadOnlySpan<byte> Utf8Bom => new byte[] { 0xEF, 0xBB, 0xBF };

        public static void Run()
        {
            // ReadAllBytes if the file encoding is UTF-8:
            string fileName = "UniversitiesUtf8.json";
            ReadOnlySpan<byte> jsonReadOnlySpan = File.ReadAllBytes(fileName);

            // Read past the UTF-8 BOM bytes if a BOM exists.
            if (jsonReadOnlySpan.StartsWith(Utf8Bom))
            {
                jsonReadOnlySpan = jsonReadOnlySpan.Slice(Utf8Bom.Length);
            }

            // Or read as UTF-16 and transcode to UTF-8 to convert to a ReadOnlySpan<byte>
            //string fileName = "Universities.json";
            //string jsonString = File.ReadAllText(fileName);
            //ReadOnlySpan<byte> jsonReadOnlySpan = Encoding.UTF8.GetBytes(jsonString);

            int count = 0;
            int total = 0;

            var reader = new Utf8JsonReader(jsonReadOnlySpan);

            while (reader.Read())
            {
                JsonTokenType tokenType = reader.TokenType;

                switch (tokenType)
                {
                    case JsonTokenType.StartObject:
                        total++;
                        break;
                    case JsonTokenType.PropertyName:
                        if (reader.ValueTextEquals(s_nameUtf8))
                        {
                            // Assume valid JSON, known schema
                            reader.Read();
                            if (reader.GetString()!.EndsWith("University"))
                            {
                                count++;
                            }
                        }
                        break;
                }
            }
            Console.WriteLine($"{count} out of {total} have names that end with 'University'");
        }
    }
}

```

' This code example doesn't apply to Visual Basic. For more information, go to the following URL:
' <https://docs.microsoft.com/dotnet/standard/serialization/system-text-json-how-to#visual-basic-support>

For an asynchronous version of this example, see [.NET samples JSON project](#).

The preceding code:

- Assumes the JSON contains an array of objects and each object may contain a "name" property of type string.
- Counts objects and "name" property values that end with "University".
- Assumes the file is encoded as UTF-16 and transcodes it into UTF-8. A file encoded as UTF-8 can be read directly into a `ReadOnlySpan<byte>`, by using the following code:

```
ReadOnlySpan<byte> jsonReadOnlySpan = File.ReadAllBytes(fileName);
```

If the file contains a UTF-8 byte order mark (BOM), remove it before passing the bytes to the `Utf8JsonReader`, since the reader expects text. Otherwise, the BOM is considered invalid JSON, and the reader throws an exception.

Here's a JSON sample that the preceding code can read. The resulting summary message is "2 out of 4 have names that end with 'University'" :

```
[
  {
    "web_pages": [ "https://contoso.edu/" ],
    "alpha_two_code": "US",
    "state-province": null,
    "country": "United States",
    "domains": [ "contoso.edu" ],
    "name": "Contoso Community College"
  },
  {
    "web_pages": [ "http://fabrikam.edu/" ],
    "alpha_two_code": "US",
    "state-province": null,
    "country": "United States",
    "domains": [ "fabrikam.edu" ],
    "name": "Fabrikam Community College"
  },
  {
    "web_pages": [ "http://www.contosouniversity.edu/" ],
    "alpha_two_code": "US",
    "state-province": null,
    "country": "United States",
    "domains": [ "contosouniversity.edu" ],
    "name": "Contoso University"
  },
  {
    "web_pages": [ "http://www.fabrikamuniversity.edu/" ],
    "alpha_two_code": "US",
    "state-province": null,
    "country": "United States",
    "domains": [ "fabrikamuniversity.edu" ],
    "name": "Fabrikam University"
  }
]
```

Read from a stream using `Utf8JsonReader`

When reading a large file (a gigabyte or more in size, for example), you might want to avoid having to load the entire file into memory at once. For this scenario, you can use a `FileStream`.

When using the `Utf8JsonReader` to read from a stream, the following rules apply:

- The buffer containing the partial JSON payload must be at least as large as the largest JSON token within it so that the reader can make forward progress.

- The buffer must be at least as large as the largest sequence of white space within the JSON.
- The reader doesn't keep track of the data it has read until it completely reads the next [TokenType](#) in the JSON payload. So when there are bytes left over in the buffer, you have to pass them to the reader again. You can use [BytesConsumed](#) to determine how many bytes are left over.

The following code illustrates how to read from a stream. The example shows a [MemoryStream](#). Similar code will work with a [FileStream](#), except when the [FileStream](#) contains a UTF-8 BOM at the start. In that case, you need to strip those three bytes from the buffer before passing the remaining bytes to the [Utf8JsonReader](#). Otherwise the reader would throw an exception, since the BOM is not considered a valid part of the JSON.

The sample code starts with a 4KB buffer and doubles the buffer size each time it finds that the size is not large enough to fit a complete JSON token, which is required for the reader to make forward progress on the JSON payload. The JSON sample provided in the snippet triggers a buffer size increase only if you set a very small initial buffer size, for example, 10 bytes. If you set the initial buffer size to 10, the [Console.WriteLine](#) statements illustrate the cause and effect of buffer size increases. At the 4KB initial buffer size, the entire sample JSON is shown by each [Console.WriteLine](#), and the buffer size never has to be increased.

```
using System.Text;
using System.Text.Json;

namespace SystemTextJsonSamples
{
    public class Utf8ReaderPartialRead
    {
        public static void Run()
        {
            var jsonString = @"
                ""Date"": ""2019-08-01T00:00:00-07:00"",
                ""Temperature"": 25,
                ""TemperatureRanges"": {
                    ""Cold"": { ""High"": 20, ""Low"": -10 },
                    ""Hot"": { ""High"": 60, ""Low"": 20 }
                },
                ""Summary"": ""Hot"",
            ";
            byte[] bytes = Encoding.UTF8.GetBytes(jsonString);
            var stream = new MemoryStream(bytes);

            var buffer = new byte[4096];

            // Fill the buffer.
            // For this snippet, we're assuming the stream is open and has data.
            // If it might be closed or empty, check if the return value is 0.
            stream.Read(buffer);

            // We set isFinalBlock to false since we expect more data in a subsequent read from the stream.
            var reader = new Utf8JsonReader(buffer, isFinalBlock: false, state: default);
            Console.WriteLine($"String in buffer is: {Encoding.UTF8.GetString(buffer)}");

            // Search for "Summary" property name
            while (reader.TokenType != JsonTokenType.PropertyName || !reader.ValueTextEquals("Summary"))
            {
                if (!reader.Read())
                {
                    // Not enough of the JSON is in the buffer to complete a read.
                    GetMoreBytesFromStream(stream, ref buffer, ref reader);
                }
            }

            // Found the "Summary" property name.
            Console.WriteLine($"String in buffer is: {Encoding.UTF8.GetString(buffer)}");
            while (!reader.Read())
            {

```

```

        // Not enough of the JSON is in the buffer to complete a read.
        GetMoreBytesFromStream(stream, ref buffer, ref reader);
    }

    // Display value of Summary property, that is, "Hot".
    Console.WriteLine($"Got property value: {reader.GetString()}");
}

private static void GetMoreBytesFromStream(
    MemoryStream stream, ref byte[] buffer, ref Utf8JsonReader reader)
{
    int bytesRead;
    if (reader.BytesConsumed < buffer.Length)
    {
        ReadOnlySpan<byte> leftover = buffer.AsSpan((int)reader.BytesConsumed);

        if (leftover.Length == buffer.Length)
        {
            Array.Resize(ref buffer, buffer.Length * 2);
            Console.WriteLine($"Increased buffer size to {buffer.Length}");
        }

        leftover.CopyTo(buffer);
        bytesRead = stream.Read(buffer.AsSpan(leftover.Length));
    }
    else
    {
        bytesRead = stream.Read(buffer);
    }
    Console.WriteLine($"String in buffer is: {Encoding.UTF8.GetString(buffer)}");
    reader = new Utf8JsonReader(buffer, isFinalBlock: bytesRead == 0, reader.CurrentState);
}
}
}
}

```

' This code example doesn't apply to Visual Basic. For more information, go to the following URL:
 ' <https://docs.microsoft.com/dotnet/standard/serialization/system-text-json-how-to#visual-basic-support>

The preceding example sets no limit to how large the buffer can grow. If the token size is too large, the code could fail with an [OutOfMemoryException](#) exception. This can happen if the JSON contains a token that is around 1 GB or more in size, because doubling the 1 GB size results in a size that is too large to fit into an `int32` buffer.

Utf8JsonReader is a ref struct

Because the `Utf8JsonReader` type is a *ref struct*, it has [certain limitations](#). For example, it can't be stored as a field on a class or struct other than a ref struct. To achieve high performance, this type must be a `ref struct` since it needs to cache the input `ReadOnlySpan<byte>`, which itself is a ref struct. In addition, this type is mutable since it holds state. Therefore, **pass it by reference** rather than by value. Passing it by value would result in a struct copy and the state changes would not be visible to the caller. For more information about how to use ref structs, see [Write safe and efficient C# code](#).

Read UTF-8 text

To achieve the best possible performance while using the `Utf8JsonReader`, read JSON payloads already encoded as UTF-8 text rather than as UTF-16 strings. For a code example, see [Filter data using Utf8JsonReader](#).

Read with multi-segment ReadOnlySequence

If your JSON input is a `ReadOnlySpan<byte>`, each JSON element can be accessed from the `ValueSpan` property on the reader as you go through the read loop. However, if your input is a `ReadOnlySequence<byte>` (which is the result of reading from a `PipeReader`), some JSON elements might straddle multiple segments of the `ReadOnlySequence<byte>` object. These elements would not be accessible from `ValueSpan` in a contiguous

memory block. Instead, whenever you have a multi-segment `ReadOnlySequence<byte>` as input, poll the `HasValueSequence` property on the reader to figure out how to access the current JSON element. Here's a recommended pattern:

```
while (reader.Read())
{
    switch (reader.TokenType)
    {
        // ...
        ReadOnlySpan<byte> jsonElement = reader.HasValueSequence ?
            reader.ValueSequence.ToArray() :
            reader.ValueSpan;
        // ...
    }
}
```

Use `ValueTextEquals` for property name lookups

Don't use `ValueSpan` to do byte-by-byte comparisons by calling `SequenceEqual` for property name lookups. Call `ValueTextEquals` instead, because that method unescapes any characters that are escaped in the JSON. Here's an example that shows how to search for a property that is named "name":

```
private static readonly byte[] s_nameUtf8 = Encoding.UTF8.GetBytes("name");
```

```
while (reader.Read())
{
    switch (reader.TokenType)
    {
        case JsonTokenType.StartObject:
            total++;
            break;
        case JsonTokenType.PropertyName:
            if (reader.ValueTextEquals(s_nameUtf8))
            {
                count++;
            }
            break;
    }
}
```

Read null values into nullable value types

The built-in `System.Text.Json` APIs return only non-nullable value types. For example, `Utf8JsonReader.GetBoolean` returns a `bool`. It throws an exception if it finds `Null` in the JSON. The following examples show two ways to handle nulls, one by returning a nullable value type and one by returning the default value:

```
public bool? ReadAsNullableBoolean()
{
    _reader.Read();
    if (_reader.TokenType == JsonTokenType.Null)
    {
        return null;
    }
    if (_reader.TokenType != JsonTokenType.True && _reader.TokenType != JsonTokenType.False)
    {
        throw new JsonException();
    }
    return _reader.GetBoolean();
}
```

```
public bool ReadAsBoolean(bool defaultValue)
{
    _reader.Read();
    if (_reader.TokenType == JsonTokenType.Null)
    {
        return defaultValue;
    }
    if (_reader.TokenType != JsonTokenType.True && _reader.TokenType != JsonTokenType.False)
    {
        throw new JsonException();
    }
    return _reader.GetBoolean();
}
```

See also

- [System.Text.Json overview](#)
- [Instantiate JsonSerializerOptions instances](#)
- [Enable case-insensitive matching](#)
- [Customize property names and values](#)
- [Ignore properties](#)
- [Allow invalid JSON](#)
- [Handle overflow JSON or use JsonElement or JsonNode](#)
- [Preserve references and handle circular references](#)
- [Deserialize to immutable types and non-public accessors](#)
- [Polymorphic serialization](#)
- [Migrate from Newtonsoft.Json to System.Text.Json](#)
- [Customize character encoding](#)
- [Use DOM, Utf8JsonReader, and Utf8JsonWriter](#)
- [Write custom converters for JSON serialization](#)
- [DateTime and DateTimeOffset support](#)
- [How to use source generation](#)
- [Supported collection types](#)
- [System.Text.Json API reference](#)
- [System.Text.Json.Serialization API reference](#)

Compare Newtonsoft.Json to System.Text.Json, and migrate to System.Text.Json

9/20/2022 • 39 minutes to read • [Edit Online](#)

This article shows how to migrate from [Newtonsoft.Json](#) to [System.Text.Json](#).

The `System.Text.Json` namespace provides functionality for serializing to and deserializing from JavaScript Object Notation (JSON). The `System.Text.Json` library is included in the runtime for [.NET Core 3.1](#) and later versions. For other target frameworks, install the [System.Text.Json](#) NuGet package. The package supports:

- .NET Standard 2.0 and later versions
- .NET Framework 4.7.2 and later versions
- .NET Core 2.0, 2.1, and 2.2

`System.Text.Json` focuses primarily on performance, security, and standards compliance. It has some key differences in default behavior and doesn't aim to have feature parity with `Newtonsoft.Json`. For some scenarios, `System.Text.Json` currently has no built-in functionality, but there are recommended workarounds. For other scenarios, workarounds are impractical.

We're investing in adding the features that have most often been requested. If your application depends on a missing feature, consider [filing an issue](#) in the `dotnet/runtime` GitHub repository to find out if support for your scenario can be added. See [epic issue #43620](#) to find out what is already planned.

Most of this article is about how to use the `JsonSerializer` API, but it also includes guidance on how to use the `JsonDocument` (which represents the Document Object Model or DOM), `Utf8JsonReader`, and `Utf8JsonWriter` types.

In Visual Basic, you can't use `Utf8JsonReader`, which also means you can't write custom converters. Most of the workarounds presented here require that you write custom converters. You can write a custom converter in C# and register it in a Visual Basic project. For more information, see [Visual Basic support](#).

Table of differences between Newtonsoft.Json and System.Text.Json

The following table lists `Newtonsoft.Json` features and `System.Text.Json` equivalents. The equivalents fall into the following categories:

- Supported by built-in functionality. Getting similar behavior from `System.Text.Json` may require the use of an attribute or global option.
- Not supported, workaround is possible. The workarounds are [custom converters](#), which may not provide complete parity with `Newtonsoft.Json` functionality. For some of these, sample code is provided as examples. If you rely on these `Newtonsoft.Json` features, migration will require modifications to your .NET object models or other code changes.
- Not supported, workaround is not practical or possible. If you rely on these `Newtonsoft.Json` features, migration will not be possible without significant changes.

NEWTONSOFT.JSON FEATURE	SYSTEM.TEXT.JSON EQUIVALENT
Case-insensitive deserialization by default	✓ PropertyNameCaseInsensitive global setting
Camel-case property names	✓ PropertyNamingPolicy global setting

Minimal character escaping	✓ Strict character escaping, configurable
<code>NullValueHandling.Ignore</code> global setting	✓ <code>DefaultIgnoreCondition</code> global option
Allow comments	✓ <code>ReadCommentHandling</code> global setting
Allow trailing commas	✓ <code>AllowTrailingCommas</code> global setting
Custom converter registration	✓ Order of precedence differs
No maximum depth by default	✓ Default maximum depth 64, configurable
<code>PreserveReferencesHandling</code> global setting	✓ <code>ReferenceHandling</code> global setting
Serialize or deserialize numbers in quotes	✓ <code>NumberHandling</code> global setting, <code>[JsonNumberHandling]</code> attribute
Deserialize to immutable classes and structs	✓ <code>JsonConstructor</code> , C# 9 Records
Support for fields	✓ <code>IncludeFields</code> global setting, <code>[JsonInclude]</code> attribute
<code>DefaultValueHandling</code> global setting	✓ <code>DefaultIgnoreCondition</code> global setting
<code>NullValueHandling</code> setting on <code>[JsonProperty]</code>	✓ <code>JsonIgnore</code> attribute
<code>DefaultValueHandling</code> setting on <code>[JsonProperty]</code>	✓ <code>JsonIgnore</code> attribute
Deserialize <code>Dictionary</code> with non-string key	✓ Supported
Support for non-public property setters and getters	✓ <code>JsonInclude</code> attribute
<code>[JsonConstructor]</code> attribute	✓ <code>[JsonConstructor]</code> attribute
<code>ReferenceLoopHandling</code> global setting	✓ <code>ReferenceHandling</code> global setting
Callbacks	✓ Callbacks
NaN, Infinity, -Infinity	✓ Supported
Support for a broad range of types	⚠ Some types require custom converters
Polymorphic serialization	⚠ Not supported, workaround, sample
Polymorphic deserialization	⚠ Not supported, workaround, sample
Deserialize inferred type to <code>object</code> properties	⚠ Not supported, workaround, sample

NEWTONSOFT.JSON FEATURE	SYSTEM.TEXT.JSON EQUIVALENT
Deserialize JSON <code>null</code> literal to non-nullable value types	⚠️ Not supported, workaround, sample
<code>Required</code> setting on <code>[JsonProperty]</code> attribute	⚠️ Not supported, workaround, sample
<code>DefaultContractResolver</code> to ignore properties	⚠️ Not supported, workaround, sample
<code>DateTimeZoneHandling</code> , <code>DateFormatString</code> settings	⚠️ Not supported, workaround, sample
<code>JsonConvert.PopulateObject</code> method	⚠️ Not supported, workaround
<code>ObjectCreationHandling</code> global setting	⚠️ Not supported, workaround
Add to collections without setters	⚠️ Not supported, workaround
Snake-case property names	⚠️ Not supported, workaround
Support for <code>System.Runtime.Serialization</code> attributes	✗ Not supported
<code>MissingMemberHandling</code> global setting	✗ Not supported
Allow property names without quotes	✗ Not supported
Allow single quotes around string values	✗ Not supported
Allow non-string JSON values for string properties	✗ Not supported
<code>TypeNameHandling.All</code> global setting	✗ Not supported
Support for <code>JsonPath</code> queries	✗ Not supported
Configurable limits	✗ Not supported

NEWTONSOFT.JSON FEATURE	SYSTEM.TEXT.JSON EQUIVALENT
Case-insensitive deserialization by default	✓ <code>PropertyNameCaseInsensitive</code> global setting
Camel-case property names	✓ <code>PropertyNameNamingPolicy</code> global setting
Minimal character escaping	✓ Strict character escaping, configurable
<code>NullValueHandling.Ignore</code> global setting	✓ <code>DefaultIgnoreCondition</code> global option
Allow comments	✓ <code>ReadCommentHandling</code> global setting
Allow trailing commas	✓ <code>AllowTrailingCommas</code> global setting
Custom converter registration	✓ Order of precedence differs
No maximum depth by default	✓ Default maximum depth 64, configurable

Newtonsoft.Json Feature	System.Text.Json Equivalent
<code>PreserveReferencesHandling</code> global setting	✓ ReferenceHandling global setting
Serialize or deserialize numbers in quotes	✓ NumberHandling global setting, [JsonNumberHandling] attribute
Deserialize to immutable classes and structs	✓ JsonConstructor , C# 9 Records
Support for fields	✓ IncludeFields global setting, [JsonInclude] attribute
<code>DefaultValueHandling</code> global setting	✓ DefaultIgnoreCondition global setting
<code>NullValueHandling</code> setting on <code>[JsonProperty]</code>	✓ JsonIgnore attribute
<code>DefaultValueHandling</code> setting on <code>[JsonProperty]</code>	✓ JsonIgnore attribute
Deserialize <code>Dictionary</code> with non-string key	✓ Supported
Support for non-public property setters and getters	✓ JsonInclude attribute
<code>[JsonConstructor]</code> attribute	✓ [JsonConstructor] attribute
NaN, Infinity, -Infinity	✓ Supported
Support for a broad range of types	⚠ Some types require custom converters
Polymorphic serialization	⚠ Not supported, workaround, sample
Polymorphic deserialization	⚠ Not supported, workaround, sample
Deserialize inferred type to <code>object</code> properties	⚠ Not supported, workaround, sample
Deserialize JSON <code>null</code> literal to non-nullable value types	⚠ Not supported, workaround, sample
<code>Required</code> setting on <code>[JsonProperty]</code> attribute	⚠ Not supported, workaround, sample
<code>DefaultContractResolver</code> to ignore properties	⚠ Not supported, workaround, sample
<code>DateTimeZoneHandling</code> , <code>DateFormatString</code> settings	⚠ Not supported, workaround, sample
Callbacks	⚠ Not supported, workaround, sample
<code>JsonConvert.PopulateObject</code> method	⚠ Not supported, workaround
<code>ObjectCreationHandling</code> global setting	⚠ Not supported, workaround
Add to collections without setters	⚠ Not supported, workaround
Snake-case property names	⚠ Not supported, workaround

Newtonsoft.Json Feature	System.Text.Json Equivalent
<code>ReferenceLoopHandling</code> global setting	✗ Not supported
Support for <code>System.Runtime.Serialization</code> attributes	✗ Not supported
<code>MissingMemberHandling</code> global setting	✗ Not supported
Allow property names without quotes	✗ Not supported
Allow single quotes around string values	✗ Not supported
Allow non-string JSON values for string properties	✗ Not supported
<code>TypeNameHandling.All</code> global setting	✗ Not supported
Support for <code>JsonPath</code> queries	✗ Not supported
Configurable limits	✗ Not supported

Newtonsoft.Json Feature	System.Text.Json Equivalent
Case-insensitive deserialization by default	✓ <code>PropertyNameCaseInsensitive</code> global setting
Camel-case property names	✓ <code>PropertyNamePolicy</code> global setting
Minimal character escaping	✓ Strict character escaping, configurable
<code>NullValueHandling.Ignore</code> global setting	✓ <code>IgnoreNullValues</code> global option
Allow comments	✓ <code>ReadCommentHandling</code> global setting
Allow trailing commas	✓ <code>AllowTrailingCommas</code> global setting
Custom converter registration	✓ Order of precedence differs
No maximum depth by default	✓ Default maximum depth 64, configurable
Support for a broad range of types	⚠ Some types require custom converters
Deserialize strings as numbers	⚠ Not supported, workaround, sample
Deserialize <code>Dictionary</code> with non-string key	⚠ Not supported, workaround, sample
Polymorphic serialization	⚠ Not supported, workaround, sample
Polymorphic deserialization	⚠ Not supported, workaround, sample
Deserialize inferred type to <code>object</code> properties	⚠ Not supported, workaround, sample
Deserialize JSON <code>null</code> literal to non-nullable value types	⚠ Not supported, workaround, sample

Newtonsoft.Json Feature	System.Text.Json Equivalent
Deserialize to immutable classes and structs	⚠️ Not supported, workaround, sample
<code>[JsonConstructor]</code> attribute	⚠️ Not supported, workaround, sample
<code>Required</code> setting on <code>[JsonProperty]</code> attribute	⚠️ Not supported, workaround, sample
<code>NullValueHandling</code> setting on <code>[JsonProperty]</code> attribute	⚠️ Not supported, workaround, sample
<code>DefaultValueHandling</code> setting on <code>[JsonProperty]</code> attribute	⚠️ Not supported, workaround, sample
<code>DefaultValueHandling</code> global setting	⚠️ Not supported, workaround, sample
<code>DefaultContractResolver</code> to ignore properties	⚠️ Not supported, workaround, sample
<code>DateTimeZoneHandling</code> , <code>DateFormatString</code> settings	⚠️ Not supported, workaround, sample
Callbacks	⚠️ Not supported, workaround, sample
Support for public and non-public fields	⚠️ Not supported, workaround
Support for non-public property setters and getters	⚠️ Not supported, workaround
<code>JsonConvert.PopulateObject</code> method	⚠️ Not supported, workaround
<code>ObjectCreationHandling</code> global setting	⚠️ Not supported, workaround
Add to collections without setters	⚠️ Not supported, workaround
Snake-case property names	⚠️ Not supported, workaround
NaN, Infinity, -Infinity	⚠️ Not supported, workaround
<code>PreserveReferencesHandling</code> global setting	✗ Not supported
<code>ReferenceLoopHandling</code> global setting	✗ Not supported
Support for <code>System.Runtime.Serialization</code> attributes	✗ Not supported
<code>MissingMemberHandling</code> global setting	✗ Not supported
Allow property names without quotes	✗ Not supported
Allow single quotes around string values	✗ Not supported
Allow non-string JSON values for string properties	✗ Not supported
<code>TypeNameHandling.All</code> global setting	✗ Not supported

Newtonsoft.Json Feature	System.Text.Json Equivalent
Support for <code>JsonPath</code> queries	✗ Not supported
Configurable limits	✗ Not supported

This is not an exhaustive list of `Newtonsoft.Json` features. The list includes many of the scenarios that have been requested in [GitHub issues](#) or [StackOverflow](#) posts. If you implement a workaround for one of the scenarios listed here that doesn't currently have sample code, and if you want to share your solution, select [This page](#) in the [Feedback](#) section at the bottom of this page. That creates an issue in this documentation's GitHub repo and lists it in the [Feedback](#) section on this page too.

Differences in default JsonSerializer behavior compared to Newtonsoft.Json

`System.Text.Json` is strict by default and avoids any guessing or interpretation on the caller's behalf, emphasizing deterministic behavior. The library is intentionally designed this way for performance and security.

`Newtonsoft.Json` is flexible by default. This fundamental difference in design is behind many of the following specific differences in default behavior.

Case-insensitive deserialization

During deserialization, `Newtonsoft.Json` does case-insensitive property name matching by default. The `System.Text.Json` default is case-sensitive, which gives better performance since it's doing an exact match. For information about how to do case-insensitive matching, see [Case-insensitive property matching](#).

If you're using `System.Text.Json` indirectly by using ASP.NET Core, you don't need to do anything to get behavior like `Newtonsoft.Json`. ASP.NET Core specifies the settings for [camel-casing property names](#) and case-insensitive matching when it uses `System.Text.Json`.

ASP.NET Core also enables deserializing [quoted numbers](#) by default.

Minimal character escaping

During serialization, `Newtonsoft.Json` is relatively permissive about letting characters through without escaping them. That is, it doesn't replace them with `\uxxxx` where `xxxx` is the character's code point. Where it does escape them, it does so by emitting a `\` before the character (for example, `"` becomes `\"`). `System.Text.Json` escapes more characters by default to provide defense-in-depth protections against cross-site scripting (XSS) or information-disclosure attacks and does so by using the six-character sequence. `System.Text.Json` escapes all non-ASCII characters by default, so you don't need to do anything if you're using `StringEscapeHandling.EscapeNonAscii` in `Newtonsoft.Json`. `System.Text.Json` also escapes HTML-sensitive characters, by default. For information about how to override the default `System.Text.Json` behavior, see [Customize character encoding](#).

Comments

During deserialization, `Newtonsoft.Json` ignores comments in the JSON by default. The `System.Text.Json` default is to throw exceptions for comments because the [RFC 8259](#) specification doesn't include them. For information about how to allow comments, see [Allow comments and trailing commas](#).

Trailing commas

During deserialization, `Newtonsoft.Json` ignores trailing commas by default. It also ignores multiple trailing commas (for example, `[{"Color": "Red"}, {"Color": "Green"}, ,]`). The `System.Text.Json` default is to throw exceptions for trailing commas because the [RFC 8259](#) specification doesn't allow them. For information about how to make `System.Text.Json` accept them, see [Allow comments and trailing commas](#). There's no way to allow multiple trailing commas.

Converter registration precedence

The `Newtonsoft.Json` registration precedence for custom converters is as follows:

- Attribute on property
- Attribute on type
- `Converters` collection

This order means that a custom converter in the `Converters` collection is overridden by a converter that is registered by applying an attribute at the type level. Both of those registrations are overridden by an attribute at the property level.

The `System.Text.Json` registration precedence for custom converters is different:

- Attribute on property
- `Converters` collection
- Attribute on type

The difference here is that a custom converter in the `Converters` collection overrides an attribute at the type level. The intention behind this order of precedence is to make run-time changes override design-time choices. There's no way to change the precedence.

For more information about custom converter registration, see [Register a custom converter](#).

Maximum depth

The latest version of `Newtonsoft.Json` has a maximum depth limit of 64 by default. `System.Text.Json` also has a default limit of 64, and it's configurable by setting `JsonSerializerOptions.MaxDepth`.

If you're using `System.Text.Json` indirectly by using ASP.NET Core, the default maximum depth limit is 32. The default value is the same as for model binding and is set in the `JsonOptions` class.

JSON strings (property names and string values)

During deserialization, `Newtonsoft.Json` accepts property names surrounded by double quotes, single quotes, or without quotes. It accepts string values surrounded by double quotes or single quotes. For example, `Newtonsoft.Json` accepts the following JSON:

```
{  
    "name1": "value",  
    'name2': "value",  
    name3: 'value'  
}
```

`System.Text.Json` only accepts property names and string values in double quotes because that format is required by the [RFC 8259](#) specification and is the only format considered valid JSON.

A value enclosed in single quotes results in a `JsonException` with the following message:

```
''' is an invalid start of a value.
```

Non-string values for string properties

`Newtonsoft.Json` accepts non-string values, such as a number or the literals `true` and `false`, for deserialization to properties of type `string`. Here's an example of JSON that `Newtonsoft.Json` successfully deserializes to the following class:

```
{  
    "String1": 1,  
    "String2": true,  
    "String3": false  
}
```

```
public class ExampleClass  
{  
    public string String1 { get; set; }  
    public string String2 { get; set; }  
    public string String3 { get; set; }  
}
```

`System.Text.Json` doesn't deserialize non-string values into string properties. A non-string value received for a string field results in a [JsonException](#) with the following message:

```
The JSON value could not be converted to System.String.
```

Scenarios using JsonSerializer

Some of the following scenarios aren't supported by built-in functionality, but workarounds are possible. The workarounds are [custom converters](#), which may not provide complete parity with `Newtonsoft.Json` functionality. For some of these, sample code is provided as examples. If you rely on these `Newtonsoft.Json` features, migration will require modifications to your .NET object models or other code changes.

For some of the following scenarios, workarounds are not practical or possible. If you rely on these `Newtonsoft.Json` features, migration will not be possible without significant changes.

Allow or write numbers in quotes

`Newtonsoft.Json` can serialize or deserialize numbers represented by JSON strings (surrounded by quotes). For example, it can accept: `{"DegreesCelsius":"23"}` instead of `{"DegreesCelsius":23}`. To enable that behavior in `System.Text.Json`, set `JsonSerializerOptions.NumberHandling` to `WriteAsString` or `AllowReadingFromString`, or use the `[JsonNumberHandling]` attribute.

If you're using `System.Text.Json` indirectly by using ASP.NET Core, you don't need to do anything to get behavior like `Newtonsoft.Json`. ASP.NET Core specifies [web defaults](#) when it uses `System.Text.Json`, and web defaults allow quoted numbers.

For more information, see [Allow or write numbers in quotes](#).

`Newtonsoft.Json` can serialize or deserialize numbers represented by JSON strings (surrounded by quotes). For example, it can accept: `{"DegreesCelsius":"23"}` instead of `{"DegreesCelsius":23}`. To enable that behavior in `System.Text.Json` in .NET Core 3.1, implement a custom converter like the following example. The converter handles properties defined as `long`:

- It serializes them as JSON strings.
- It accepts JSON numbers and numbers within quotes while deserializing.

```

using System.Buffers;
using System.Buffers.Text;
using System.Text.Json;
using System.Text.Json.Serialization;

namespace SystemTextJsonSamples
{
    public class LongToStringConverter : JsonConverter<long>
    {
        public override long Read(
            ref Utf8JsonReader reader, Type type, JsonSerializerOptions options)
        {
            if (reader.TokenType == JsonTokenType.String)
            {
                ReadOnlySpan<byte> span =
                    reader.HasValueSequence ? reader.ValueSequence.ToArray() : reader.ValueSpan;

                if (Utf8Parser.TryParse(span, out long number, out int bytesConsumed) &&
                    span.Length == bytesConsumed)
                {
                    return number;
                }

                if (long.TryParse(reader.GetString(), out number))
                {
                    return number;
                }
            }

            return reader.GetInt64();
        }

        public override void Write(
            Utf8JsonWriter writer, long longValue, JsonSerializerOptions options) =>
            writer.WriteStringValue(longValue.ToString());
    }
}

```

Register this custom converter by [using an attribute](#) on individual `long` properties or by [adding the converter](#) to the [Converters](#) collection.

Specify constructor to use when deserializing

The `Newtonsoft.Json [JsonConstructor]` attribute lets you specify which constructor to call when deserializing to a POCO.

`System.Text.Json` also has a `[JsonConstructor]` attribute. For more information, see [Immutable types and Records](#).

`System.Text.Json` in .NET Core 3.1 supports only parameterless constructors. As a workaround, you can call whichever constructor you need in a custom converter. See the example for [Deserialize to immutable classes and structs](#).

Conditionally ignore a property

`Newtonsoft.Json` has several ways to conditionally ignore a property on serialization or deserialization:

- `DefaultContractResolver` lets you select properties to include or ignore, based on arbitrary criteria.
- The `NullValueHandling` and `DefaultValueHandling` settings on `JsonSerializerSettings` let you specify that all null-value or default-value properties should be ignored.
- The `NullValueHandling` and `DefaultValueHandling` settings on the `[JsonProperty]` attribute let you specify individual properties that should be ignored when set to null or the default value.

`System.Text.Json` provides the following ways to ignore properties or fields while serializing:

- The `[JsonIgnore]` attribute on a property causes the property to be omitted from the JSON during serialization.
- The `IgnoreReadOnlyProperties` global option lets you ignore all read-only properties.
- If you're `Including` fields, the `JsonSerializerOptions.IgnoreReadOnlyFields` global option lets you ignore all read-only fields.
- The `DefaultIgnoreCondition` global option lets you [ignore all value type properties that have default values](#), or [ignore all reference type properties that have null values](#).

`System.Text.Json` in .NET Core 3.1 provides the following ways to ignore properties while serializing:

- The `[JsonIgnore]` attribute on a property causes the property to be omitted from the JSON during serialization.
- The `IgnoreNullValues` global option lets you ignore all null-value properties. `IgnoreNullValues` is deprecated in .NET 5 and later versions, so it isn't shown by IntelliSense. For the current way to ignore null values, see [how to ignore all null-value properties in .NET 5 and later](#).
- The `IgnoreReadOnlyProperties` global option lets you ignore all read-only properties.

These options **don't** let you:

- Ignore selected properties based on arbitrary criteria evaluated at run time.
- Ignore all properties that have the default value for the type.
- Ignore selected properties that have the default value for the type.
- Ignore selected properties if their value is null.
- Ignore selected properties based on arbitrary criteria evaluated at run time.

For that functionality, you can write a custom converter. Here's a sample POCO and a custom converter for it that illustrates this approach:

```
public class WeatherForecast
{
    public DateTimeOffset Date { get; set; }
    public int TemperatureCelsius { get; set; }
    public string? Summary { get; set; }
}
```

```

using System.Text.Json;
using System.Text.Json.Serialization;

namespace SystemTextJsonSamples
{
    public class WeatherForecastRuntimeIgnoreConverter : JsonConverter<WeatherForecast>
    {
        public override WeatherForecast Read(
            ref Utf8JsonReader reader,
            Type typeToConvert,
            JsonSerializerOptions options)
        {
            if (reader.TokenType != JsonTokenType.StartObject)
            {
                throw new JsonException();
            }

            var wf = new WeatherForecast();

            while (reader.Read())
            {
                if (reader.TokenType == JsonTokenType.EndObject)
                {
                    return wf;
                }

                if (reader.TokenType == JsonTokenType.PropertyName)
                {
                    string propertyName = reader.GetString()!;
                    reader.Read();
                    switch (propertyName)
                    {
                        case "Date":
                            DateTimeOffset date = reader.GetDateTimeOffset();
                            wf.Date = date;
                            break;
                        case "TemperatureCelsius":
                            int temperatureCelsius = reader.GetInt32();
                            wf.TemperatureCelsius = temperatureCelsius;
                            break;
                        case "Summary":
                            string summary = reader.GetString()?;
                            wf.Summary = string.IsNullOrWhiteSpace(summary) ? "N/A" : summary;
                            break;
                    }
                }
            }

            throw new JsonException();
        }

        public override void Write(Utf8JsonWriter writer, WeatherForecast wf, JsonSerializerOptions options)
        {
            writer.WriteStartObject();

            writer.WriteString("Date", wf.Date);
            writer.WriteNumber("TemperatureCelsius", wf.TemperatureCelsius);
            if (!string.IsNullOrWhiteSpace(wf.Summary) && wf.Summary != "N/A")
            {
                writer.WriteString("Summary", wf.Summary);
            }

            writer.WriteEndObject();
        }
    }
}

```

The converter causes the `Summary` property to be omitted from serialization if its value is null, an empty string, or "N/A".

Register this custom converter by [using an attribute on the class](#) or by [adding the converter](#) to the `Converters` collection.

This approach requires additional logic if:

- The POCO includes complex properties.
- You need to handle attributes such as `[JsonIgnore]` or options such as custom encoders.

Public and non-public fields

`Newtonsoft.Json` can serialize and deserialize fields as well as properties.

In `System.Text.Json`, use the `JsonSerializerOptions.IncludeFields` global setting or the `[JsonInclude]` attribute to include public fields when serializing or deserializing. For an example, see [Include fields](#).

`System.Text.Json` in .NET Core 3.1 only works with public properties. Custom converters can provide this functionality.

Preserve object references and handle loops

By default, `Newtonsoft.Json` serializes by value. For example, if an object contains two properties that contain a reference to the same `Person` object, the values of that `Person` object's properties are duplicated in the JSON.

`Newtonsoft.Json` has a `PreserveReferencesHandling` setting on `JsonSerializerSettings` that lets you serialize by reference:

- An identifier metadata is added to the JSON created for the first `Person` object.
- The JSON that is created for the second `Person` object contains a reference to that identifier instead of property values.

`Newtonsoft.Json` also has a `ReferenceLoopHandling` setting that lets you ignore circular references rather than throw an exception.

To preserve references and handle circular references in `System.Text.Json`, set

`JsonSerializerOptions.ReferenceHandler` to `Preserve`. The `ReferenceHandler.Preserve` setting is equivalent to `PreserveReferencesHandling = PreserveReferencesHandling.All` in `Newtonsoft.Json`.

The `ReferenceHandler.IgnoreCycles` option has behavior similar to `Newtonsoft.Json ReferenceLoopHandling.Ignore`. One difference is that the `System.Text.Json` implementation replaces reference loops with the `null` JSON token instead of ignoring the object reference. For more information, see [Ignore circular references](#).

Like the `Newtonsoft.Json ReferenceResolver`, the `System.Text.Json.Serialization.ReferenceResolver` class defines the behavior of preserving references on serialization and deserialization. Create a derived class to specify custom behavior. For an example, see [GuidReferenceResolver](#).

Some related `Newtonsoft.Json` features are not supported:

- [JsonPropertyAttribute.IsReference](#)
- [JsonPropertyAttribute.ReferenceLoopHandling](#)

For more information, see [Preserve references and handle circular references](#).

- [JsonPropertyAttribute.IsReference](#)
- [JsonPropertyAttribute.ReferenceLoopHandling](#)
- [JsonSerializerSettings.ReferenceLoopHandling](#)

For more information, see [Preserve references and handle circular references](#).

`System.Text.Json` in .NET Core 3.1 only supports serialization by value and throws an exception for circular references.

Dictionary with non-string key

Both `Newtonsoft.Json` and `System.Text.Json` support collections of type `Dictionary<TKey, TValue>`. However, in `System.Text.Json`, `TKey` must be a primitive type, not a custom type. For more information, see [Supported key types](#).

Caution

Deserializing to a `Dictionary<TKey, TValue>` where `TKey` is typed as anything other than `string` could introduce a security vulnerability in the consuming application. For more information, see [dotnet/runtime#4761](#).

`Newtonsoft.Json` supports collections of type `Dictionary<TKey, TValue>`. The built-in support for dictionary collections in `System.Text.Json` in .NET Core 3.1 is limited to `Dictionary<string, TValue>`. That is, the key must be a string.

To support a dictionary with an integer or some other type as the key in .NET Core 3.1, create a converter like the example in [How to write custom converters](#).

Types without built-in support

`System.Text.Json` doesn't provide built-in support for the following types:

- `DataTable` and related types (for more information, see [Supported collection types](#))
- F# types, such as [discriminated unions](#), [Record types](#) and [anonymous record types](#) are treated as immutable POCOs and thus are supported.
- `TimeSpan`
- F# types, such as [discriminated unions](#), [record types](#), and [anonymous record types](#).
- `TimeSpan`
- `ExpandoObject`
- `TimeZoneInfo`
- `BigInteger`
- `DBNull`
- `Type`
- `ValueTuple` and its associated generic types

Custom converters can be implemented for types that don't have built-in support.

Polymorphic serialization

`Newtonsoft.Json` automatically does polymorphic serialization. For information about the limited polymorphic serialization capabilities of `System.Text.Json`, see [Serialize properties of derived classes](#).

The workaround described there is to define properties that may contain derived classes as type `object`. If that isn't possible, another option is to create a converter with a `Write` method for the whole inheritance type hierarchy like the example in [How to write custom converters](#).

Polymorphic deserialization

`Newtonsoft.Json` has a `TypeNameHandling` setting that adds type name metadata to the JSON while serializing. It uses the metadata while deserializing to do polymorphic deserialization. `System.Text.Json` can do a limited range of [polymorphic serialization](#) but not polymorphic deserialization.

To support polymorphic deserialization, create a converter like the example in [How to write custom converters](#).

Deserialization of object properties

When `Newtonsoft.Json` deserializes to `Object`, it:

- Infers the type of primitive values in the JSON payload (other than `null`) and returns the stored `string`, `long`, `double`, `boolean`, or `DateTime` as a boxed object. *Primitive values* are single JSON values such as a JSON number, string, `true`, `false`, or `null`.
- Returns a `JObject` or `JArray` for complex values in the JSON payload. *Complex values* are collections of JSON key-value pairs within braces (`{}`) or lists of values within brackets (`[]`). The properties and values within the braces or brackets can have additional properties or values.
- Returns a null reference when the payload has the `null` JSON literal.

`System.Text.Json` stores a boxed `JsonElement` for both primitive and complex values whenever deserializing to `Object`, for example:

- An `object` property.
- An `object` dictionary value.
- An `object` array value.
- A root `object`.

However, `System.Text.Json` treats `null` the same as `Newtonsoft.Json` and returns a null reference when the payload has the `null` JSON literal in it.

To implement type inference for `object` properties, create a converter like the example in [How to write custom converters](#).

Deserialize null to non-nullable type

`Newtonsoft.Json` doesn't throw an exception in the following scenario:

- `NullValueHandling` is set to `Ignore`, and
- During deserialization, the JSON contains a null value for a non-nullable value type.

In the same scenario, `System.Text.Json` does throw an exception. (The corresponding null-handling setting in `System.Text.Json` is `JsonSerializerOptions.IgnoreNullValues = true`.)

If you own the target type, the best workaround is to make the property in question nullable (for example, change `int` to `int?`).

Another workaround is to make a converter for the type, such as the following example that handles null values for `DateTimeOffset` types:

```

using System.Text.Json;
using System.Text.Json.Serialization;

namespace SystemTextJsonSamples
{
    public class DateTimeOffsetNullHandlingConverter : JsonConverter<DateTimeOffset>
    {
        public override DateTimeOffset Read(
            ref Utf8JsonReader reader,
            Type typeToConvert,
            JsonSerializerOptions options) =>
            reader.TokenType == JsonTokenType.Null
                ? default
                : reader.GetDateTimeOffset();

        public override void Write(
            Utf8JsonWriter writer,
            DateTimeOffset dateTimeValue,
            JsonSerializerOptions options) =>
            writer.WriteStringValue(dateTimeValue);
    }
}

```

Register this custom converter by [using an attribute on the property](#) or by [adding the converter](#) to the [Converters](#) collection.

Note: The preceding converter **handles null values differently** than [Newtonsoft.Json](#) does for POCOs that specify default values. For example, suppose the following code represents your target object:

```

public class WeatherForecastWithDefault
{
    public WeatherForecastWithDefault()
    {
        Date = DateTimeOffset.Parse("2001-01-01");
        Summary = "No summary";
    }
    public DateTimeOffset Date { get; set; }
    public int TemperatureCelsius { get; set; }
    public string Summary { get; set; }
}

```

And suppose the following JSON is deserialized by using the preceding converter:

```
{
    "Date": null,
    "TemperatureCelsius": 25,
    "Summary": null
}
```

After deserialization, the `Date` property has 1/1/0001 (`default(DateTimeOffset)`), that is, the value set in the constructor is overwritten. Given the same POCO and JSON, [Newtonsoft.Json](#) deserialization would leave 1/1/2001 in the `Date` property.

Deserialize to immutable classes and structs

[Newtonsoft.Json](#) can deserialize to immutable classes and structs because it can use constructors that have parameters.

In [System.Text.Json](#), use the [\[JsonConstructor\]](#) attribute to specify use of a parameterized constructor. Records in C# 9 are also immutable and are supported as deserialization targets. For more information, see [Immutable types and Records](#).

[System.Text.Json](#) in .NET Core 3.1 supports only public parameterless constructors. As a workaround, you can call a constructor with parameters in a custom converter.

Here's an immutable struct with multiple constructor parameters:

```
public readonly struct ImmutablePoint
{
    public ImmutablePoint(int x, int y)
    {
        X = x;
        Y = y;
    }

    public int X { get; }
    public int Y { get; }
}
```

And here's a converter that serializes and deserializes this struct:

```
using System.Diagnostics;
using System.Text.Json;
using System.Text.Json.Serialization;

namespace SystemTextJsonSamples
{
    public class ImmutablePointConverter : JsonConverter<ImmutablePoint>
    {
        private readonly JsonEncodedText _xName = JsonEncodedText.Encode("X");
        private readonly JsonEncodedText _yName = JsonEncodedText.Encode("Y");

        private readonly JsonConverter<int> _intConverter;

        public ImmutablePointConverter(JsonSerializerOptions options) =>
            _intConverter = options?.GetConverter(typeof(int)) is JsonConverter<int> intConverter
                ? intConverter
                : throw new InvalidOperationException();

        public override ImmutablePoint Read(
            ref Utf8JsonReader reader,
            Type typeToConvert,
            JsonSerializerOptions options)
        {
            if (reader.TokenType != JsonTokenType.StartObject)
            {
                throw new JsonException();
            };

            int? x = default;
            int? y = default;

            // Get the first property.
            reader.Read();
            if (reader.TokenType != JsonTokenType.PropertyName)
            {
                throw new JsonException();
            }

            if (reader.ValueTextEquals(_xName.EncodedUtf8Bytes))
            {
                x = ReadProperty(ref reader, options);
            }
            else if (reader.ValueTextEquals(_yName.EncodedUtf8Bytes))
            {
                y = ReadProperty(ref reader, options);
            }
            else
            {
                throw new JsonException();
            }
        }

        public void Write(ImmutablePoint value, Utf8JsonWriter writer, JsonSerializerOptions options)
        {
            writer.WriteStartObject();
            writer.WriteString(_xName, value.X.ToString());
            writer.WriteString(_yName, value.Y.ToString());
            writer.WriteEndObject();
        }
    }
}
```

```

        }
        throw new JsonException();
    }

    // Get the second property.
    reader.Read();
    if (reader.TokenType != JsonTokenType.PropertyName)
    {
        throw new JsonException();
    }

    if (x.HasValue && reader.ValueTextEquals(_yName.EncodedUtf8Bytes))
    {
        y = ReadProperty(ref reader, options);
    }
    else if (y.HasValue && reader.ValueTextEquals(_xName.EncodedUtf8Bytes))
    {
        x = ReadProperty(ref reader, options);
    }
    else
    {
        throw new JsonException();
    }

    reader.Read();

    if (reader.TokenType != JsonTokenType.EndObject)
    {
        throw new JsonException();
    }

    return new ImmutablePoint(x.GetValueOrDefault(), y.GetValueOrDefault());
}

private int ReadProperty(ref Utf8JsonReader reader, JsonSerializerOptions options)
{
    Debug.Assert(reader.TokenType == JsonTokenType.PropertyName);

    reader.Read();
    return _intConverter.Read(ref reader, typeof(int), options);
}

private void WriteProperty(Utf8JsonWriter writer, JsonEncodedText name, int intValue,
JsonSerializerOptions options)
{
    writer.WritePropertyName(name);
    _intConverter.Write(writer, intValue, options);
}

public override void Write(
    Utf8JsonWriter writer,
    ImmutablePoint point,
    JsonSerializerOptions options)
{
    writer.WriteStartObject();
    WriteProperty(writer, _xName, point.X, options);
    WriteProperty(writer, _yName, point.Y, options);
    writer.WriteEndObject();
}
}
}

```

Register this custom converter by [adding the converter](#) to the [Converters](#) collection.

For an example of a similar converter that handles open generic properties, see the [built-in converter for key-value pairs](#).

Required properties

In `Newtonsoft.Json`, you specify that a property is required by setting `Required` on the `[JsonProperty]` attribute. `Newtonsoft.Json` throws an exception if no value is received in the JSON for a property marked as required.

`System.Text.Json` doesn't throw an exception if no value is received for one of the properties of the target type. For example, if you have a `WeatherForecast` class:

```
public class WeatherForecast
{
    public DateTimeOffset Date { get; set; }
    public int TemperatureCelsius { get; set; }
    public string? Summary { get; set; }
}
```

The following JSON is deserialized without error:

```
{
    "TemperatureCelsius": 25,
    "Summary": "Hot"
}
```

To make deserialization fail if no `Date` property is in the JSON, choose one of the following options:

- Implement a custom converter.
- Implement an [OnDeserialized callback \(.NET 6 and later\)](#).

The following sample converter code throws an exception if the `Date` property isn't set after deserialization is complete:

```
using System.Text.Json;
using System.Text.Json.Serialization;

namespace SystemTextJsonSamples
{
    public class WeatherForecastRequiredPropertyConverter : JsonConverter<WeatherForecast>
    {
        public override WeatherForecast Read(
            ref Utf8JsonReader reader,
            Type type,
            JsonSerializerOptions options)
        {
            // Don't pass in options when recursively calling Deserialize.
            WeatherForecast forecast = JsonSerializer.Deserialize<WeatherForecast>(ref reader)!;

            // Check for required fields set by values in JSON
            return forecast!.Date == default
                ? throw new JsonException("Required property not received in the JSON")
                : forecast;
        }

        public override void Write(
            Utf8JsonWriter writer,
            WeatherForecast forecast, JsonSerializerOptions options)
        {
            // Don't pass in options when recursively calling Serialize.
            JsonSerializer.Serialize(writer, forecast);
        }
    }
}
```

Register this custom converter by [adding the converter](#) to the [JsonSerializerOptions.Converters](#) collection.

This pattern of recursively calling the converter requires that you register the converter by using [JsonSerializerOptions](#), not by using an attribute. If you register the converter by using an attribute, the custom converter recursively calls into itself. The result is an infinite loop that ends in a stack overflow exception.

When you register the converter by using the options object, avoid an infinite loop by not passing in the options object when recursively calling [Serialize](#) or [Deserialize](#). The options object contains the [Converters](#) collection. If you pass it in to [Serialize](#) or [Deserialize](#), the custom converter calls into itself, making an infinite loop that results in a stack overflow exception. If the default options are not feasible, create a new instance of the options with the settings that you need. This approach will be slow since each new instance caches independently.

There is an alternative pattern that can use [JsonConverterAttribute](#) registration on the class to be converted. In this approach, the converter code calls [Serialize](#) or [Deserialize](#) on a class that derives from the class to be converted. The derived class doesn't have a [JsonConverterAttribute](#) applied to it. In the following example of this alternative:

- `WeatherForecastWithRequiredPropertyConverterAttribute` is the class to be deserialized and has the [JsonConverterAttribute](#) applied to it.
- `WeatherForecastWithoutRequiredPropertyConverterAttribute` is the derived class that doesn't have the converter attribute.
- The code in the converter calls [Serialize](#) and [Deserialize](#) on `WeatherForecastWithoutRequiredPropertyConverterAttribute` to avoid an infinite loop. There is a performance cost to this approach on serialization due to an extra object instantiation and copying of property values.

Here are the `WeatherForecast*` types:

```
[JsonConverter(typeof(WeatherForecastRequiredPropertyConverterForAttributeRegistration))]
public class WeatherForecastWithRequiredPropertyConverterAttribute
{
    public DateTimeOffset Date { get; set; }
    public int TemperatureCelsius { get; set; }
    public string? Summary { get; set; }
}

public class WeatherForecastWithoutRequiredPropertyConverterAttribute :
    WeatherForecastWithRequiredPropertyConverterAttribute
{
}
```

And here is the converter:

```

using System.Text.Json;
using System.Text.Json.Serialization;

namespace SystemTextJsonSamples
{
    public class WeatherForecastRequiredPropertyConverterForAttributeRegistration :
        JsonConverter<WeatherForecastWithRequiredPropertyConverterAttribute>
    {
        public override WeatherForecastWithRequiredPropertyConverterAttribute Read(
            ref Utf8JsonReader reader,
            Type type,
            JsonSerializerOptions options)
        {
            // OK to pass in options when recursively calling Deserialize.
            WeatherForecastWithRequiredPropertyConverterAttribute forecast =
                JsonSerializer.Deserialize<WeatherForecastWithoutRequiredPropertyConverterAttribute>(
                    ref reader,
                    options)!;

            // Check for required fields set by values in JSON.
            return forecast!.Date == default
                ? throw new JsonException("Required property not received in the JSON")
                : forecast;
        }

        public override void Write(
            Utf8JsonWriter writer,
            WeatherForecastWithRequiredPropertyConverterAttribute forecast,
            JsonSerializerOptions options)
        {
            var weatherForecastWithoutConverterAttributeOnClass =
                new WeatherForecastWithoutRequiredPropertyConverterAttribute
                {
                    Date = forecast.Date,
                    TemperatureCelsius = forecast.TemperatureCelsius,
                    Summary = forecast.Summary
                };

            // OK to pass in options when recursively calling Serialize.
            JsonSerializer.Serialize(
                writer,
                weatherForecastWithoutConverterAttributeOnClass,
                options);
        }
    }
}

```

The required properties converter would require additional logic if you need to handle attributes such as [\[JsonIgnore\]](#) or different options, such as custom encoders. Also, the example code doesn't handle properties for which a default value is set in the constructor. And this approach doesn't differentiate between the following scenarios:

- A property is missing from the JSON.
- A property for a non-nullable type is present in the JSON, but the value is the default for the type, such as zero for an `int`.
- A property for a nullable value type is present in the JSON, but the value is null.

Note: If you're using `System.Text.Json` from an ASP.NET Core controller, you might be able to use a [\[Required\]](#) attribute on properties of the model class instead of implementing a `System.Text.Json` converter.

Specify date format

`Newtonsoft.Json` provides several ways to control how properties of `DateTime` and `DateTimeOffset` types are serialized and deserialized:

- The `DateTimeZoneHandling` setting can be used to serialize all `DateTime` values as UTC dates.
 - The `DateFormatString` setting and `DateTime` converters can be used to customize the format of date strings.

[System.Text.Json](#) supports ISO 8601-1:2019, including the RFC 3339 profile. This format is widely adopted, unambiguous, and makes round trips precisely. To use any other format, create a custom converter. For example, the following converters serialize and deserialize JSON that uses Unix epoch format with or without a time zone offset (values such as `/Date(1590863400000-0700)/` or `/Date(1590863400000)/`):

```

sealed class UnixEpochDateTimeOffsetConverter : JsonConverter<DateTimeOffset>
{
    static readonly DateTimeOffset s_epoch = new DateTimeOffset(1970, 1, 1, 0, 0, 0, TimeSpan.Zero);
    static readonly Regex s_regex = new Regex("^/Date\\(((\\[+-]\\d{1,2})\\((\\[+-]\\d{1,2})\\d{2}\\))\\d{2}\\)\\$/",
        RegexOptions.CultureInvariant);

    public override DateTimeOffset Read(ref Utf8JsonReader reader, Type typeToConvert, JsonSerializerOptions options)
    {
        string formatted = reader.GetString()!;
        Match match = s_regex.Match(formatted);

        if (
            !match.Success
            || !long.TryParse(match.Groups[1].Value, System.Globalization.NumberStyles.Integer,
CultureInfo.InvariantCulture, out long unixTime)
            || !int.TryParse(match.Groups[3].Value, System.Globalization.NumberStyles.Integer,
CultureInfo.InvariantCulture, out int hours)
            || !int.TryParse(match.Groups[4].Value, System.Globalization.NumberStyles.Integer,
CultureInfo.InvariantCulture, out int minutes))
        {
            throw new JsonException();
        }

        int sign = match.Groups[2].Value[0] == '+' ? 1 : -1;
        TimeSpan utcOffset = new TimeSpan(hours * sign, minutes * sign, 0);

        return s_epoch.AddMilliseconds(unixTime).ToOffset(utcOffset);
    }

    public override void Write(Utf8JsonWriter writer, DateTimeOffset value, JsonSerializerOptions options)
    {
        long unixTime = Convert.ToInt64((value - s_epoch).TotalMilliseconds);
        TimeSpan utcOffset = value.Offset;

        string formatted = FormattableString.Invariant($""/Date({unixTime}{{(utcOffset >= TimeSpan.Zero ? "+" :
"-")}{(utcOffset:hhmm)}})/");
        writer.WriteStringValue(formatted);
    }
}

```

```

sealed class UnixEpochDateTimeConverter : JsonConverter<DateTime>
{
    static readonly DateTime s_epoch = new DateTime(1970, 1, 1, 0, 0, 0);
    static readonly Regex s_regex = new Regex("^/Date\\\"(([+-]*/\\d+)\\\")/", RegexOptions.CultureInvariant);

    public override DateTime Read(ref Utf8JsonReader reader, Type typeToConvert, JsonSerializerOptions options)
    {
        string formatted = reader.GetString()!;
        Match match = s_regex.Match(formatted);

        if (
            !match.Success
            || !long.TryParse(match.Groups[1].Value, System.Globalization.NumberStyles.Integer,
CultureInfo.InvariantCulture, out long unixTime))
        {
            throw new JsonException();
        }

        return s_epoch.AddMilliseconds(unixTime);
    }

    public override void Write(Utf8JsonWriter writer, DateTime value, JsonSerializerOptions options)
    {
        long unixTime = Convert.ToInt64((value - s_epoch).TotalMilliseconds);

        string formatted = FormattableString.Invariant($""/Date({unixTime})/");
        writer.WriteStringValue(formatted);
    }
}

```

For more information, see [DateTime and DateTimeOffset support in System.Text.Json](#).

Callbacks

`Newtonsoft.Json` lets you execute custom code at several points in the serialization or deserialization process:

- `OnDeserializing` (when beginning to deserialize an object)
- `OnDeserialized` (when finished deserializing an object)
- `OnSerializing` (when beginning to serialize an object)
- `OnSerialized` (when finished serializing an object)

`System.Text.Json` exposes the same notifications during serialization and deserialization. To use them, implement one or more of the following interfaces from the [System.Text.Json.Serialization](#) namespace:

- [IJsonOnDeserializing](#)
- [IJsonOnDeserialized](#)
- [IJsonOnSerializing](#)
- [IJsonOnSerialized](#)

Here's an example that checks for a null property and writes messages at start and end of serialization and deserialization:

```

using System.Text.Json;
using System.Text.Json.Serialization;

namespace Callbacks
{
    public class WeatherForecast :
        IJsonOnDeserializing, IJsonOnDeserialized,
        IJsonOnSerializing, IJsonOnSerialized
    {
        public void OnSerializing(JsonSerializerContext context)
        {
            if (context.JsonPropertyInfo?.Name == "Temperature")
                context.JsonPropertyInfo.Value = 10;
        }

        public void OnSerialized(JsonSerializerContext context)
        {
            Console.WriteLine("Serialized");
        }

        public void OnDeserializing(JsonSerializerContext context)
        {
            if (context.JsonPropertyInfo?.Name == "Temperature")
                context.JsonPropertyInfo.Value = null;
        }

        public void OnDeserialized(JsonSerializerContext context)
        {
            Console.WriteLine("Deserialized");
        }
    }
}

```

```

    {
        public DateTime Date { get; set; }
        public int TemperatureCelsius { get; set; }
        public string? Summary { get; set; }

        void IJsonOnDeserializing.OnDeserializing() => Console.WriteLine("\nBegin deserializing");
        void IJsonOnDeserialized.OnDeserialized()
        {
            Validate();
            Console.WriteLine("Finished deserializing");
        }
        void IJsonOnSerializing.OnSerializing()
        {
            Console.WriteLine("Begin serializing");
            Validate();
        }
        void IJsonOnSerialized.OnSerialized() => Console.WriteLine("Finished serializing");

        private void Validate()
        {
            if (Summary is null)
            {
                Console.WriteLine("The 'Summary' property is 'null'.");
            }
        }
    }

    public class Program
    {
        public static void Main()
        {
            var weatherForecast = new WeatherForecast
            {
                Date = DateTime.Parse("2019-08-01"),
                TemperatureCelsius = 25,
            };

            string jsonString = JsonSerializer.Serialize(weatherForecast);
            Console.WriteLine(jsonString);

            weatherForecast = JsonSerializer.Deserialize<WeatherForecast>(jsonString);
            Console.WriteLine($"Date={weatherForecast?.Date}");
            Console.WriteLine($"TemperatureCelsius={weatherForecast?.TemperatureCelsius}");
            Console.WriteLine($"Summary={weatherForecast?.Summary}");
        }
    }
}

// output:
//Begin serializing
//The 'Summary' property is 'null'.
//Finished serializing
//{"Date":"2019-08-01T00:00:00","TemperatureCelsius":25,"Summary":null}

//Begin deserializing
//The 'Summary' property is 'null'.
//Finished deserializing
//Date=8/1/2019 12:00:00 AM
//TemperatureCelsius = 25
//Summary=

```

The `OnDeserializing` code doesn't have access to the new POCO instance. To manipulate the new POCO instance at the start of deserialization, put that code in the POCO constructor.

In [System.Text.Json](#), you can simulate callbacks by writing a custom converter. The following example shows a custom converter for a POCO. The converter includes code that displays a message at each point that corresponds to a `Newtonsoft.Json` callback.

```

using System.Text.Json;
using System.Text.Json.Serialization;

namespace SystemTextJsonSamples
{
    public class WeatherForecastCallbacksConverter : JsonConverter<WeatherForecast>
    {
        public override WeatherForecast Read(
            ref Utf8JsonReader reader,
            Type type,
            JsonSerializerOptions options)
        {
            // Place "before" code here (OnDeserializing),
            // but note that there is no access here to the POCO instance.
            Console.WriteLine("OnDeserializing");

            // Don't pass in options when recursively calling Deserialize.
            WeatherForecast forecast = JsonSerializer.Deserialize<WeatherForecast>(ref reader)!;

            // Place "after" code here (OnDeserialized)
            Console.WriteLine("OnDeserialized");

            return forecast;
        }

        public override void Write(
            Utf8JsonWriter writer,
            WeatherForecast forecast, JsonSerializerOptions options)
        {
            // Place "before" code here (OnSerializing)
            Console.WriteLine("OnSerializing");

            // Don't pass in options when recursively calling Serialize.
            JsonSerializer.Serialize(writer, forecast);

            // Place "after" code here (OnSerialized)
            Console.WriteLine("OnSerialized");
        }
    }
}

```

Register this custom converter by [adding the converter](#) to the [Converters](#) collection.

If you use a custom converter that follows the preceding sample:

- The `OnDeserializing` code doesn't have access to the new POCO instance. To manipulate the new POCO instance at the start of deserialization, put that code in the POCO constructor.
- Avoid an infinite loop by registering the converter in the options object and not passing in the options object when recursively calling `Serialize` or `Deserialize`.

For more information about custom converters that recursively call `Serialize` or `Deserialize`, see the [Required properties](#) section earlier in this article.

Non-public property setters and getters

`Newtonsoft.Json` can use private and internal property setters and getters via the `JsonProperty` attribute.

`System.Text.Json` supports private and internal property setters and getters via the `[JsonInclude]` attribute. For sample code, see [Non-public property accessors](#).

`System.Text.Json` in .NET Core 3.1 supports only public setters. Custom converters can provide this functionality.

Populate existing objects

The `JsonConvert.PopulateObject` method in `Newtonsoft.Json` deserializes a JSON document to an existing

instance of a class, instead of creating a new instance. [System.Text.Json](#) always creates a new instance of the target type by using the default public parameterless constructor. Custom converters can deserialize to an existing instance.

Reuse rather than replace properties

The `Newtonsoft.Json ObjectCreationHandling` setting lets you specify that objects in properties should be reused rather than replaced during deserialization. [System.Text.Json](#) always replaces objects in properties. Custom converters can provide this functionality.

Add to collections without setters

During deserialization, `Newtonsoft.Json` adds objects to a collection even if the property has no setter. [System.Text.Json](#) ignores properties that don't have setters. Custom converters can provide this functionality.

Snake case naming policy

The only built-in property naming policy in [System.Text.Json](#) is for [camel case](#). `Newtonsoft.Json` can convert property names to snake case. A [custom naming policy](#) can provide this functionality. For more information, see GitHub issue [dotnet/runtime #782](#).

System.Runtime.Serialization attributes

[System.Text.Json](#) doesn't support attributes from the `System.Runtime.Serialization` namespace, such as `DataMemberAttribute` and `IgnoreDataMemberAttribute`.

Octal numbers

`Newtonsoft.Json` treats numbers with a leading zero as octal numbers. [System.Text.Json](#) doesn't allow leading zeroes because the [RFC 8259](#) specification doesn't allow them.

MissingMemberHandling

`Newtonsoft.Json` can be configured to throw exceptions during deserialization if the JSON includes properties that are missing in the target type. [System.Text.Json](#) ignores extra properties in the JSON, except when you use the `[JsonExtensionData]` attribute. There's no workaround for the missing member feature.

TraceWriter

`Newtonsoft.Json` lets you debug by using a `TraceWriter` to view logs that are generated by serialization or deserialization. [System.Text.Json](#) doesn't do logging.

JsonDocument and JsonElement compared to JToken (like JObject, JArray)

[System.Text.Json.JsonDocument](#) provides the ability to parse and build a **read-only** Document Object Model (DOM) from existing JSON payloads. The DOM provides random access to data in a JSON payload. The JSON elements that compose the payload can be accessed via the `JsonElement` type. The `JsonElement` type provides APIs to convert JSON text to common .NET types. `JsonDocument` exposes a `RootElement` property.

Starting in .NET 6, you can parse and build a **mutable** DOM from existing JSON payloads by using the `JsonNode` type and other types in the `System.Text.Json.Nodes` namespace. For more information, see [Use JsonNode](#).

JsonDocument is IDisposable

`JsonDocument` builds an in-memory view of the data into a pooled buffer. Therefore, unlike `JObject` or `JArray` from `Newtonsoft.Json`, the `JsonDocument` type implements `IDisposable` and needs to be used inside a using block. For more information, see [JsonDocument is IDisposable](#).

JsonDocument is read-only

The `System.Text.Json` DOM can't add, remove, or modify JSON elements. It's designed this way for performance

and to reduce allocations for parsing common JSON payload sizes (that is, < 1 MB).

If your scenario currently uses a modifiable DOM, one of the following workarounds might be feasible:

- To build a `JsonDocument` from scratch (that is, without passing in an existing JSON payload to the `Parse` method), write the JSON text by using the `Utf8JsonWriter` and parse the output from that to make a new `JsonDocument`.
- To modify an existing `JsonDocument`, use it to write JSON text, making changes while you write, and parse the output from that to make a new `JsonDocument`.
- To merge existing JSON documents, equivalent to the `JObject.Merge` or `JContainer.Merge` APIs from `Newtonsoft.Json`, see [this GitHub issue](#).

These workarounds are necessary only for versions of `System.Text.Json` earlier than 6.0. In 6.0 you can use `JsonNode` to work with a mutable DOM.

JsonElement is a union struct

`JsonDocument` exposes the `RootElement` as a property of type `JsonElement`, which is a union struct type that encompasses any JSON element. `Newtonsoft.Json` uses dedicated hierarchical types like `JObject`, `JArray`, `JToken`, and so forth. `JsonElement` is what you can search and enumerate over, and you can use `JsonElement` to materialize JSON elements into .NET types.

Starting in .NET 6, you can use `JsonNode` type and types in the `System.Text.Json.Nodes` namespace that correspond to `JObject`, `JArray`, and `JToken`. For more information, see [Use JsonNode](#).

How to search a JsonDocument and JsonElement for sub-elements

Searches for JSON tokens using `JObject` or `JArray` from `Newtonsoft.Json` tend to be relatively fast because they're lookups in some dictionary. By comparison, searches on `JsonElement` require a sequential search of the properties and hence are relatively slow (for example when using `TryGetProperty`). `System.Text.Json` is designed to minimize initial parse time rather than lookup time. For more information, see [How to search a JsonDocument and JsonElement for sub-elements](#).

Utf8JsonReader compared to JsonTextReader

`System.Text.Json.Utf8JsonReader` is a high-performance, low allocation, forward-only reader for UTF-8 encoded JSON text, read from a `ReadOnlySpan<byte>` or `ReadOnlySequence<byte>`. The `Utf8JsonReader` is a low-level type that can be used to build custom parsers and deserializers.

Utf8JsonReader is a ref struct

The `JsonTextReader` in `Newtonsoft.Json` is a class. The `Utf8JsonReader` type differs in that it's a *ref struct*. For more information, see [Utf8JsonReader is a ref struct](#).

Read null values into nullable value types

`Newtonsoft.Json` provides APIs that return `Nullable<T>`, such as `ReadAsBoolean`, which handles a `NullTokenType` for you by returning a `bool?`. The built-in `System.Text.Json` APIs return only non-nullable value types. For more information, see [Read null values into nullable value types](#).

Multi-targeting

If you need to continue to use `Newtonsoft.Json` for certain target frameworks, you can multi-target and have two implementations. However, this is not trivial and would require some `#ifdefs` and source duplication. One way to share as much code as possible is to create a `ref struct` wrapper around `Utf8JsonReader` and `Newtonsoft.Json.JsonTextReader`. This wrapper would unify the public surface area while isolating the behavioral differences. This lets you isolate the changes mainly to the construction of the type, along with passing the new type around by reference. This is the pattern that the `Microsoft.Extensions.DependencyModel` library follows:

- [UnifiedJsonReader.JsonTextReader.cs](#)
- [UnifiedJsonReader.Utf8JsonReader.cs](#)

Utf8JsonWriter compared to JsonTextWriter

`System.Text.Json.Utf8JsonWriter` is a high-performance way to write UTF-8 encoded JSON text from common .NET types like `String`, `Int32`, and `DateTime`. The writer is a low-level type that can be used to build custom serializers.

Write raw values

The `Newtonsoft.Json.WriteRawValue` method writes raw JSON where a value is expected. `System.Text.Json` has a direct equivalent: [Utf8JsonWriter.WriteRawValue](#). For more information, see [Write raw JSON](#).

The `Newtonsoft.Json.WriteRawValue` method writes raw JSON where a value is expected. There is an equivalent method, [Utf8JsonWriter.WriteRawValue](#), in .NET 6. For more information, see [Write raw JSON](#).

For versions earlier than 6.0, `System.Text.Json` has no equivalent method for writing raw JSON. However, the following workaround ensures only valid JSON is written:

```
using JsonDocument doc = JsonDocument.Parse(string);
doc.WriteTo(writer);
```

Customize JSON format

`JsonTextWriter` includes the following settings, for which `Utf8JsonWriter` has no equivalent:

- `Indentation` - Specifies how many characters to indent. `Utf8JsonWriter` always does 2-character indentation.
- `IndentChar` - Specifies the character to use for indentation. `Utf8JsonWriter` always uses whitespace.
- `QuoteChar` - Specifies the character to use to surround string values. `Utf8JsonWriter` always uses double quotes.
- `QuoteName` - Specifies whether or not to surround property names with quotes. `Utf8JsonWriter` always surrounds them with quotes.

There are no workarounds that would let you customize the JSON produced by `Utf8JsonWriter` in these ways.

Write Timespan, Uri, or char values

`JsonTextWriter` provides `WriteValue` methods for `TimeSpan`, `Uri`, and `char` values. `Utf8JsonWriter` doesn't have equivalent methods. Instead, format these values as strings (by calling `ToString()`, for example) and call [WriteStringValue](#).

Multi-targeting

If you need to continue to use `Newtonsoft.Json` for certain target frameworks, you can multi-target and have two implementations. However, this is not trivial and would require some `#ifdefs` and source duplication. One way to share as much code as possible is to create a wrapper around `Utf8JsonWriter` and `Newtonsoft.JsonTextWriter`. This wrapper would unify the public surface area while isolating the behavioral differences. This lets you isolate the changes mainly to the construction of the type. `Microsoft.Extensions.DependencyModel` library follows:

- [UnifiedJsonWriter.JsonTextWriter.cs](#)
- [UnifiedJsonWriter.Utf8JsonWriter.cs](#)

TypeNameHandling.All not supported

The decision to exclude `TypeNameHandling.All`-equivalent functionality from `System.Text.Json` was intentional.

Allowing a JSON payload to specify its own type information is a common source of vulnerabilities in web applications. In particular, configuring `Newtonsoft.Json` with `TypeNameHandling.All` allows the remote client to embed an entire executable application within the JSON payload itself, so that during deserialization the web application extracts and runs the embedded code. For more information, see [Friday the 13th JSON attacks PowerPoint](#) and [Friday the 13th JSON attacks details](#).

JSON Path queries not supported

The `JsonDocument` DOM doesn't support querying by using [JSON Path](#).

In a `JsonNode` DOM, each `JsonNode` instance has a `GetPath` method that returns a path to that node. But there is no built-in API to handle queries based on JSON Path query strings.

For more information, see the [dotnet/runtime #31068 GitHub issue](#).

Some limits not configurable

`System.Text.Json` sets limits that can't be changed for some values, such as the maximum token size in characters (166 MB) and in base 64 (125 MB). For more information, see [JsonConstants](#) in the source code and GitHub issue [dotnet/runtime #39953](#).

NaN, Infinity, -Infinity

Newtonsoft parses `NaN`, `Infinity`, and `-Infinity` JSON string tokens. In .NET Core 3.1, `System.Text.Json` doesn't support these tokens but you can write a custom converter to handle them. In .NET 5 and later versions, use `JsonNumberHandling.AllowNamedFloatingPointLiterals`. For information about how to use this setting, see [Allow or write numbers in quotes](#).

Additional resources

- [System.Text.Json overview](#)
- [How to serialize and deserialize JSON](#)
- [Instantiate JsonSerializerOptions instances](#)
- [Enable case-insensitive matching](#)
- [Customize property names and values](#)
- [Ignore properties](#)
- [Allow invalid JSON](#)
- [Handle overflow JSON or use JsonElement or JsonNode](#)
- [Preserve references and handle circular references](#)
- [Deserialize to immutable types and non-public accessors](#)
- [Polymorphic serialization](#)
- [Customize character encoding](#)
- [Use DOM, Utf8JsonReader, and Utf8JsonWriter](#)
- [Write custom converters for JSON serialization](#)
- [DateTime and DateTimeOffset support](#)
- [How to use source generation](#)
- [Supported collection types](#)
- [System.Text.Json API reference](#)
- [System.Text.Json.Serialization API reference](#)

Supported collection types in System.Text.Json

9/20/2022 • 6 minutes to read • [Edit Online](#)

This article gives an overview of which collections are supported for serialization and deserialization.

`System.Text.Json.JsonSerializer` supports a collection type for serialization if it:

- Derives from `IEnumerable`.
- Contains elements that are serializable.
- Derives from `IEnumerable` or `IAsyncEnumerable<T>`
- Contains elements that are serializable.

The serializer calls the `GetEnumerator()` method, and writes the elements.

Deserialization is more complicated and is not supported for some collection types.

The following sections are organized by namespace and show which types are supported for serialization and deserialization.

System.Array namespace

TYPE	SERIALIZATION	DESERIALIZATION
Single-dimensional arrays	✓	✓
Multi-dimensional arrays	✗	✗
Jagged arrays	✓	✓

System.Collections namespace

TYPE	SERIALIZATION	DESERIALIZATION
ArrayList	✓	✓
BitArray	✓	✗
DictionaryEntry	✓	✓
Hashtable	✓	✓
ICollection	✓	✓
IDictionary	✓	✓
IEnumerable	✓	✓
IList	✓	✓

Type	Serialization	Deserialization
Queue	✓	✓
SortedList	✓	✓
Stack	✓	✓

System.Collections.Generic namespace

Type	Serialization	Deserialization
Dictionary<TKey,TValue> *	✓	✓
HashSet<T>	✓	✓
IAsyncEnumerable<T> **	✓	✓
ICollection<T>	✓	✓
IDictionary<TKey,TValue> *	✓	✓
IEnumerable<T>	✓	✓
IList<T>	✓	✓
IReadOnlyCollection<T>	✓	✓
IReadOnlyDictionary<TKey,TValue> *	✓	✓
IReadOnlyList<T>	✓	✓
ISet<T>	✓	✓
KeyValuePair<TKey,TValue>	✓	✓
LinkedList<T>	✓	✓
LinkedListNode<T>	✓	✗
List<T>	✓	✓
Queue<T>	✓	✓
SortedDictionary<TKey,TValue> *	✓	✓
SortedList<TKey,TValue> *	✓	✓
SortedSet<T>	✓	✓
Stack<T>	✓	✓

* See [Supported key types](#).

** See the following section on [IAsyncEnumerable<T>](#).

IAsyncEnumerable<T>

The following examples use streams as a representation of any async source of data. The source could be files on a local machine, or results from a database query or web service API call.

Stream serialization

`System.Text.Json` supports serializing [IAsyncEnumerable<T>](#) values as JSON arrays, as shown in the following example:

```
using System.Text.Json;

namespace IAsyncEnumerableSerialize;

public class Program
{
    public static async Task Main()
    {
        using Stream stream = Console.OpenStandardOutput();
        var data = new { Data = PrintNumbers(3) };
        await JsonSerializer.SerializeAsync(stream, data);
    }

    static async IAsyncEnumerable<int> PrintNumbers(int n)
    {
        for (int i = 0; i < n; i++)
        {
            await Task.Delay(1000);
            yield return i;
        }
    }
}
// output:
// {"Data": [0,1,2]}
```

[IAsyncEnumerable<T>](#) values are only supported by the asynchronous serialization methods, such as [JsonSerializer.SerializeAsync](#).

Stream deserialization

The [DeserializeAsyncEnumerable](#) method supports streaming deserialization, as shown in the following example:

```

using System.Text;
using System.Text.Json;

namespace IAsyncEnumerableDeserialize;

public class Program
{
    public static async Task Main()
    {
        using var stream = new MemoryStream(Encoding.UTF8.GetBytes("[0,1,2,3,4]"));
        await foreach (int item in JsonSerializer.DeserializeAsyncEnumerable<int>(stream))
        {
            Console.WriteLine(item);
        }
    }
}
// output:
//0
//1
//2
//3
//4

```

The `DeserializeAsyncEnumerable` method only supports reading from root-level JSON arrays.

The `DeserializeAsync` method supports `IAsyncEnumerable<T>`, but its signature doesn't allow streaming. It returns the final result as a single value, as shown in the following example.

```

using System.Text;
using System.Text.Json;

namespace IAsyncEnumerableDeserializeNonStreaming;

public class MyPoco
{
    public IAsyncEnumerable<int>? Data { get; set; }
}

public class Program
{
    public static async Task Main()
    {
        using var stream = new MemoryStream(Encoding.UTF8.GetBytes(@"{""Data"": [0,1,2,3,4]}"));
        MyPoco? result = await JsonSerializer.DeserializeAsync<MyPoco>(stream)!;
        await foreach (int item in result!.Data!)
        {
            Console.WriteLine(item);
        }
    }
}
// output:
//0
//1
//2
//3
//4

```

In this example, the deserializer buffers all `IAsyncEnumerable<T>` contents in memory before returning the deserialized object. This behavior is necessary because the deserializer needs to read the entire JSON payload before returning a result.

TYPE	SERIALIZATION	DESERIALIZATION
Dictionary<TKey,TValue> *	✓	✓
HashSet<T>	✓	✓
IAsyncEnumerable<T>	✗	✗
ICollection<T>	✓	✓
IDictionary<TKey,TValue> *	✓	✓
IEnumerable<T>	✓	✓
IList<T>	✓	✓
IReadOnlyCollection<T>	✓	✓
IReadOnlyDictionary<TKey,TValue> *	✓	✓
IReadOnlyList<T>	✓	✓
ISet<T>	✓	✓
KeyValuePair<TKey,TValue>	✓	✓
LinkedList<T>	✓	✓
LinkedListNode<T>	✓	✗
List<T>	✓	✓
Queue<T>	✓	✓
SortedDictionary<TKey,TValue> *	✓	✓
SortedList<TKey,TValue> *	✓	✓
SortedSet<T>	✓	✓
Stack<T>	✓	✓

* See [Supported key types](#).

TYPE	SERIALIZATION	DESERIALIZATION
Dictionary<string, TValue> *	✓	✓
HashSet<T>	✓	✓
IAsyncEnumerable<T>	✗	✗

TYPE	SERIALIZATION	DESERIALIZATION
<code>ICollection<T></code>	✓	✓
<code>IDictionary<string, TValue> *</code>	✓	✓
<code>IEnumerable<T></code>	✓	✓
<code>IList<T></code>	✓	✓
<code>IReadOnlyCollection<T></code>	✓	✓
<code>IReadOnlyDictionary<string, TValue> *</code>	✓	✓
<code>IReadOnlyList<T></code>	✓	✓
<code>ISet<T></code>	✓	✓
<code>KeyValuePair<TKey,TValue></code>	✓	✓
<code>LinkedList<T></code>	✓	✓
<code>LinkedListNode<T></code>	✓	✗
<code>List<T></code>	✓	✓
<code>Queue<T></code>	✓	✓
<code>SortedDictionary<string, TValue> *</code>	✓	✓
<code>SortedList<string, TValue> *</code>	✓	✓
<code>SortedSet<T></code>	✓	✓
<code>Stack<T></code>	✓	✓

* See [Supported key types](#).

System.Collections.Immutable namespace

TYPE	SERIALIZATION	DESERIALIZATION
<code>IImmutableDictionary< TKey, TValue ></code> **	✓	✓
<code>IImmutableList<T></code>	✓	✓
<code>IImmutableQueue<T></code>	✓	✓
<code>IImmutableSet<T></code>	✓	✓
<code>IImmutableStack<T> *</code>	✓	✓

TYPE	SERIALIZATION	DESERIALIZATION
ImmutableArray<T>	✓	✓
ImmutableDictionary< TKey, TValue > **	✓	✓
ImmutableHashSet<T>	✓	✓
ImmutableQueue<T>	✓	✓
ImmutableSortedDictionary< TKey, TValue > **	✓	✓
ImmutableSortedSet<T>	✓	✓
ImmutableStack<T> *	✓	✓

TYPE	SERIALIZATION	DESERIALIZATION
IImmutableDictionary< string, TValue > **	✓	✓
IImmutableList<T>	✓	✓
IImmutableQueue<T>	✓	✓
IImmutableSet<T>	✓	✓
IImmutableStack<T> *	✓	✓
ImmutableArray<T>	✓	✓
ImmutableDictionary< string, TValue > **	✓	✓
ImmutableHashSet<T>	✓	✓
IImmutableList<T>	✓	✓
ImmutableQueue<T>	✓	✓
ImmutableSortedDictionary< string, TValue > **	✓	✓
ImmutableSortedSet<T>	✓	✓
ImmutableStack<T> *	✓	✓

* See [Support round trip for Stack<T>](#).

** See [Supported key types](#).

System.Collections.Specialized namespace

TYPE	SERIALIZATION	DESERIALIZATION
BitVector32	✓	✗ *
HybridDictionary	✓	✓
IOrderedDictionary	✓	✗
ListDictionary	✓	✓
NameValueCollection	✓	✗
StringCollection	✓	✗
StringDictionary	✓	✗

* When [BitVector32](#) is deserialized, the [Data](#) property is skipped because it doesn't have a public setter. No exception is thrown.

System.Collections.Concurrent namespace

TYPE	SERIALIZATION	DESERIALIZATION
BlockingCollection<T>	✓	✗
ConcurrentBag<T>	✓	✗
ConcurrentDictionary<TKey,TValue> **	✓	✓
ConcurrentQueue<T>	✓	✓
ConcurrentStack<T> *	✓	✓

TYPE	SERIALIZATION	DESERIALIZATION
BlockingCollection<T>	✓	✗
ConcurrentBag<T>	✓	✗
ConcurrentDictionary<string, TValue> **	✓	✓
ConcurrentQueue<T>	✓	✓
ConcurrentStack<T> *	✓	✓

* See [Support round trip for Stack<T>](#).

** See [Supported key types](#).

System.Collections.ObjectModel namespace

TYPE	SERIALIZATION	DESERIALIZATION
Collection<T>	✓	✓
KeyedCollection<string, TValue> *	✓	✗
ObservableCollection<T>	✓	✓
ReadOnlyCollection<T>	✓	✗
ReadOnlyDictionary<TKey,TValue>	✓	✗
ReadOnlyObservableCollection<T>	✓	✗

* Non-`string` keys are not supported.

TYPE	SERIALIZATION	DESERIALIZATION
Collection<T>	✓	✓
KeyedCollection<string, TValue> *	✓	✗
ObservableCollection<T>	✓	✓
ReadOnlyCollection<T>	✓	✗
ReadOnlyDictionary<string, TValue> *	✓	✗
ReadOnlyObservableCollection<T>	✓	✗

* See [Supported key types](#).

Custom collections

Any collection type that isn't in one of the preceding namespaces is considered a custom collection. Such types include user-defined types and types defined by ASP.NET Core. For example, `Microsoft.Extensions.Primitives` is in this group.

All custom collections (everything that derives from `IEnumerable`) are supported for serialization, as long as their element types are supported.

Custom collections with deserialization support

A custom collection is supported for deserialization if it:

- Isn't an interface or abstract.
- Has a parameterless constructor.
- Contains element types that are supported by `JsonSerializer`.
- Implements or inherits one or more of the following interfaces or classes:
 - `ConcurrentQueue<T>`
 - `ConcurrentStack<T>` *
 - `ICollection<T>`
 - `IDictionary`
 - `IDictionary<TKey,TValue>` **

- [IList](#)
- [IList<T>](#)
- [Queue](#)
- [Queue<T>](#)
- [Stack *](#)
- [Stack<T> *](#)

- Isn't an interface or abstract.
- Has a parameterless constructor.
- Contains element types that are supported by [JsonSerializer](#).
- Implements or inherits one or more of the following interfaces or classes:
 - [ConcurrentQueue<T>](#)
 - [ConcurrentStack<T> *](#)
 - [ICollection<T>](#)
 - [IDictionary](#)
 - [IDictionary<string, TValue> **](#)
 - [IList](#)
 - [IList<T>](#)
 - [Queue](#)
 - [Queue<T>](#)
 - [Stack *](#)
 - [Stack<T> *](#)

* See [Support round trip for Stack<T>](#).

** See [Supported key types](#).

Custom collections with known issues

There are known issues with the following custom collections:

- [ExpandoObject](#): See [dotnet/runtime#29690](#).
- [DynamicObject](#): See [dotnet/runtime#1808](#).
- [DataTable](#): See [dotnet/docs#21366](#).
- [Microsoft.AspNetCore.Http.FormFile](#): See [dotnet/runtime#1559](#).
- [Microsoft.AspNetCore.Http.IFormCollection](#): See [dotnet/runtime#1559](#).

For more information about known issues, see the [open issues in System.Text.Json](#).

Supported key types

Supported types for the keys of [Dictionary](#) and [SortedList](#) types include the following:

- [Boolean](#)
- [Byte](#)
- [DateTime](#)
- [DateTimeOffset](#)
- [Decimal](#)
- [Double](#)
- [Enum](#)
- [Guid](#)
- [Int16](#)

- `Int32`
- `Int64`
- `Object` (Only on serialization and if the runtime type is one of the supported types in this list.)
- `SByte`
- `Single`
- `String`
- `UInt16`
- `UInt32`
- `UInt64`

** Non-`string` keys for `Dictionary` and `SortedList` types are not supported in .NET Core 3.1.

System.Data namespace

There are no built-in converters for `DataSet`, `DataTable`, and related types in the `System.Data` namespace. Deserializing these types from untrusted input is not safe, as explained in [the security guidance](#). However, you can write a custom converter to support these types. For sample custom converter code that serializes and deserializes a `DataTable`, see [RoundtripDataTable.cs](#).

See also

- [System.Text.Json overview](#)
- [Instantiate JsonSerializerOptions instances](#)
- [Enable case-insensitive matching](#)
- [Customize property names and values](#)
- [Ignore properties](#)
- [Allow invalid JSON](#)
- [Handle overflow JSON or use JsonElement or JsonNode](#)
- [Preserve references and handle circular references](#)
- [Deserialize to immutable types and non-public accessors](#)
- [Polymorphic serialization](#)
- [Migrate from Newtonsoft.Json to System.Text.Json](#)
- [Customize character encoding](#)
- [Use DOM, Utf8JsonReader, and Utf8JsonWriter](#)
- [Write custom converters for JSON serialization](#)
- [DateTime and DateTimeOffset support](#)
- [How to use source generation](#)
- [System.Text.Json API reference](#)
- [System.Text.Json.Serialization API reference](#)

How to customize character encoding with System.Text.Json

9/20/2022 • 3 minutes to read • [Edit Online](#)

By default, the serializer escapes all non-ASCII characters. That is, it replaces them with `\uxxxx` where `xxxx` is the Unicode code of the character. For example, if the `Summary` property in the following JSON is set to Cyrillic `жарко`, the `WeatherForecast` object is serialized as shown in this example:

```
{  
    "Date": "2019-08-01T00:00:00-07:00",  
    "TemperatureCelsius": 25,  
    "Summary": "\u0436\u0430\u0440\u0440\u043A\u043E"  
}
```

Serialize language character sets

To serialize the character set(s) of one or more languages without escaping, specify [Unicode range\(s\)](#) when creating an instance of `System.Text.Encodings.Web.JavaScriptEncoder`, as shown in the following example:

```
using System.Text.Encodings.Web;  
using System.Text.Json;  
using System.Text.Unicode;
```

```
Imports System.Text.Encodings.Web  
Imports System.Text.Json  
Imports System.Text.Unicode
```

```
options = new JsonSerializerOptions  
{  
    Encoder = JavaScriptEncoder.Create(UnicodeRanges.BasicLatin, UnicodeRanges.Cyrillic),  
    WriteIndented = true  
};  
jsonString = JsonSerializer.Serialize(weatherForecast, options);
```

```
options = New JsonSerializerOptions With {  
    .Encoder = JavaScriptEncoder.Create(UnicodeRanges.BasicLatin, UnicodeRanges.Cyrillic),  
    .WriteIndented = True  
}  
jsonString = JsonSerializer.Serialize(weatherForecast1, options)
```

This code doesn't escape Cyrillic or Greek characters. If the `Summary` property is set to Cyrillic `жарко`, the `WeatherForecast` object is serialized as shown in this example:

```
{  
    "Date": "2019-08-01T00:00:00-07:00",  
    "TemperatureCelsius": 25,  
    "Summary": "жарко"  
}
```

By default, the encoder is initialized with the [BasicLatin](#) range.

To serialize all language sets without escaping, use [UnicodeRanges.All](#).

Serialize specific characters

An alternative is to specify individual characters that you want to allow through without being escaped. The following example serializes only the first two characters of `жарко`:

```
using System.Text.Encodings.Web;
using System.Text.Json;
using System.Text.Unicode;
```

```
Imports System.Text.Encodings.Web
Imports System.Text.Json
Imports System.Text.Unicode
```

```
var encoderSettings = new TextEncoderSettings();
encoderSettings.AllowCharacters('\u0436', '\u0430');
encoderSettings.AllowRange(UnicodeRanges.BasicLatin);
options = new JsonSerializerOptions
{
    Encoder = JavaScriptEncoder.Create(encoderSettings),
    WriteIndented = true
};
jsonString = JsonSerializer.Serialize(weatherForecast, options);
```

```
Dim encoderSettings As TextEncoderSettings = New TextEncoderSettings
encoderSettings.AllowCharacters(ChrW(&H436), ChrW(&H430))
encoderSettings.AllowRange(UnicodeRanges.BasicLatin)
options = New JsonSerializerOptions With {
    .Encoder = JavaScriptEncoder.Create(encoderSettings),
    .WriteIndented = True
}
jsonString = JsonSerializer.Serialize(weatherForecast1, options)
```

Here's an example of JSON produced by the preceding code:

```
{
    "Date": "2019-08-01T00:00:00-07:00",
    "TemperatureCelsius": 25,
    "Summary": "\u043A\u043E\u0431\u043E\u0437\u0430\u043D\u0438\u044F"
}
```

Block lists

The preceding sections show how to specify allow lists of code points or ranges that you don't want to be escaped. However, there are global and encoder-specific block lists that can override certain code points in your allow list. Code points in a block list are always escaped, even if they're included in your allow list.

Global block list

The global block list includes things like private-use characters, control characters, undefined code points, and certain Unicode categories, such as the [Space_Separator category](#), excluding `U+0020 SPACE`. For example, `U+3000 IDEOGRAPHIC SPACE` is escaped even if you specify Unicode range [CJK Symbols and Punctuation \(U+3000-U+303F\)](#) as your allow list.

The global block list is an implementation detail that has changed in every release of .NET Core and in .NET 5. Don't take a dependency on a character being a member of (or not being a member of) the global block list.

Encoder-specific block lists

Examples of encoder-specific blocked code points include '`<`' and '`&`' for the [HTML encoder](#), '`\'` for the [JSON encoder](#), and '`%`' for the [URL encoder](#). For example, the HTML encoder always escapes ampersands (`&`), even though the ampersand is in the `BasicLatin` range and all the encoders are initialized with `BasicLatin` by default.

Serialize all characters

To minimize escaping you can use `JavaScriptEncoder.UnsafeRelaxedJsonEscaping`, as shown in the following example:

```
using System.Text.Encodings.Web;
using System.Text.Json;
using System.Text.Unicode;
```

```
Imports System.Text.Encodings.Web
Imports System.Text.Json
Imports System.Text.Unicode
```

```
options = new JsonSerializerOptions
{
    Encoder = JavaScriptEncoder.UnsafeRelaxedJsonEscaping,
    WriteIndented = true
};
jsonString = JsonSerializer.Serialize(weatherForecast, options);
```

```
options = New JsonSerializerOptions With {
    .Encoder = JavaScriptEncoder.UnsafeRelaxedJsonEscaping,
    .WriteIndented = True
}
jsonString = JsonSerializer.Serialize(weatherForecast1, options)
```

Caution

Compared to the default encoder, the `UnsafeRelaxedJsonEscaping` encoder is more permissive about allowing characters to pass through unescaped:

- It doesn't escape HTML-sensitive characters such as `<`, `>`, `&`, and `'`.
- It doesn't offer any additional defense-in-depth protections against XSS or information disclosure attacks, such as those which might result from the client and server disagreeing on the `charset`.

Use the unsafe encoder only when it's known that the client will be interpreting the resulting payload as UTF-8 encoded JSON. For example, you can use it if the server is sending the response header

`Content-Type: application/json; charset=utf-8`. Never allow the raw `UnsafeRelaxedJsonEscaping` output to be emitted into an HTML page or a `<script>` element.

See also

- [System.Text.Json overview](#)
- [How to serialize and deserialize JSON](#)
- [Instantiate JsonSerializerOptions instances](#)

- [Enable case-insensitive matching](#)
- [Customize property names and values](#)
- [Ignore properties](#)
- [Allow invalid JSON](#)
- [Handle overflow JSON or use JsonElement or JsonNode](#)
- [Preserve references and handle circular references](#)
- [Deserialize to immutable types and non-public accessors](#)
- [Polymorphic serialization](#)
- [Migrate from Newtonsoft.Json to System.Text.Json](#)
- [Use DOM, Utf8JsonReader, and Utf8JsonWriter](#)
- [Write custom converters for JSON serialization](#)
- [DateTime and DateTimeOffset support](#)
- [How to use source generation](#)
- [Supported collection types](#)
- [System.Text.Json API reference](#)
- [System.Text.Json.Serialization API reference](#)

How to use source generation in System.Text.Json

9/20/2022 • 8 minutes to read • [Edit Online](#)

This article shows how to use the source generation features of [System.Text.Json](#).

For information about how to use source generation in System.Text.Json, see the [.NET 6 version of this article](#).

Use source generation defaults

To use source generation with all defaults (both modes, default options):

- Create a partial class that derives from [JsonSerializerContext](#).
- Specify the type to serialize or deserialize by applying [JsonSerializableAttribute](#) to the context class.
- Call a [JsonSerializer](#) method that takes a [JsonTypeInfo<T>](#) instance. An alternative is to call a [JsonSerializer](#) method that takes a [JsonSerializerContext](#) instance.

By default, both source generation modes are used if you don't specify one. For information about how to specify the mode to use, see [Specify source generation mode](#) later in this article.

Here's the type that is used in the following examples:

```
public class WeatherForecast
{
    public DateTime Date { get; set; }
    public int TemperatureCelsius { get; set; }
    public string? Summary { get; set; }
}
```

Here's the context class configured to do source generation for the preceding `WeatherForecast` class:

```
[JsonSourceGenerationOptions(WriteIndented = true)]
[JsonSerializable(typeof(WeatherForecast))]
internal partial class SourceGenerationContext : JsonSerializerContext
{}
```

The types of `WeatherForecast` members don't need to be explicitly specified with `[JsonSerializable]` attributes.

Members declared as `object` are an exception to this rule. The runtime type for a member declared as `object` needs to be specified. For example, suppose you have the following class:

```
public class WeatherForecast
{
    public object? Data { get; set; }
    public List<object>? DataList { get; set; }
}
```

And you know that at runtime it may have `boolean` and `int` objects:

```
WeatherForecast wf = new() { Data = true, DataList = new List<object> { true, 1 } };
```

Then `boolean` and `int` have to be declared as `[JsonSerializable]`:

```
[JsonSerializable(typeof(WeatherForecast))]  
[JsonSerializable(typeof(bool))]  
[JsonSerializable(typeof(int))]  
public partial class WeatherForecastContext : JsonSerializerContext  
{  
}
```

To specify source generation for a collection, use `[JsonSerializable]` with the collection type. For example:

```
[JsonSerializable(typeof(List<WeatherForecast>))].
```

`JsonSerializer` methods that use source generation

The following examples call methods that serialize:

```
jsonString = JsonSerializer.Serialize(  
    weatherForecast!, SourceGenerationContext.Default.WeatherForecast);
```

```
jsonString = JsonSerializer.Serialize(  
    weatherForecast, typeof(WeatherForecast), SourceGenerationContext.Default);
```

The following examples call methods that deserialize:

```
weatherForecast = JsonSerializer.Deserialize<WeatherForecast>(  
    jsonString, SourceGenerationContext.Default.WeatherForecast);
```

```
weatherForecast = JsonSerializer.Deserialize(  
    jsonString, typeof(WeatherForecast), SourceGenerationContext.Default)  
    as WeatherForecast;
```

In the preceding examples, the static `Default` property of the context type provides an instance of the context type with default options. The context instance provides a `WeatherForecast` property that returns a `JsonTypeInfo<WeatherForecast>` instance. You can specify a different name for this property by using the `TypeInfoPropertyName` property of the `[JsonSerializable]` attribute.

Complete program example

Here are the preceding examples in a complete program:

```

using System.Text.Json;
using System.Text.Json.Serialization;

namespace BothModesNoOptions
{
    public class WeatherForecast
    {
        public DateTime Date { get; set; }
        public int TemperatureCelsius { get; set; }
        public string? Summary { get; set; }
    }

    [JsonSourceGenerationOptions(WriteIndented = true)]
    [JsonSerializable(typeof(WeatherForecast))]
    internal partial class SourceGenerationContext : JsonSerializerContext
    {
    }

    public class Program
    {
        public static void Main()
        {
            string jsonString =
@"
{
    ""Date"": ""2019-08-01T00:00:00"",
    ""TemperatureCelsius"": 25,
    ""Summary"": ""Hot"""
}

        WeatherForecast? weatherForecast;

        weatherForecast = JsonSerializer.Deserialize<WeatherForecast>(
            jsonString, SourceGenerationContext.Default.WeatherForecast);
        Console.WriteLine($"Date={weatherForecast?.Date}");
        // output:
        //Date=8/1/2019 12:00:00 AM

        weatherForecast = JsonSerializer.Deserialize(
            jsonString, typeof(WeatherForecast), SourceGenerationContext.Default)
            as WeatherForecast;
        Console.WriteLine($"Date={weatherForecast?.Date}");
        // output:
        //Date=8/1/2019 12:00:00 AM

        jsonString = JsonSerializer.Serialize(
            weatherForecast!, SourceGenerationContext.Default.WeatherForecast);
        Console.WriteLine(jsonString);
        // output:
        //{"Date":"2019-08-01T00:00:00","TemperatureCelsius":25,"Summary":"Hot"}

        jsonString = JsonSerializer.Serialize(
            weatherForecast, typeof(WeatherForecast), SourceGenerationContext.Default);
        Console.WriteLine(jsonString);
        // output:
        //{"Date":"2019-08-01T00:00:00","TemperatureCelsius":25,"Summary":"Hot"}
    }
}
}

```

Specify source generation mode

You can specify metadata collection mode or serialization optimization mode for an entire context, which may include multiple types. Or you can specify the mode for an individual type. If you do both, the mode specification for a type wins.

- For an entire context, use the [JsonSourceGenerationOptionsAttribute.GenerationMode](#) property.
- For an individual type, use the [JsonSerializableAttribute.GenerationMode](#) property.

Serialization optimization mode example

- For an entire context:

```
[JsonSourceGenerationOptions(GenerationMode = JsonSourceGenerationMode.Serialization)]
[JsonSerializable(typeof(WeatherForecast))]
internal partial class SerializeOnlyContext : JsonSerializerContext
{
}
```

- For an individual type:

```
[JsonSerializable(typeof(WeatherForecast), GenerationMode = JsonSourceGenerationMode.Serialization)]
internal partial class SerializeOnlyWeatherForecastOnlyContext : JsonSerializerContext
{}
```

- Complete program example

```

using System.Text.Json;
using System.Text.Json.Serialization;

namespace SerializeOnlyNoOptions
{
    public class WeatherForecast
    {
        public DateTime Date { get; set; }
        public int TemperatureCelsius { get; set; }
        public string? Summary { get; set; }
    }

    [JsonSourceGenerationOptions(GenerationMode = JsonSourceGenerationMode.Serialization)]
    [JsonSerializable(typeof(WeatherForecast))]
    internal partial class SerializeOnlyContext : JsonSerializerContext
    {

    }

    [JsonSerializable(typeof(WeatherForecast), GenerationMode =
JsonSourceGenerationMode.Serialization)]
    internal partial class SerializeOnlyWeatherForecastOnlyContext : JsonSerializerContext
    {

    }

    public class Program
    {
        public static void Main()
        {
            string jsonString;
            WeatherForecast weatherForecast = new()
            { Date = DateTime.Parse("2019-08-01"), TemperatureCelsius = 25, Summary = "Hot" };

            // Use context that selects Serialization mode only for WeatherForecast.
            jsonString = JsonSerializer.Serialize(weatherForecast,
                SerializeOnlyWeatherForecastOnlyContext.Default.WeatherForecast);
            Console.WriteLine(jsonString);
            // output:
            // {"Date":"2019-08-01T00:00:00", "TemperatureCelsius":25, "Summary":"Hot"}

            // Use a context that selects Serialization mode.
            jsonString = JsonSerializer.Serialize(weatherForecast,
                SerializeOnlyContext.Default.WeatherForecast);
            Console.WriteLine(jsonString);
            // output:
            // {"Date":"2019-08-01T00:00:00", "TemperatureCelsius":25, "Summary":"Hot"}
        }
    }
}

```

Metadata collection mode example

- For an entire context:

```

[JsonSourceGenerationOptions(GenerationMode = JsonSourceGenerationMode.Metadata)]
[JsonSerializable(typeof(WeatherForecast))]
internal partial class MetadataOnlyContext : JsonSerializerContext
{
}

```

```

jsonString = JsonSerializer.Serialize(
    weatherForecast!, MetadataOnlyContext.Default.WeatherForecast);

```

```
weatherForecast = JsonSerializer.Deserialize<WeatherForecast>(
    jsonString, MetadataOnlyContext.Default.WeatherForecast);
```

- For an individual type:

```
[JsonSerializable(typeof(WeatherForecast), GenerationMode = JsonSourceGenerationMode.Metadata)]
internal partial class MetadataOnlyWeatherForecastOnlyContext : JsonSerializerContext
{
}
```

```
jsonString = JsonSerializer.Serialize(
    weatherForecast!,
    MetadataOnlyWeatherForecastOnlyContext.Default.WeatherForecast);
```

```
weatherForecast = JsonSerializer.Deserialize<WeatherForecast>(
    jsonString, MetadataOnlyWeatherForecastOnlyContext.Default.WeatherForecast);
```

- Complete program example

```
using System.Text.Json;
using System.Text.Json.Serialization;

namespace MetadataOnlyNoOptions
{
    public class WeatherForecast
    {
        public DateTime Date { get; set; }
        public int TemperatureCelsius { get; set; }
        public string? Summary { get; set; }
    }

    [JsonSerializable(typeof(WeatherForecast), GenerationMode = JsonSourceGenerationMode.Metadata)]
    internal partial class MetadataOnlyWeatherForecastOnlyContext : JsonSerializerContext
    {
    }

    [JsonSourceGenerationOptions(GenerationMode = JsonSourceGenerationMode.Metadata)]
    [JsonSerializable(typeof(WeatherForecast))]
    internal partial class MetadataOnlyContext : JsonSerializerContext
    {
    }

    public class Program
    {
        public static void Main()
        {
            string jsonString =
@"
{
    ""Date"": ""2019-08-01T00:00:00"",
    ""TemperatureCelsius"": 25,
    ""Summary"": ""Hot""
}
";  
            WeatherForecast? weatherForecast;  
  
            // Deserialize with context that selects metadata mode only for WeatherForecast only.  
            weatherForecast = JsonSerializer.Deserialize<WeatherForecast>(
                jsonString, MetadataOnlyWeatherForecastOnlyContext.Default.WeatherForecast);
            Console.WriteLine($"Date={weatherForecast?.Date}");
            // output:  
            //Date=8/1/2019 12:00:00 AM
```

```

        // Serialize with context that selects metadata mode only for WeatherForecast only.
        jsonString = JsonSerializer.Serialize(
            weatherForecast!,
            MetadataOnlyWeatherForecastOnlyContext.Default.WeatherForecast);
        Console.WriteLine(jsonString);
        // output:
        // {"Date": "2019-08-01T00:00:00", "TemperatureCelsius": 25, "Summary": "Hot"}

        // Deserialize with context that selects metadata mode only.
        weatherForecast = JsonSerializer.Deserialize<WeatherForecast>(
            jsonString, MetadataOnlyContext.Default.WeatherForecast);
        Console.WriteLine($"Date={weatherForecast?.Date}");
        // output:
        // Date=8/1/2019 12:00:00 AM

        // Serialize with context that selects metadata mode only.
        jsonString = JsonSerializer.Serialize(
            weatherForecast!, MetadataOnlyContext.Default.WeatherForecast);
        Console.WriteLine(jsonString);
        // output:
        // {"Date": "2019-08-01T00:00:00", "TemperatureCelsius": 0, "Summary": "Hot"}
    }
}
}

```

Specify options for serialization optimization mode

Use `JsonSourceGenerationOptionsAttribute` to specify options that are supported by serialization optimization mode. You can use these options without causing a fallback to `JsonSerializer` code. For example,

`WriteIndented` and `CamelCase` are supported:

```

[JsonSourceGenerationOptions(
    WriteIndented = true,
    PropertyNamingPolicy = JsonKnownNamingPolicy.CamelCase,
    GenerationMode = JsonSourceGenerationMode.Serialization)]
[JsonSerializable(typeof(WeatherForecast))]
internal partial class SerializationModeOptionsContext : JsonSerializerContext
{
}

```

When using `JsonSourceGenerationOptionsAttribute` to specify serialization options, call one of the following serialization methods:

- A `JsonSerializer.Serialize` method that takes a `TypeInfo< TValue >`. Pass it the `Default.<TypeName>` property of your context class:

```

jsonString = JsonSerializer.Serialize(
    weatherForecast, SerializationModeOptionsContext.Default.WeatherForecast);

```

- A `JsonSerializer.Serialize` method that takes a context. Pass it the `Default` static property of your context class.

```

jsonString = JsonSerializer.Serialize(
    weatherForecast, typeof(WeatherForecast), SerializationModeOptionsContext.Default);

```

If you call a method that lets you pass in your own instance of `Utf8JsonWriter`, the writer's `Indented` setting is honored instead of the `JsonSourceGenerationOptionsAttribute.WriteIndented` option.

If you create and use a context instance by calling the constructor that takes a `JsonSerializerOptions` instance, the supplied instance will be used instead of the options specified by `JsonSourceGenerationOptionsAttribute`.

Here are the preceding examples in a complete program:

```
using System.Text.Json;
using System.Text.Json.Serialization;

namespace SerializeOnlyWithOptions
{
    public class WeatherForecast
    {
        public DateTime Date { get; set; }
        public int TemperatureCelsius { get; set; }
        public string? Summary { get; set; }
    }

    [JsonSourceGenerationOptions(
        WriteIndented = true,
        PropertyNamingPolicy = JsonKnownNamingPolicy.CamelCase,
        GenerationMode = JsonSourceGenerationMode.Serialization)]
    [JsonSerializable(typeof(WeatherForecast))]
    internal partial class SerializationModeOptionsContext : JsonSerializerContext
    {
    }

    public class Program
    {
        public static void Main()
        {
            string jsonString;
            WeatherForecast weatherForecast = new()
            {
                Date = DateTime.Parse("2019-08-01"),
                TemperatureCelsius = 25,
                Summary = "Hot"
            };

            // Serialize using TypeInfo< TValue > provided by the context
            // and options specified by [JsonSourceGenerationOptions].
            jsonString = JsonSerializer.Serialize(
                weatherForecast,
                SerializationModeOptionsContext.Default.WeatherForecast);
            Console.WriteLine(jsonString);
            // output:
            //{
            //   "date": "2019-08-01T00:00:00",
            //   "temperatureCelsius": 0,
            //   "summary": "Hot"
            //}

            // Serialize using Default context
            // and options specified by [JsonSourceGenerationOptions].
            jsonString = JsonSerializer.Serialize(
                weatherForecast,
                typeof(WeatherForecast),
                SerializationModeOptionsContext.Default);
            Console.WriteLine(jsonString);
            // output:
            //{
            //   "date": "2019-08-01T00:00:00",
            //   "temperatureCelsius": 0,
            //   "summary": "Hot"
            //}
        }
    }
}
```

Specify options by using `JsonSerializerOptions`

Some options of `JsonSerializerOptions` aren't supported by serialization optimization mode. Such options cause a fallback to the non-source-generated `JsonSerializer` code. For more information, see [Serialization](#)

optimization.

To specify options by using [JsonSerializerOptions](#):

- Create an instance of `JsonSerializerOptions`.
- Create an instance of your class that derives from [JsonSerializerContext](#), and pass the `JsonSerializerOptions` instance to the constructor.
- Call serialization or deserialization methods of `JsonSerializer` that take a context instance or `TypeInfo< TValue >`.

Here's an example context class followed by serialization and deserialization example code:

```
[JsonSerializable(typeof(WeatherForecast))]
internal partial class OptionsExampleContext : JsonSerializerContext
{}
```

```
jsonString = JsonSerializer.Serialize(
    weatherForecast,
    typeof(WeatherForecast),
    new OptionsExampleContext(
        new JsonSerializerOptions(JsonSerializerDefaults.Web)));
```

```
weatherForecast = JsonSerializer.Deserialize(
    jsonString,
    typeof(WeatherForecast),
    new OptionsExampleContext(
        new JsonSerializerOptions(JsonSerializerDefaults.Web)))
    as WeatherForecast;
```

Here are the preceding examples in a complete program:

```

using System.Text.Json;
using System.Text.Json.Serialization;

namespace JsonSerializerOptionsExample
{
    public class WeatherForecast
    {
        public DateTime Date { get; set; }
        public int TemperatureCelsius { get; set; }
        public string? Summary { get; set; }
    }

    [JsonSerializable(typeof(WeatherForecast))]
    internal partial class OptionsExampleContext : JsonSerializerContext
    {
    }

    public class Program
    {
        public static void Main()
        {
            string jsonString =
@"
{
    ""date"": ""2019-08-01T00:00:00"",
    ""temperatureCelsius"": 25,
    ""summary"": ""Hot""
}
";
            WeatherForecast? weatherForecast;

            weatherForecast = JsonSerializer.Deserialize(
                jsonString,
                typeof(WeatherForecast),
                new OptionsExampleContext(
                    new JsonSerializerOptions(JsonSerializerDefaults.Web))
                as WeatherForecast;
            Console.WriteLine($"Date={weatherForecast?.Date}");
            // output:
            //Date=8/1/2019 12:00:00 AM

            jsonString = JsonSerializer.Serialize(
                weatherForecast,
                typeof(WeatherForecast),
                new OptionsExampleContext(
                    new JsonSerializerOptions(JsonSerializerDefaults.Web)));
            Console.WriteLine(jsonString);
            // output:
            //{"date":"2019-08-01T00:00:00","temperatureCelsius":25,"summary":"Hot"}
        }
    }
}

```

Source generation support in ASP.NET Core

- In Blazor apps:

Use overloads of [HttpClientJsonExtensions.GetFromJsonAsync](#) and [HttpClientJsonExtensions.PostAsJsonAsync](#) extension methods that take a source generation context or `TypeInfo< TValue >`.

- In Razor Pages, MVC, SignalR, and Web API apps:

Use the [AddContext](#) method of [JsonSerializerOptions](#), as shown in the following example:

```
[JsonSerializable(typeof(WeatherForecast[]))]  
internal partial class MyJsonContext : JsonSerializerContext { }
```

```
services.AddControllers().AddJsonOptions(options =>  
    options.JsonSerializerOptions.AddContext<MyJsonContext>());
```

See also

- [Try the new System.Text.Json source generator](#)
- [JSON serialization and deserialization in .NET - overview](#)
- [How to use the library](#)
- [Instantiate JsonSerializerOptions instances](#)
- [Enable case-insensitive matching](#)
- [Customize property names and values](#)
- [Ignore properties](#)
- [Allow invalid JSON](#)
- [Handle overflow JSON or use JsonElement or JsonNode](#)
- [Preserve references and handle circular references](#)
- [Deserialize to immutable types and non-public accessors](#)
- [Polymorphic serialization](#)
- [Migrate from Newtonsoft.Json to System.Text.Json](#)
- [Customize character encoding](#)
- [Use DOM, Utf8JsonReader, and Utf8JsonWriter](#)
- [Write custom converters for JSON serialization](#)
- [DateTime and DateTimeOffset support](#)
- [Supported collection types in System.Text.Json](#)
- [System.Text.Json API reference](#)
- [System.Text.Json.Serialization API reference](#)

How to write custom converters for JSON serialization (marshalling) in .NET

9/20/2022 • 27 minutes to read • [Edit Online](#)

This article shows how to create custom converters for the JSON serialization classes that are provided in the `System.Text.Json` namespace. For an introduction to `System.Text.Json`, see [How to serialize and deserialize JSON in .NET](#).

A *converter* is a class that converts an object or a value to and from JSON. The `System.Text.Json` namespace has built-in converters for most primitive types that map to JavaScript primitives. You can write custom converters:

- To override the default behavior of a built-in converter. For example, you might want `DateTime` values to be represented by mm/dd/yyyy format. By default, ISO 8601-1:2019 is supported, including the RFC 3339 profile. For more information, see [DateTime and DateTimeOffset support in System.Text.Json](#).
- To support a custom value type. For example, a `PhoneNumber` struct.

You can also write custom converters to customize or extend `System.Text.Json` with functionality not included in the current release. The following scenarios are covered later in this article:

- [Deserialize inferred types to object properties](#).
- [Support polymorphic deserialization](#).
- [Support round-trip for Stack<T>](#).
- [Support enum string value deserialization](#).
- [Use default system converter](#).
- [Deserialize inferred types to object properties](#).
- [Support polymorphic deserialization](#).
- [Support round-trip for Stack<T>](#).
- [Support enum string value deserialization](#).
- [Deserialize inferred types to object properties](#).
- [Support Dictionary with non-string key](#).
- [Support polymorphic deserialization](#).
- [Support round-trip for Stack<T>](#).

In the code you write for a custom converter, be aware of the substantial performance penalty for using new `JsonSerializerOptions` instances. For more information, see [Reuse JsonSerializerOptions instances](#).

Visual Basic can't be used to write custom converters but can call converters that are implemented in C# libraries. For more information, see [Visual Basic support](#).

Custom converter patterns

There are two patterns for creating a custom converter: the basic pattern and the factory pattern. The factory pattern is for converters that handle type `Enum` or open generics. The basic pattern is for non-generic and closed generic types. For example, converters for the following types require the factory pattern:

- `Dictionary< TKey, TValue >`
- `Enum`

- [List<T>](#)

Some examples of types that can be handled by the basic pattern include:

- [Dictionary<int, string>](#)
- [WeekdaysEnum](#)
- [List<DateTimeOffset>](#)
- [DateTime](#)
- [Int32](#)

The basic pattern creates a class that can handle one type. The factory pattern creates a class that determines, at run time, which specific type is required and dynamically creates the appropriate converter.

Sample basic converter

The following sample is a converter that overrides default serialization for an existing data type. The converter uses mm/dd/yyyy format for [DateTimeOffset](#) properties.

```
using System.Globalization;
using System.Text.Json;
using System.Text.Json.Serialization;

namespace SystemTextJsonSamples
{
    public class DateTimeOffsetJsonConverter : JsonConverter<DateTimeOffset>
    {
        public override DateTimeOffset Read(
            ref Utf8JsonReader reader,
            Type typeToConvert,
            JsonSerializerOptions options) =>
            DateTimeOffset.ParseExact(reader.GetString()!,
                "MM/dd/yyyy", CultureInfo.InvariantCulture);

        public override void Write(
            Utf8JsonWriter writer,
            DateTimeOffset dateTimeValue,
            JsonSerializerOptions options) =>
            writer.WriteStringValue(dateTimeValue.ToString(
                "MM/dd/yyyy", CultureInfo.InvariantCulture));
    }
}
```

Sample factory pattern converter

The following code shows a custom converter that works with [Dictionary<Enum, TValue>](#). The code follows the factory pattern because the first generic type parameter is [Enum](#) and the second is open. The [CanConvert](#) method returns [true](#) only for a [Dictionary](#) with two generic parameters, the first of which is an [Enum](#) type. The inner converter gets an existing converter to handle whichever type is provided at run time for [TValue](#).

```
using System.Reflection;
using System.Text.Json;
using System.Text.Json.Serialization;

namespace SystemTextJsonSamples
{
    public class DictionaryTKeyEnumTValueConverterFactory : JsonConverterFactory
    {
        public override bool CanConvert(Type typeToConvert)
        {
            if (!typeToConvert.IsGenericType)
                return false;
            var typeInfo = typeToConvert.GetGenericTypeDefinition();
            return typeInfo == typeof(Dictionary<IConvertible, object>);
        }
    }
}
```

```

        ...
    }

    return false;
}

if (typeToConvert.GetGenericTypeDefinition() != typeof(Dictionary<,>))
{
    return false;
}

return typeToConvert.GetGenericArguments()[0].IsEnum;
}

public override JsonConverter CreateConverter(
    Type type,
    JsonSerializerOptions options)
{
    Type keyType = type.GetGenericArguments()[0];
    Type valueType = type.GetGenericArguments()[1];

    JsonConverter converter = (JsonConverter)Activator.CreateInstance(
        typeof(DictionaryEnumConverterInner<,>).MakeGenericType(
            new Type[] { keyType, valueType }),
        BindingFlags.Instance | BindingFlags.Public,
        binder: null,
        args: new object[] { options },
        culture: null)!;

    return converter;
}

private class DictionaryEnumConverterInner<TKey, TValue> :
    JsonConverter<Dictionary<TKey, TValue>> where TKey : struct, Enum
{
    private readonly JsonConverter<TValue> _valueConverter;
    private readonly Type _keyType;
    private readonly Type _valueType;

    public DictionaryEnumConverterInner(JsonSerializerOptions options)
    {
        // For performance, use the existing converter if available.
        _valueConverter = (JsonConverter<TValue>)options
            .GetConverter(typeof(TValue));

        // Cache the key and value types.
        _keyType = typeof(TKey);
        _valueType = typeof(TValue);
    }

    public override Dictionary<TKey, TValue> Read(
        ref Utf8JsonReader reader,
        Type typeToConvert,
        JsonSerializerOptions options)
    {
        if (reader.TokenType != JsonTokenType.StartObject)
        {
            throw new JsonException();
        }

        var dictionary = new Dictionary<TKey, TValue>();

        while (reader.Read())
        {
            if (reader.TokenType == JsonTokenType.EndObject)
            {
                return dictionary;
            }

            // Get the key.
            if (reader.TokenType != JsonTokenType.PropertyName)

```

The preceding code is the same as what is shown in the [Support Dictionary with non-string key](#) later in this article.

Steps to follow the basic pattern

The following steps explain how to create a converter by following the basic pattern:

- Create a class that derives from `JsonConverter<T>` where `T` is the type to be serialized and deserialized.
- Override the `Read` method to deserialize the incoming JSON and convert it to type `T`. Use the `Utf8JsonReader` that is passed to the method to read the JSON. You don't have to worry about handling partial data, as the serializer passes all the data for the current JSON scope. So it isn't necessary to call `Skip` or `TrySkip` or to validate that `Read` returns `true`.
- Override the `Write` method to serialize the incoming object of type `T`. Use the `Utf8JsonWriter` that is passed to the method to write the JSON.
- Override the `CanConvert` method only if necessary. The default implementation returns `true` when the type to convert is of type `T`. Therefore, converters that support only type `T` don't need to override this method. For an example of a converter that does need to override this method, see the [polymorphic deserialization](#) section later in this article.

You can refer to the [built-in converters source code](#) as reference implementations for writing custom converters.

Steps to follow the factory pattern

The following steps explain how to create a converter by following the factory pattern:

- Create a class that derives from `JsonConverterFactory`.
- Override the `CanConvert` method to return true when the type to convert is one that the converter can handle. For example, if the converter is for `List<T>` it might only handle `List<int>`, `List<string>`, and `List<DateTime>`.
- Override the `CreateConverter` method to return an instance of a converter class that will handle the type-to-convert that is provided at run time.
- Create the converter class that the `CreateConverter` method instantiates.

The factory pattern is required for open generics because the code to convert an object to and from a string isn't the same for all types. A converter for an open generic type (`List<T>`, for example) has to create a converter for a closed generic type (`List<DateTime>`, for example) behind the scenes. Code must be written to handle each closed-generic type that the converter can handle.

The `Enum` type is similar to an open generic type: a converter for `Enum` has to create a converter for a specific `Enum` (`WeekdaysEnum`, for example) behind the scenes.

The use of `Utf8JsonReader` in the `Read` method

If your converter is converting a JSON object, the `Utf8JsonReader` will be positioned on the begin object token when the `Read` method begins. You must then read through all the tokens in that object and exit the method with the reader positioned on **the corresponding end object token**. If you read beyond the end of the object, or if you stop before reaching the corresponding end token, you get a `JsonException` exception indicating that:

The converter 'ConverterName' read too much or not enough.

For an example, see the preceding factory pattern sample converter. The `Read` method starts by verifying that the reader is positioned on a start object token. It reads until it finds that it is positioned on the next end object token. It stops on the next end object token because there are no intervening start object tokens that would indicate an object within the object. The same rule about begin token and end token applies if you are converting an array. For an example, see the `Stack<T>` sample converter later in this article.

Error handling

The serializer provides special handling for exception types [JsonException](#) and [NotSupportedException](#).

JsonException

If you throw a `JsonException` without a message, the serializer creates a message that includes the path to the part of the JSON that caused the error. For example, the statement `throw new JsonException()` produces an error message like the following example:

```
Unhandled exception. System.Text.Json.JsonException:  
The JSON value could not be converted to System.Object.  
Path: $.Date | LineNumber: 1 | BytePositionInLine: 37.
```

If you do provide a message (for example, `throw new JsonException("Error occurred")`), the serializer still sets the [Path](#), [LineNumber](#), and [BytePositionInLine](#) properties.

NotSupportedException

If you throw a `NotSupportedException`, you always get the path information in the message. If you provide a message, the path information is appended to it. For example, the statement

```
throw new NotSupportedException("Error occurred.")
```

```
Error occurred. The unsupported member type is located on type  
'System.Collections.Generic.Dictionary`2[Samples.SummaryWords,System.Int32]'.  
Path: $.TemperatureRanges | LineNumber: 4 | BytePositionInLine: 24
```

When to throw which exception type

When the JSON payload contains tokens that are not valid for the type being deserialized, throw a `JsonException`.

When you want to disallow certain types, throw a `NotSupportedException`. This exception is what the serializer automatically throws for types that are not supported. For example, `System.Type` is not supported for security reasons, so an attempt to deserialize it results in a `NotSupportedException`.

You can throw other exceptions as needed, but they don't automatically include JSON path information.

Register a custom converter

Register a custom converter to make the `Serialize` and `Deserialize` methods use it. Choose one of the following approaches:

- Add an instance of the converter class to the [JsonSerializerOptions.Converters](#) collection.
- Apply the [\[JsonConverter\]](#) attribute to the properties that require the custom converter.
- Apply the [\[JsonConverter\]](#) attribute to a class or a struct that represents a custom value type.

Registration sample - Converters collection

Here's an example that makes the [DateTimeOffsetJsonConverter](#) the default for properties of type `DateTimeOffset`:

```

var serializeOptions = new JsonSerializerOptions
{
    WriteIndented = true,
    Converters =
    {
        new DateTimeOffsetJsonConverter()
    }
};

jsonString = JsonSerializer.Serialize(weatherForecast, serializeOptions);

```

Suppose you serialize an instance of the following type:

```

public class WeatherForecast
{
    public DateTimeOffset Date { get; set; }
    public int TemperatureCelsius { get; set; }
    public string? Summary { get; set; }
}

```

Here's an example of JSON output that shows the custom converter was used:

```
{
    "Date": "08/01/2019",
    "TemperatureCelsius": 25,
    "Summary": "Hot"
}
```

The following code uses the same approach to deserialize using the custom `DateTimeOffset` converter:

```

var deserializeOptions = new JsonSerializerOptions();
deserializeOptions.Converters.Add(new DateTimeOffsetJsonConverter());
weatherForecast = JsonSerializer.Deserialize<WeatherForecast>(jsonString, deserializeOptions)!;

```

Registration sample - [JsonConverter] on a property

The following code selects a custom converter for the `Date` property:

```

public class WeatherForecastWithConverterAttribute
{
    [JsonConverter(typeof(DateTimeOffsetJsonConverter))]
    public DateTimeOffset Date { get; set; }
    public int TemperatureCelsius { get; set; }
    public string? Summary { get; set; }
}

```

The code to serialize `WeatherForecastWithConverterAttribute` doesn't require the use of `JsonSerializerOptions.Converters`:

```

var serializeOptions = new JsonSerializerOptions
{
    WriteIndented = true
};
jsonString = JsonSerializer.Serialize(weatherForecast, serializeOptions);

```

The code to deserialize also doesn't require the use of `Converters`:

```
weatherForecast = JsonSerializer.Deserialize<WeatherForecastWithConverterAttribute>(jsonString);
```

Registration sample - [JsonConverter] on a type

Here's code that creates a struct and applies the `[JsonConverter]` attribute to it:

```
using System.Text.Json.Serialization;

namespace SystemTextJsonSamples
{
    [JsonConverter(typeof(TemperatureConverter))]
    public struct Temperature
    {
        public Temperature(int degrees, bool celsius)
        {
            Degrees = degrees;
            IsCelsius = celsius;
        }

        public int Degrees { get; }
        public bool IsCelsius { get; }
        public bool IsFahrenheit => !IsCelsius;

        public override string ToString() =>
            $"{Degrees}{(IsCelsius ? "C" : "F")}";
    }

    public static Temperature Parse(string input)
    {
        int degrees = int.Parse(input.Substring(0, input.Length - 1));
        bool celsius = input.Substring(input.Length - 1) == "C";

        return new Temperature(degrees, celsius);
    }
}
```

Here's the custom converter for the preceding struct:

```
using System.Text.Json;
using System.Text.Json.Serialization;

namespace SystemTextJsonSamples
{
    public class TemperatureConverter : JsonConverter<Temperature>
    {
        public override Temperature Read(
            ref Utf8JsonReader reader,
            Type typeToConvert,
            JsonSerializerOptions options) =>
            Temperature.Parse(reader.GetString()!);

        public override void Write(
            Utf8JsonWriter writer,
            Temperature temperature,
            JsonSerializerOptions options) =>
            writer.WriteStringValue(temperature.ToString());
    }
}
```

The `[JsonConverter]` attribute on the struct registers the custom converter as the default for properties of type `Temperature`. The converter is automatically used on the `TemperatureCelsius` property of the following type

when you serialize or deserialize it:

```
public class WeatherForecastWithTemperatureStruct
{
    public DateTimeOffset Date { get; set; }
    public Temperature TemperatureCelsius { get; set; }
    public string? Summary { get; set; }
}
```

Converter registration precedence

During serialization or deserialization, a converter is chosen for each JSON element in the following order, listed from highest priority to lowest:

- `[JsonConverter]` applied to a property.
- A converter added to the `Converters` collection.
- `[JsonConverter]` applied to a custom value type or POCO.

If multiple custom converters for a type are registered in the `Converters` collection, the first converter that returns true for `CanConvert` is used.

A built-in converter is chosen only if no applicable custom converter is registered.

Converter samples for common scenarios

The following sections provide converter samples that address some common scenarios that built-in functionality doesn't handle.

- [Deserialize inferred types to object properties.](#)
- [Support polymorphic deserialization.](#)
- [Support round-trip for Stack<T>.](#)
- [Support enum string value deserialization.](#)
- [Use default system converter.](#)
- [Deserialize inferred types to object properties.](#)
- [Support polymorphic deserialization.](#)
- [Support round-trip for Stack<T>.](#)
- [Support enum string value deserialization.](#)
- [Deserialize inferred types to object properties.](#)
- [Support Dictionary with non-string key.](#)
- [Support polymorphic deserialization.](#)
- [Support round-trip for Stack<T>.](#)

For a sample `DataTable` converter, see [Supported collection types](#).

Deserialize inferred types to object properties

When deserializing to a property of type `object`, a `JsonElement` object is created. The reason is that the deserializer doesn't know what CLR type to create, and it doesn't try to guess. For example, if a JSON property has "true", the deserializer doesn't infer that the value is a `Boolean`, and if an element has "01/01/2019", the deserializer doesn't infer that it's a `DateTime`.

Type inference can be inaccurate. If the deserializer parses a JSON number that has no decimal point as a `long`, that might result in out-of-range issues if the value was originally serialized as a `ulong` or `BigInteger`. Parsing a

number that has a decimal point as a `double` might lose precision if the number was originally serialized as a `decimal`.

For scenarios that require type inference, the following code shows a custom converter for `object` properties. The code converts:

- `true` and `false` to `Boolean`
- Numbers without a decimal to `long`
- Numbers with a decimal to `double`
- Dates to `DateTime`
- Strings to `string`
- Everything else to `JsonElement`

```
using System.Text.Json;
using System.Text.Json.Serialization;

namespace CustomConverterInferredTypesToObject
{
    public class ObjectToInferredTypesConverter : JsonConverter<object>
    {
        public override object Read(
            ref Utf8JsonReader reader,
            Type typeToConvert,
            JsonSerializerOptions options) => reader.TokenType switch
        {
            JsonTokenType.True => true,
            JsonTokenType.False => false,
            JsonTokenType.Number when reader.TryGetInt64(out long l) => l,
            JsonTokenType.Number => reader.GetDouble(),
            JsonTokenType.String when reader.TryGetDateTime(out DateTime datetime) => datetime,
            JsonTokenType.String => reader.GetString()!,
            _ => JsonDocument.ParseValue(ref reader).RootElement.Clone()
        };

        public override void Write(
            Utf8JsonWriter writer,
            object objectToWrite,
            JsonSerializerOptions options) =>
            JsonSerializer.Serialize(writer, objectToWrite, objectToWrite.GetType(), options);
    }

    public class WeatherForecast
    {
        public object? Date { get; set; }
        public object? TemperatureCelsius { get; set; }
        public object? Summary { get; set; }
    }

    public class Program
    {
        public static void Main()
        {
            string jsonString = @"{
                ""Date"": ""2019-08-01T00:00:00-07:00"",
                ""TemperatureCelsius"": 25,
                ""Summary"": ""Hot""
            }";

            WeatherForecast weatherForecast = JsonSerializer.Deserialize<WeatherForecast>(jsonString)!;
            Console.WriteLine($"Type of Date property no converter = {weatherForecast.Date!.GetType()}");

            var options = new JsonSerializerOptions();
            options.WriteIndented = true;
            options.Converters.Add(new ObjectToInferredTypesConverter());
        }
    }
}
```

```

        weatherForecast = JsonSerializer.Deserialize<WeatherForecast>(jsonString, options)!;
        Console.WriteLine($"Type of Date property with converter = {weatherForecast.Date!.GetType()}");

        Console.WriteLine(JsonSerializer.Serialize(weatherForecast, options));
    }
}

// Produces output like the following example:
//
//Type of Date property no converter = System.Text.Json.JsonElement
//Type of Date property with converter = System.DateTime
//{
//  "Date": "2019-08-01T00:00:00-07:00",
//  "TemperatureCelsius": 25,
//  "Summary": "Hot"
//}

```

The example shows the converter code and a `WeatherForecast` class with `object` properties. The `Main` method deserializes a JSON string into a `WeatherForecast` instance, first without using the converter, and then using the converter. The console output shows that without the converter the run time type for the `Date` property is `JsonElement`; with the converter, the run time type is `DateTime`.

The [unit tests folder](#) in the `System.Text.Json.Serialization` namespace has more examples of custom converters that handle deserialization to `object` properties.

```

using System.Text.Json;
using System.Text.Json.Serialization;

namespace SystemTextJsonSamples
{
    public class ObjectToInferredTypesConverter
        : JsonConverter<object>
    {
        public override object Read(
            ref Utf8JsonReader reader,
            Type typeToConvert,
            JsonSerializerOptions options) => reader.TokenType switch
        {
            JsonTokenType.True => true,
            JsonTokenType.False => false,
            JsonTokenType.Number when reader.TryGetInt64(out long l) => l,
            JsonTokenType.Number => reader.GetDouble(),
            JsonTokenType.String when reader.TryGetDateTime(out DateTime datetime) => datetime,
            JsonTokenType.String => reader.GetString()!,
            _ => JsonDocument.ParseValue(ref reader).RootElement.Clone()
        };

        public override void Write(
            Utf8JsonWriter writer,
            object objectToWrite,
            JsonSerializerOptions options) =>
            throw new InvalidOperationException("Should not get here.");
    }
}

```

The following code registers the converter:

```
var deserializeOptions = new JsonSerializerOptions
{
    Converters =
    {
        new ObjectToInferredTypesConverter()
    }
};
```

Here's an example type with `object` properties:

```
public class WeatherForecastWithObjectProperties
{
    public object? Date { get; set; }
    public object? TemperatureCelsius { get; set; }
    public object? Summary { get; set; }
}
```

The following example of JSON to deserialize contains values that will be deserialized as `DateTime`, `long`, and `string`:

```
{
    "Date": "2019-08-01T00:00:00-07:00",
    "TemperatureCelsius": 25,
    "Summary": "Hot",
}
```

Without the custom converter, deserialization puts a `JsonElement` in each property.

The [unit tests folder](#) in the `System.Text.Json.Serialization` namespace has more examples of custom converters that handle deserialization to `object` properties.

Support Dictionary with non-string key

The built-in support for dictionary collections is for `Dictionary<string, TValue>`. That is, the key must be a string. To support a dictionary with an integer or some other type as the key, a custom converter is required.

The following code shows a custom converter that works with `Dictionary<Enum, TValue>`:

```
using System.Reflection;
using System.Text.Json;
using System.Text.Json.Serialization;

namespace SystemTextJsonSamples
{
    public class DictionaryTKeyEnumTValueConverter : JsonConverterFactory
    {
        public override bool CanConvert(Type typeToConvert)
        {
            if (!typeToConvert.IsGenericType)
            {
                return false;
            }

            if (typeToConvert.GetGenericTypeDefinition() != typeof(Dictionary<, >))
            {
                return false;
            }

            return typeToConvert.GetGenericArguments()[0].IsEnum;
        }
    }
}
```

```

public override JsonConverter CreateConverter(
    Type type,
    JsonSerializerOptions options)
{
    Type keyType = type.GetGenericArguments()[0];
    Type valueType = type.GetGenericArguments()[1];

    JsonConverter converter = (JsonConverter)Activator.CreateInstance(
        typeof(DictionaryEnumConverterInner<,>).MakeGenericType(
            new Type[] { keyType, valueType })),
        BindingFlags.Instance | BindingFlags.Public,
        binder: null,
        args: new object[] { options },
        culture: null)!;

    return converter;
}

private class DictionaryEnumConverterInner<TKey, TValue> :
    JsonConverter<Dictionary<TKey, TValue>> where TKey : struct, Enum
{
    private readonly JsonConverter<TValue> _valueConverter;
    private readonly Type _keyType;
    private readonly Type _valueType;

    public DictionaryEnumConverterInner(JsonSerializerOptions options)
    {
        // For performance, use the existing converter if available.
        _valueConverter = (JsonConverter<TValue>)options
            .GetConverter(typeof(TValue));

        // Cache the key and value types.
        _keyType = typeof(TKey);
        _valueType = typeof(TValue);
    }

    public override Dictionary<TKey, TValue> Read(
        ref Utf8JsonReader reader,
        Type typeToConvert,
        JsonSerializerOptions options)
    {
        if (reader.TokenType != JsonTokenType.StartObject)
        {
            throw new JsonException();
        }

        var dictionary = new Dictionary<TKey, TValue>();

        while (reader.Read())
        {
            if (reader.TokenType == JsonTokenType.EndObject)
            {
                return dictionary;
            }

            // Get the key.
            if (reader.TokenType != JsonTokenType.PropertyName)
            {
                throw new JsonException();
            }

            string? propertyName = reader.GetString();

            // For performance, parse with ignoreCase:false first.
            if (!Enum.TryParse(propertyName, ignoreCase: false, out TKey key) &&
                !Enum.TryParse(propertyName, ignoreCase: true, out key))
            {
                throw new JsonException(
                    $"Unable to convert \'{propertyName}\' to Enum \'{_keyType}\'.");
            }

            TValue value;
            if (_valueConverter.CanConvert(_valueType))
            {
                value = _valueConverter.Read(
                    ref reader,
                    _valueType,
                    options);
            }
            else
            {
                value = Activator.CreateInstance(_valueType);
                _valueConverter.Read(
                    ref reader,
                    value.GetType(),
                    options);
            }

            dictionary.Add(key, value);
        }
    }

    public void Write(
        Dictionary<TKey, TValue> dictionary,
        Utf8JsonWriter writer,
        JsonSerializerOptions options)
    {
        writer.WriteStartObject();
        foreach (var item in dictionary)
        {
            writer.WritePropertyName(item.Key.ToString());
            _valueConverter.Write(item.Value, writer, options);
        }
        writer.WriteEndObject();
    }
}

```

```

        }

        // Get the value.
        TValue value;
        if (_valueConverter != null)
        {
            reader.Read();
            value = _valueConverter.Read(ref reader, _valueType, options)!;
        }
        else
        {
            value = JsonSerializer.Deserialize<TValue>(ref reader, options)!;
        }

        // Add to dictionary.
        dictionary.Add(key, value);
    }

    throw new JsonException();
}

public override void Write(
    Utf8JsonWriter writer,
    Dictionary< TKey, TValue> dictionary,
    JsonSerializerOptions options)
{
    writer.WriteStartObject();

    foreach ((TKey key, TValue value) in dictionary)
    {
        var propertyName = key.ToString();
        writer.WritePropertyName
            (options.PropertyNamingPolicy?.ConvertName(propertyName) ?? propertyName);

        if (_valueConverter != null)
        {
            _valueConverter.Write(writer, value, options);
        }
        else
        {
            JsonSerializer.Serialize(writer, value, options);
        }
    }

    writer.WriteEndObject();
}
}
}
}

```

The following code registers the converter:

```

var serializeOptions = new JsonSerializerOptions();
serializeOptions.Converters.Add(new DictionaryTKeyEnumTValueConverter());

```

The converter can serialize and deserialize the `TemperatureRanges` property of the following class that uses the following `Enum`:

```

public class WeatherForecastWithEnumDictionary
{
    public DateTimeOffset Date { get; set; }
    public int TemperatureCelsius { get; set; }
    public string? Summary { get; set; }
    public Dictionary<SummaryWordsEnum, int>? TemperatureRanges { get; set; }
}

public enum SummaryWordsEnum
{
    Cold, Hot
}

```

The JSON output from serialization looks like the following example:

```
{
    "Date": "2019-08-01T00:00:00-07:00",
    "TemperatureCelsius": 25,
    "Summary": "Hot",
    "TemperatureRanges": {
        "Cold": 20,
        "Hot": 40
    }
}
```

The [unit tests folder](#) in the `System.Text.Json.Serialization` namespace has more examples of custom converters that handle non-string-key dictionaries.

Support polymorphic deserialization

Built-in features provide a limited range of [polymorphic serialization](#) but no support for deserialization at all. Deserialization requires a custom converter.

Suppose, for example, you have a `Person` abstract base class, with `Employee` and `Customer` derived classes. Polymorphic deserialization means that at design time you can specify `Person` as the deserialization target, and `Customer` and `Employee` objects in the JSON are correctly deserialized at run time. During deserialization, you have to find clues that identify the required type in the JSON. The kinds of clues available vary with each scenario. For example, a discriminator property might be available or you might have to rely on the presence or absence of a particular property. The current release of `System.Text.Json` doesn't provide attributes to specify how to handle polymorphic deserialization scenarios, so custom converters are required.

The following code shows a base class, two derived classes, and a custom converter for them. The converter uses a discriminator property to do polymorphic deserialization. The type discriminator isn't in the class definitions but is created during serialization and is read during deserialization.

IMPORTANT

The example code requires JSON object name/value pairs to stay in order, which is not a standard requirement of JSON.

```

public class Person
{
    public string? Name { get; set; }
}

public class Customer : Person
{
    public decimal CreditLimit { get; set; }
}

public class Employee : Person
{
    public string? OfficeNumber { get; set; }
}

```

```

using System.Text.Json;
using System.Text.Json.Serialization;

namespace SystemTextJsonSamples
{
    public class PersonConverterWithTypeDiscriminator : JsonConverter<Person>
    {
        enum TypeDiscriminator
        {
            Customer = 1,
            Employee = 2
        }

        public override bool CanConvert(Type typeToConvert) =>
            typeof(Person).IsAssignableFrom(typeToConvert);

        public override Person Read(
            ref Utf8JsonReader reader, Type typeToConvert, JsonSerializerOptions options)
        {
            if (reader.TokenType != JsonTokenType.StartObject)
            {
                throw new JsonException();
            }

            reader.Read();
            if (reader.TokenType != JsonTokenType.PropertyName)
            {
                throw new JsonException();
            }

            string? propertyName = reader.GetString();
            if (propertyName != "TypeDiscriminator")
            {
                throw new JsonException();
            }

            reader.Read();
            if (reader.TokenType != JsonTokenType.Number)
            {
                throw new JsonException();
            }

            TypeDiscriminator typeDiscriminator = (TypeDiscriminator)reader.GetInt32();
            Person person = typeDiscriminator switch
            {
                TypeDiscriminator.Customer => new Customer(),
                TypeDiscriminator.Employee => new Employee(),
                _ => throw new JsonException()
            };

            while (reader.Read())

```

```

    {
        if (reader.TokenType == JsonTokenType.EndObject)
        {
            return person;
        }

        if (reader.TokenType == JsonTokenType.PropertyName)
        {
            propertyName = reader.GetString();
            reader.Read();
            switch (propertyName)
            {
                case "CreditLimit":
                    decimal creditLimit = reader.GetDecimal();
                    ((Customer)person).CreditLimit = creditLimit;
                    break;
                case "OfficeNumber":
                    string? officeNumber = reader.GetString();
                    ((Employee)person).OfficeNumber = officeNumber;
                    break;
                case "Name":
                    string? name = reader.GetString();
                    person.Name = name;
                    break;
            }
        }
    }

    throw new JsonException();
}

public override void Write(
    Utf8JsonWriter writer, Person person, JsonSerializerOptions options)
{
    writer.WriteStartObject();

    if (person is Customer customer)
    {
        writer.WriteNumber("TypeDiscriminator", (int)TypeDiscriminator.Customer);
        writer.WriteNumber("CreditLimit", customer.CreditLimit);
    }
    else if (person is Employee employee)
    {
        writer.WriteNumber("TypeDiscriminator", (int)TypeDiscriminator.Employee);
        writer.WriteString("OfficeNumber", employee.OfficeNumber);
    }

    writer.WriteString("Name", person.Name);

    writer.WriteEndObject();
}
}
}

```

The following code registers the converter:

```

var serializeOptions = new JsonSerializerOptions();
serializeOptions.Converters.Add(new PersonConverterWithTypeDiscriminator());

```

The converter can deserialize JSON that was created by using the same converter to serialize, for example:

```
[  
 {  
   "TypeDiscriminator": 1,  
   "CreditLimit": 10000,  
   "Name": "John"  
 },  
 {  
   "TypeDiscriminator": 2,  
   "OfficeNumber": "555-1234",  
   "Name": "Nancy"  
 }  
 ]
```

The converter code in the preceding example reads and writes each property manually. An alternative is to call `Deserialize` or `Serialize` to do some of the work. For an example, see [this StackOverflow post](#).

An alternative way to do polymorphic deserialization

You can call `Deserialize` in the `Read` method:

- Make a clone of the `Utf8JsonReader` instance. Since `Utf8JsonReader` is a struct, this just requires an assignment statement.
- Use the clone to read through the discriminator tokens.
- Call `Deserialize` using the original `Reader` instance once you know the type you need. You can call `Deserialize` because the original `Reader` instance is still positioned to read the begin object token.

A disadvantage of this method is you can't pass in the original options instance that registers the converter to `Deserialize`. Doing so would cause a stack overflow, as explained in [Required properties](#). The following example shows a `Read` method that uses this alternative:

```

public override Person Read(
    ref Utf8JsonReader reader, Type typeToConvert, JsonSerializerOptions options)
{
    Utf8JsonReader readerClone = reader;

    if (readerClone.TokenType != JsonTokenType.StartObject)
    {
        throw new JsonException();
    }

    readerClone.Read();
    if (readerClone.TokenType != JsonTokenType.PropertyName)
    {
        throw new JsonException();
    }

    string? propertyName = readerClone.GetString();
    if (propertyName != "TypeDiscriminator")
    {
        throw new JsonException();
    }

    readerClone.Read();
    if (readerClone.TokenType != JsonTokenType.Number)
    {
        throw new JsonException();
    }

    TypeDiscriminator typeDiscriminator = (TypeDiscriminator)readerClone.GetInt32();
    Person person = typeDiscriminator switch
    {
        TypeDiscriminator.Customer => JsonSerializer.Deserialize<Customer>(ref reader)!,
        TypeDiscriminator.Employee => JsonSerializer.Deserialize<Employee>(ref reader)!,
        _ => throw new JsonException()
    };
    return person;
}

```

Support round trip for Stack<T>

If you deserialize a JSON string into a [Stack<T>](#) object and then serialize that object, the contents of the stack are in reverse order. This behavior applies to the following types and interface, and user-defined types that derive from them:

- [Stack](#)
- [Stack<T>](#)
- [ConcurrentStack<T>](#)
- [ImmutableStack<T>](#)
- [IImmutableStack<T>](#)

To support serialization and deserialization that retains the original order in the stack, a custom converter is required.

The following code shows a custom converter that enables round-tripping to and from [Stack<T>](#) objects:

```

using System.Diagnostics;
using System.Reflection;
using System.Text.Json;
using System.Text.Json.Serialization;

namespace SystemTextJsonSamples
{
    public class JsonConverterFactoryForStackOfT : JsonConverterFactory

```

```

{
    public override bool CanConvert(Type typeToConvert)
        => typeToConvert.IsGenericType
        && typeToConvert.GetGenericTypeDefinition() == typeof(Stack<>);

    public override JsonConverter CreateConverter(
        Type typeToConvert, JsonSerializerOptions options)
    {
        Debug.Assert(typeToConvert.IsGenericType &&
            typeToConvert.GetGenericTypeDefinition() == typeof(Stack<>));

        Type elementType = typeToConvert.GetGenericArguments()[0];

        JsonConverter converter = (JsonConverter)Activator.CreateInstance(
            typeof(JsonConverterForStackOfT<>)
                .MakeGenericType(new Type[] { elementType }),
            BindingFlags.Instance | BindingFlags.Public,
            binder: null,
            args: null,
            culture: null)!;

        return converter;
    }
}

public class JsonConverterForStackOfT<T> : JsonConverter<Stack<T>>
{
    public override Stack<T> Read(
        ref Utf8JsonReader reader, Type typeToConvert, JsonSerializerOptions options)
    {
        if (reader.TokenType != JsonTokenType.StartArray)
        {
            throw new JsonException();
        }
        reader.Read();

        var elements = new Stack<T>();

        while (reader.TokenType != JsonTokenType.EndArray)
        {
            elements.Push(JsonSerializer.Deserialize<T>(ref reader, options)!);

            reader.Read();
        }

        return elements;
    }

    public override void Write(
        Utf8JsonWriter writer, Stack<T> value, JsonSerializerOptions options)
    {
        writer.WriteStartArray();

        var reversed = new Stack<T>(value);

        foreach (T item in reversed)
        {
            JsonSerializer.Serialize(writer, item, options);
        }

        writer.WriteEndArray();
    }
}
}

```

The following code registers the converter:

```
var options = new JsonSerializerOptions
{
    Converters = { new JsonConverterFactoryForStackOfT() },
};
```

Support enum string value deserialization

By default, the built-in [JsonStringEnumConverter](#) can serialize and deserialize string values for enums. It works without a specified naming policy or with the [CamelCase](#) naming policy. It doesn't support other naming policies, such as snake case. For information about custom converter code that can support round-tripping to and from enum string values while using a snake case naming policy, see GitHub issue [dotnet/runtime #31619](#).

Use default system converter

In some scenarios, you might want to use the default system converter in a custom converter. To do that, get the system converter from the [JsonSerializerOptions.Default](#) property, as shown in the following example:

```
public class MyCustomConverter : JsonConverter<int>
{
    private readonly static JsonConverter<int> s_defaultConverter =
        (JsonConverter<int>)JsonSerializerOptions.Default.GetConverter(typeof(int));

    // Custom serialization logic
    public override void Write(
        Utf8JsonWriter writer, int value, JsonSerializerOptions options)
    {
        writer.WriteStringValue(value.ToString());
    }

    // Fall back to default deserialization logic
    public override int Read(
        ref Utf8JsonReader reader, Type typeToConvert, JsonSerializerOptions options)
    {
        return s_defaultConverter.Read(ref reader, typeToConvert, options);
    }
}
```

Handle null values

By default, the serializer handles null values as follows:

- For reference types and [Nullable<T>](#) types:
 - It does not pass `null` to custom converters on serialization.
 - It does not pass `JsonTokenType.Null` to custom converters on deserialization.
 - It returns a `null` instance on deserialization.
 - It writes `null` directly with the writer on serialization.
- For non-nullable value types:
 - It passes `JsonTokenType.Null` to custom converters on deserialization. (If no custom converter is available, a `JsonException` exception is thrown by the internal converter for the type.)

This null-handling behavior is primarily to optimize performance by skipping an extra call to the converter. In addition, it avoids forcing converters for nullable types to check for `null` at the start of every `Read` and `Write` method override.

To enable a custom converter to handle `null` for a reference or value type, override [JsonConverter<T>.HandleNull](#) to return `true`, as shown in the following example:

```

using System.Text.Json;
using System.Text.Json.Serialization;

namespace CustomConverterHandleNull
{
    public class Point
    {
        public int X { get; set; }
        public int Y { get; set; }

        [JsonConverter(typeof(DescriptionConverter))]
        public string? Description { get; set; }
    }

    public class DescriptionConverter : JsonConverter<string>
    {
        public override bool HandleNull => true;

        public override string Read(
            ref Utf8JsonReader reader,
            Type typeToConvert,
            JsonSerializerOptions options) =>
            reader.GetString() ?? "No description provided.";

        public override void Write(
            Utf8JsonWriter writer,
            string value,
            JsonSerializerOptions options) =>
            writer.WriteStringValue(value);
    }

    public class Program
    {
        public static void Main()
        {
            string json = @"""x"":1,""y"":2,""Description"":null};

            Point point = JsonSerializer.Deserialize<Point>(json)!;
            Console.WriteLine($"Description: {point.Description}");
        }
    }
}

// Produces output like the following example:
// 
//Description: No description provided.

```

Preserve references

By default, reference data is only cached for each call to `Serialize` or `Deserialize`. To persist references from one `Serialize` / `Deserialize` call to another one, root the `ReferenceResolver` instance in the call site of `Serialize` / `Deserialize`. The following code shows an example for this scenario:

- You write a custom converter for the `Company` type.
- You don't want to manually serialize the `Supervisor` property, which is an `Employee`. You want to delegate that to the serializer and you also want to preserve the references that you have already saved.

Here are the `Employee` and `Company` classes:

```

public class Employee
{
    public string? Name { get; set; }
    public Employee? Manager { get; set; }
    public List<Employee>? DirectReports { get; set; }
    public Company? Company { get; set; }
}

public class Company
{
    public string? Name { get; set; }
    public Employee? Supervisor { get; set; }
}

```

The converter looks like this:

```

class CompanyConverter : JsonConverter<Company>
{
    public override Company Read(ref Utf8JsonReader reader, Type typeToConvert, JsonSerializerOptions options)
    {
        throw new NotImplementedException();
    }

    public override void Write(Utf8JsonWriter writer, Company value, JsonSerializerOptions options)
    {
        writer.WriteStartObject();

        writer.WriteString("Name", value.Name);

        writer.WritePropertyName("Supervisor");
        JsonSerializer.Serialize(writer, value.Supervisor, options);

        writer.WriteEndObject();
    }
}

```

A class that derives from [ReferenceResolver](#) stores the references in a dictionary:

```

class MyReferenceResolver : ReferenceResolver
{
    private uint _referenceCount;
    private readonly Dictionary<string, object> _referenceIdToObjectMap = new ();
    private readonly Dictionary<object, string> _objectToReferenceIdMap = new
(ReferenceEqualityComparer.Instance);

    public override void AddReference(string referenceId, object value)
    {
        if (!_referenceIdToObjectMap.TryAdd(referenceId, value))
        {
            throw new JsonException();
        }
    }

    public override string GetReference(object value, out bool alreadyExists)
    {
        if (_objectToReferenceIdMap.TryGetValue(value, out string? referenceId))
        {
            alreadyExists = true;
        }
        else
        {
            _referenceCount++;
            referenceId = _referenceCount.ToString();
            _objectToReferenceIdMap.Add(value, referenceId);
            alreadyExists = false;
        }

        return referenceId;
    }

    public override object ResolveReference(string referenceId)
    {
        if (!_referenceIdToObjectMap.TryGetValue(referenceId, out object? value))
        {
            throw new JsonException();
        }

        return value;
    }
}

```

A class that derives from [ReferenceHandler](#) holds an instance of `MyReferenceResolver` and creates a new instance only when needed (in a method named `Reset` in this example):

```

class MyReferenceHandler : ReferenceHandler
{
    public MyReferenceHandler() => Reset();

    private ReferenceResolver? _rootedResolver;
    public override ReferenceResolver CreateResolver() => _rootedResolver!;
    public void Reset() => _rootedResolver = new MyReferenceResolver();

}

```

When the sample code calls the serializer, it uses a [JsonSerializerOptions](#) instance in which the [ReferenceHandler](#) property is set to an instance of `MyReferenceHandler`. When you follow this pattern, be sure to reset the `ReferenceResolver` dictionary when you're finished serializing, to keep it from growing forever.

```
var options = new JsonSerializerOptions();

options.Converters.Add(new CompanyConverter());
var myReferenceHandler = new MyReferenceHandler();
options.ReferenceHandler = myReferenceHandler;
options.DefaultIgnoreCondition = JsonIgnoreCondition.WhenWritingNull;
options.WriteIndented = true;

string str = JsonSerializer.Serialize(tyler, options);

// Reset after serializing to avoid out of bounds memory growth in the resolver.
myReferenceHandler.Reset();
```

The preceding example only does serialization, but a similar approach can be adopted for deserialization.

For information about how to preserve references, see [the .NET 5 version of this page](#).

Other custom converter samples

The [Migrate from Newtonsoft.Json to System.Text.Json](#) article contains additional samples of custom converters.

The [unit tests folder](#) in the `System.Text.Json.Serialization` source code includes other custom converter samples, such as:

- [Int32 converter that converts null to 0 on deserialize](#)
- [Int32 converter that allows both string and number values on deserialize](#)
- [Enum converter](#)
- [List<T> converter that accepts external data](#)
- [Long\[\] converter that works with a comma-delimited list of numbers](#)

If you need to make a converter that modifies the behavior of an existing built-in converter, you can get [the source code of the existing converter](#) to serve as a starting point for customization.

Additional resources

- [Source code for built-in converters](#)
- [System.Text.Json overview](#)
- [How to serialize and deserialize JSON](#)
- [Instantiate JsonSerializerOptions instances](#)
- [Enable case-insensitive matching](#)
- [Customize property names and values](#)
- [Ignore properties](#)
- [Allow invalid JSON](#)
- [Handle overflow JSON or use JsonElement or JsonNode](#)
- [Preserve references and handle circular references](#)
- [Deserialize to immutable types and non-public accessors](#)
- [Polymorphic serialization](#)
- [Migrate from Newtonsoft.Json to System.Text.Json](#)
- [Customize character encoding](#)
- [Use DOM, Utf8JsonReader, and Utf8JsonWriter](#)
- [DateTime and DateTimeOffset support](#)
- [How to use source generation](#)
- [Supported collection types](#)

- [System.Text.Json API reference](#)
- [System.Text.Json.Serialization API reference](#)

Binary serialization

9/20/2022 • 8 minutes to read • [Edit Online](#)

Serialization can be defined as the process of storing the state of an object to a storage medium. During this process, the public and private fields of the object and the name of the class, including the assembly containing the class, are converted to a stream of bytes, which is then written to a data stream. When the object is subsequently deserialized, an exact clone of the original object is created.

When implementing a serialization mechanism in an object-oriented environment, you have to make a number of tradeoffs between ease of use and flexibility. The process can be automated to a large extent, provided you are given sufficient control over the process. For example, situations may arise where simple binary serialization is not sufficient, or there might be a specific reason to decide which fields in a class need to be serialized. The following sections examine the robust serialization mechanism provided with .NET and highlight a number of important features that allow you to customize the process to meet your needs.

NOTE

The state of a UTF-8 or UTF-7 encoded object is not preserved if the object is serialized and deserialized using different .NET versions.

WARNING

Binary serialization can be dangerous. For more information, see [BinaryFormatter security guide](#).

Binary serialization allows modifying private members inside an object and therefore changing the state of it. Because of this, other serialization frameworks, like [System.Text.Json](#), that operate on the public API surface are recommended.

.NET Core

.NET Core supports binary serialization for a subset of types. You can see the list of supported types in the [Serializable types](#) section that follows. The listed types are guaranteed to be serializable between .NET Framework 4.5.1 and later versions and between .NET Core 2.0 and later versions. Other .NET implementations, such as Mono, aren't officially supported but should also work.

Serializable types

TYPE	NOTES
Microsoft.CSharp.RuntimeBinder.RuntimeBinderException	Starting in .NET Core 2.0.4.
Microsoft.CSharp.RuntimeBinder.RuntimeBinderInternalCompilerException	Starting in .NET Core 2.0.4.
System.AccessViolationException	Starting in .NET Core 2.0.4.
System.AggregateException	Starting in .NET Core 2.0.4.
System.AppDomainUnloadedException	Starting in .NET Core 2.0.4.

TYPE	NOTES
System.ApplicationException	Starting in .NET Core 2.0.4.
System.ArgumentException	Starting in .NET Core 2.0.4.
System.ArgumentNullException	Starting in .NET Core 2.0.4.
System.ArgumentOutOfRangeException	Starting in .NET Core 2.0.4.
System.ArithmetricException	Starting in .NET Core 2.0.4.
System.Array	
System.ArraySegment<T>	
System.ArrayTypeMismatchException	Starting in .NET Core 2.0.4.
System.Attribute	
System.BadImageFormatException	Starting in .NET Core 2.0.4.
System.Boolean	
System.Byte	
System.CannotUnloadAppDomainException	Starting in .NET Core 2.0.4.
System.Char	
System.Collections.ArrayList	
System.Collections.BitArray	
System.Collections.Comparer	
System.Collections.DictionaryEntry	
System.Collections.Generic.Comparer<T>	
System.Collections.Generic.Dictionary<TKey,TValue>	
System.Collections.Generic.EqualityComparer<T>	
System.Collections.Generic.HashSet<T>	
System.Collections.Generic.KeyNotFoundException	Starting in .NET Core 2.0.4.
System.Collections.Generic.KeyValuePair<TKey,TValue>	
System.Collections.Generic.LinkedList<T>	

TYPE	NOTES
System.Collections.Generic.List<T>	
System.Collections.Generic.Queue<T>	
System.Collections.Generic.SortedDictionary< TKey, TValue >	
System.Collections.Generic.SortedList< TKey, TValue >	
System.Collections.Generic.SortedSet<T>	
System.Collections.Generic.Stack<T>	
System.Collections.Hashtable	
System.Collections.ObjectModel.Collection<T>	
System.Collections.ObjectModel.KeyedCollection< TKey, TItem >	
System.Collections.ObjectModel.ObservableCollection<T>	
System.Collections.ObjectModel.ReadOnlyCollection<T>	
System.Collections.ObjectModel.ReadOnlyDictionary< TKey, TValue >	
System.Collections.ObjectModel.ReadOnlyObservableCollecti on<T>	
System.Collections.Queue	
System.Collections.SortedList	
System.Collections.Specialized.HybridDictionary	
System.Collections.Specialized.ListDictionary	
System.Collections.Specialized.OrderedDictionary	
System.Collections.Specialized.StringCollection	
System.Collections.Specialized.StringDictionary	
System.Collections.Stack	
System.Collections.Generic.NonRandomizedStringEqualityCompar	Starting in .NET Core 2.0.4.
System.ComponentModel.BindingList<T>	
System.ComponentModel.DataAnnotations.ValidationExcepti on	Starting in .NET Core 2.0.4.

TYPE	NOTES
System.ComponentModel.Design.CheckoutException	Starting in .NET Core 2.0.4.
System.ComponentModel.InvalidAsynchronousStateException	Starting in .NET Core 2.0.4.
System.ComponentModel.InvalidEnumArgumentException	Starting in .NET Core 2.0.4.
System.ComponentModel.LicenseException	Starting in .NET Core 2.0.4. Serialization from .NET Framework to .NET Core is not supported.
System.ComponentModel.WarningException	Starting in .NET Core 2.0.4.
System.ComponentModel.Win32Exception	Starting in .NET Core 2.0.4.
System.Configuration.ConfigurationErrorsException	Starting in .NET Core 2.0.4.
System.Configuration.ConfigurationException	Starting in .NET Core 2.0.4.
System.Configuration.Provider.ProviderException	Starting in .NET Core 2.0.4.
System.Configuration.SettingsPropertyIsReadOnlyException	Starting in .NET Core 2.0.4.
System.Configuration.SettingsPropertyNotFoundException	Starting in .NET Core 2.0.4.
System.Configuration.SettingsPropertyWrongTypeException	Starting in .NET Core 2.0.4.
System.ContextMarshalException	Starting in .NET Core 2.0.4.
System.DBNull	Starting in .NET Core 2.0.2 and later versions.
System.Data.Common.DbException	Starting in .NET Core 2.0.4.
System.Data.ConstraintException	Starting in .NET Core 2.0.4.
System.Data.DBConcurrencyException	Starting in .NET Core 2.0.4.
System.Data.DataException	Starting in .NET Core 2.0.4.
System.Data.DataSet	
System.Data.DataTable	If you set <code>RemotingFormat</code> to <code>SerializationFormat.Binary</code> , it can only be exchanged with .NET Core 2.1 and later versions.
System.Data.DeletedRowInaccessibleException	Starting in .NET Core 2.0.4.
System.Data.DuplicateNameException	Starting in .NET Core 2.0.4.
System.Data.EvaluateException	Starting in .NET Core 2.0.4.

TYPE	NOTES
System.Data.InRowChangingException	Starting in .NET Core 2.0.4.
System.Data.InvalidConstraintException	Starting in .NET Core 2.0.4.
System.Data.InvalidExpressionException	Starting in .NET Core 2.0.4.
System.Data.MissingPrimaryKeyException	Starting in .NET Core 2.0.4.
System.Data.NoNullAllowedException	Starting in .NET Core 2.0.4.
System.Data.Odbc.OdbcException	Starting in .NET Core 2.0.4.
System.Data.OperationAbortedException	Starting in .NET Core 2.0.4.
System.Data.PropertyCollection	
System.Data.ReadOnlyException	Starting in .NET Core 2.0.4.
System.Data.RowNotInTableException	Starting in .NET Core 2.0.4.
System.Data.SqlClient.SqlException	Starting in .NET Core 2.0.4. Serialization from .NET Framework to .NET Core is not supported
System.Data.SqlTypes.SqlAlreadyFilledException	Starting in .NET Core 2.0.4.
System.Data.SqlTypes.SqlBoolean	
System.Data.SqlTypes.SqlByte	
System.Data.SqlTypes.SqlDateTime	
System.Data.SqlTypes.SqlDouble	
System.Data.SqlTypes.SqlGuid	
System.Data.SqlTypes.SqlInt16	
System.Data.SqlTypes.SqlInt32	
System.Data.SqlTypes.SqlInt64	
System.Data.SqlTypes.SqlNotFilledException	Starting in .NET Core 2.0.4.
System.Data.SqlTypes.SqlNullValueException	Starting in .NET Core 2.0.4.
System.Data.SqlTypes.SqlString	
System.Data.SqlTypes.SqlTruncateException	Starting in .NET Core 2.0.4.

TYPE	NOTES
System.Data.SqlTypes.SqlTypeException	Starting in .NET Core 2.0.4.
System.Data.StrongTypingException	Starting in .NET Core 2.0.4.
System.Data.SyntaxException	Starting in .NET Core 2.0.4.
System.Data.VersionNotFoundException	Starting in .NET Core 2.0.4.
System.DataMisalignedException	Starting in .NET Core 2.0.4.
System.DateTime	
System.DateTimeOffset	
System.Decimal	
System.Diagnostics.Contracts.ContractException	Starting in .NET Core 2.0.4.
System.Diagnostics.Tracing.EventSourceException	Starting in .NET Core 2.0.4.
System.IO.DirectoryNotFoundException	Starting in .NET Core 2.0.4.
System.DirectoryServices.AccountManagement.MultipleMatchesException	Starting in .NET Core 2.0.4.
System.DirectoryServices.AccountManagement.NoMatchingPrincipalException	Starting in .NET Core 2.0.4.
System.DirectoryServices.AccountManagement.PasswordException	Starting in .NET Core 2.0.4.
System.DirectoryServices.AccountManagement.PrincipalException	Starting in .NET Core 2.0.4.
System.DirectoryServices.AccountManagement.PrincipalExistsException	Starting in .NET Core 2.0.4.
System.DirectoryServices.AccountManagement.PrincipalOperationException	Starting in .NET Core 2.0.4.
System.DirectoryServices.AccountManagement.PrincipalServerDownException	Starting in .NET Core 2.0.4.
System.DirectoryServices.ActiveDirectory.ActiveDirectoryObjectExistsException	Starting in .NET Core 2.0.4.
System.DirectoryServices.ActiveDirectory.ActiveDirectoryObjectNotFoundException	Starting in .NET Core 2.0.4.
System.DirectoryServices.ActiveDirectory.ActiveDirectoryOperationException	Starting in .NET Core 2.0.4.

TYPE	NOTES
System.DirectoryServices.ActiveDirectory.ActiveDirectoryServerDownException	Starting in .NET Core 2.0.4.
System.DirectoryServices.ActiveDirectory.ForestTrustCollisionException	Starting in .NET Core 2.0.4.
System.DirectoryServices.ActiveDirectory.SyncFromAllServersOperationException	Starting in .NET Core 2.0.4.
System.DirectoryServices.DirectoryServicesCOMException	Starting in .NET Core 2.0.4.
System.DirectoryServices.Protocols.BerConversionException	Starting in .NET Core 2.0.4.
System.DirectoryServices.Protocols.DirectoryException	Starting in .NET Core 2.0.4.
System.DirectoryServices.Protocols.DirectoryOperationException	Starting in .NET Core 2.0.4.
System.DirectoryServices.Protocols.LdapException	Starting in .NET Core 2.0.4.
System.DirectoryServices.Protocols.TlsOperationException	Starting in .NET Core 2.0.4.
System.DivideByZeroException	Starting in .NET Core 2.0.4.
System.DllNotFoundException	Starting in .NET Core 2.0.4.
System.Double	
System.Drawing.Bitmap	Starting in .NET Core 3.0
System.Drawing.Color	
System.Drawing.Icon	Starting in .NET Core 3.0
System.Drawing.Image	Starting in .NET Core 3.0
System.Drawing.Imaging.Metafile	Starting in .NET Core 3.0
System.Drawing.Point	
System.Drawing.PointF	
System.Drawing.Rectangle	
System.Drawing.RectangleF	
System.Drawing.Size	
System.Drawing.SizeF	
System.DuplicateWaitObjectException	Starting in .NET Core 2.0.4.

TYPE	NOTES
System.EntryPointNotFoundException	Starting in .NET Core 2.0.4.
System.Enum	
System.EventArgs	Starting in .NET Core 2.0.6.
System.Exception	
System.ExecutionEngineException	Starting in .NET Core 2.0.4.
System.FieldAccessException	Starting in .NET Core 2.0.4.
System.FormatException	Starting in .NET Core 2.0.4.
System.Globalization.CompareInfo	
System.Globalization.CultureNotFoundException	Starting in .NET Core 2.0.4.
System.Globalization.SortVersion	
System.Guid	
System.IO.Compression.ZLibException	Starting in .NET Core 2.0.4.
System.IO.DriveNotFoundException	Starting in .NET Core 2.0.4.
System.IO.EndOfStreamException	Starting in .NET Core 2.0.4.
System.IO.FileFormatException	Starting in .NET Core 2.0.4.
System.IO.FileLoadException	Starting in .NET Core 2.0.4.
System.IO.FileNotFoundException	Starting in .NET Core 2.0.4.
System.IO.IOException	Starting in .NET Core 2.0.4.
System.IO.InternalBufferOverflowException	Starting in .NET Core 2.0.4.
System.IO.InvalidDataException	Starting in .NET Core 2.0.4.
System.IO.IsolatedStorage.IsolatedStorageException	Starting in .NET Core 2.0.4.
System.IO.PathTooLongException	Starting in .NET Core 2.0.4.
System.IndexOutOfRangeException	Starting in .NET Core 2.0.4.
System.InsufficientExecutionStackException	Starting in .NET Core 2.0.4.
System.InsufficientMemoryException	Starting in .NET Core 2.0.4.

TYPE	NOTES
System.Int16	
System.Int32	
System.Int64	
System.IntPtr	
System.InvalidCastException	Starting in .NET Core 2.0.4.
System.InvalidOperationException	Starting in .NET Core 2.0.4.
System.InvalidProgramException	Starting in .NET Core 2.0.4.
System.InvalidTimeZoneException	Starting in .NET Core 2.0.4.
System.MemberAccessException	Starting in .NET Core 2.0.4.
System.MethodAccessException	Starting in .NET Core 2.0.4.
System.MissingFieldException	Starting in .NET Core 2.0.4.
System.MissingMemberException	Starting in .NET Core 2.0.4.
System.MissingMethodException	Starting in .NET Core 2.0.4.
System.MulticastNotSupportedException	Starting in .NET Core 2.0.4.
System.Net.Cookie	
System.Net.CookieCollection	
System.Net.CookieContainer	
System.Net.CookieException	Starting in .NET Core 2.0.4.
System.Net.HttpListenerException	Starting in .NET Core 2.0.4.
System.Net.Mail.SmtpException	Starting in .NET Core 2.0.4.
System.Net.Mail.SmtpFailedRecipientException	Starting in .NET Core 2.0.4.
System.Net.Mail.SmtpFailedRecipientsException	Starting in .NET Core 2.0.4.
System.Net.NetworkInformation.NetworkInformationException	Starting in .NET Core 2.0.4.
System.Net.NetworkInformation.PingException	Starting in .NET Core 2.0.4.
System.Net.ProtocolViolationException	Starting in .NET Core 2.0.4.

TYPE	NOTES
System.Net.Sockets.SocketException	Starting in .NET Core 2.0.4.
System.Net.WebException	Starting in .NET Core 2.0.4.
System.Net.WebSockets.WebSocketException	Starting in .NET Core 2.0.4.
System.NotFiniteNumberException	Starting in .NET Core 2.0.4.
System.NotImplementedException	Starting in .NET Core 2.0.4.
System.NotSupportedException	Starting in .NET Core 2.0.4.
System.NullReferenceException	Starting in .NET Core 2.0.4.
System.Nullable<T>	
System.Numerics.BigInteger	
System.Numerics.Complex	
System.Object	
System.ObjectDisposedException	Starting in .NET Core 2.0.4.
System.OperationCanceledException	Starting in .NET Core 2.0.4.
System.OutOfMemoryException	Starting in .NET Core 2.0.4.
System.OverflowException	Starting in .NET Core 2.0.4.
System.PlatformNotSupportedException	Starting in .NET Core 2.0.4.
System.RankException	Starting in .NET Core 2.0.4.
System.Reflection.AmbiguousMatchException	Starting in .NET Core 2.0.4.
System.Reflection.CustomAttributeFormatException	Starting in .NET Core 2.0.4.
System.Reflection.InvalidFilterCriteriaException	Starting in .NET Core 2.0.4.
System.Reflection.ReflectionTypeLoadException	Starting in .NET Core 2.0.4. Serialization from .NET Framework to .NET Core is not supported.
System.Reflection.TargetException	Starting in .NET Core 2.0.4.
System.Reflection.TargetInvocationException	Starting in .NET Core 2.0.4.
System.Reflection.TargetParameterCountException	Starting in .NET Core 2.0.4.

TYPE	NOTES
System.Resources.MissingManifestResourceException	Starting in .NET Core 2.0.4.
System.Resources.MissingSatelliteAssemblyException	Starting in .NET Core 2.0.4.
System.Runtime.CompilerServices.RuntimeWrappedException	Starting in .NET Core 2.0.4.
System.Runtime.InteropServices.COMException	Starting in .NET Core 2.0.4.
System.Runtime.InteropServices.ExternalException	Starting in .NET Core 2.0.4.
System.Runtime.InteropServices.InvalidComObjectException	Starting in .NET Core 2.0.4.
System.Runtime.InteropServices.InvalidOleVariantTypeException	Starting in .NET Core 2.0.4.
System.Runtime.InteropServices.MarshalDirectiveException	Starting in .NET Core 2.0.4.
System.Runtime.InteropServices.SEHException	Starting in .NET Core 2.0.4.
System.Runtime.InteropServices.SafeArrayRankMismatchException	Starting in .NET Core 2.0.4.
System.Runtime.InteropServices.SafeArrayTypeMismatchException	Starting in .NET Core 2.0.4.
System.Runtime.Serialization.InvalidDataContractException	Starting in .NET Core 2.0.4.
System.Runtime.Serialization.SerializationException	Starting in .NET Core 2.0.4.
System.SByte	
System.Security.AccessControl.PrivilegeNotHeldException	Starting in .NET Core 2.0.4.
System.Security.Authentication.AuthenticationException	Starting in .NET Core 2.0.4.
System.Security.Authentication.InvalidCredentialException	Starting in .NET Core 2.0.4.
System.Security.Cryptography.CryptographicException	Starting in .NET Core 2.0.4.
System.Security.Cryptography.CryptographicUnexpectedOperationException	Starting in .NET Core 2.0.4.
System.Security.Cryptography.Xml.CryptoSignedXmlRecursionException	Starting in .NET Core 2.0.4.
System.Security.HostProtectionException	Starting in .NET Core 2.0.4.
System.Security.Policy.PolicyException	Starting in .NET Core 2.0.4.
System.Security.Principal.IdentityNotMappedException	Starting in .NET Core 2.0.4.

TYPE	NOTES
System.Security.SecurityException	Starting in .NET Core 2.0.4. Limited serialization data.
System.Security.VerificationException	Starting in .NET Core 2.0.4.
System.Security.XmlSyntaxException	Starting in .NET Core 2.0.4.
System.ServiceProcess.TimeoutException	Starting in .NET Core 2.0.4.
System.Single	
System.StackOverflowException	Starting in .NET Core 2.0.4.
System.String	
System.StringComparer	
System.SystemException	Starting in .NET Core 2.0.4.
System.Text.DecoderFallbackException	Starting in .NET Core 2.0.4.
System.Text.EncoderFallbackException	Starting in .NET Core 2.0.4.
System.Text.RegularExpressions.RegexMatchTimeoutException	Starting in .NET Core 2.0.4.
System.Text.StringBuilder	
System.Threading.AbandonedMutexException	Starting in .NET Core 2.0.4.
System.Threading.BarrierPostPhaseException	Starting in .NET Core 2.0.4.
System.Threading.LockRecursionException	Starting in .NET Core 2.0.4.
System.ThreadingSemaphoreFullException	Starting in .NET Core 2.0.4.
System.Threading.SynchronizationLockException	Starting in .NET Core 2.0.4.
System.Threading.Tasks.TaskCanceledException	Starting in .NET Core 2.0.4.
System.Threading.Tasks.TaskSchedulerException	Starting in .NET Core 2.0.4.
System.Threading.ThreadAbortException	Starting in .NET Core 2.0.4.
System.Threading.ThreadInterruptedException	Starting in .NET Core 2.0.4.
System.Threading.ThreadStartException	Starting in .NET Core 2.0.4.
System.Threading.ThreadStateException	Starting in .NET Core 2.0.4.

TYPE	NOTES
System.Threading.WaitHandleCannotBeOpenedException	Starting in .NET Core 2.0.4.
System.TimeSpan	
System.TimeZoneInfo.AdjustmentRule	
System.TimeZoneInfo	
System.TimeZoneNotFoundException	Starting in .NET Core 2.0.4.
System.TimeoutException	Starting in .NET Core 2.0.4.
System.Transactions.TransactionAbortedException	Starting in .NET Core 2.0.4.
System.Transactions.TransactionException	Starting in .NET Core 2.0.4.
System.Transactions.TransactionInDoubtException	Starting in .NET Core 2.0.4.
System.Transactions.TransactionManagerCommunicationException	Starting in .NET Core 2.0.4.
System.Transactions.TransactionPromotionException	Starting in .NET Core 2.0.4.
System.Tuple	
System.TypeAccessException	Starting in .NET Core 2.0.4.
System.TypeInitializationException	Starting in .NET Core 2.0.4.
System.TypeLoadException	Starting in .NET Core 2.0.4.
System.TypeUnloadedException	Starting in .NET Core 2.0.4.
System.UInt16	
System.UInt32	
System.UInt64	
System.UIntPtr	
System.UnauthorizedAccessException	Starting in .NET Core 2.0.4.
System.Uri	
System.UriFormatException	Starting in .NET Core 2.0.4.
System.ValueTuple	Not serializable in .NET Framework 4.7 and earlier versions.
System.ValueType	

TYPE	NOTES
System.Version	
System.WeakReference<T>	
System.WeakReference	
System.Xml.Schema.XmlSchemaException	Starting in .NET Core 2.0.4.
System.Xml.Schema.XmlSchemaInferenceException	Starting in .NET Core 2.0.4.
System.Xml.Schema.XmlSchemaValidationException	Starting in .NET Core 2.0.4.
System.Xml.XPath.XPathException	Starting in .NET Core 2.0.4.
System.Xml.XmlException	Starting in .NET Core 2.0.4.
System.Xml.Xsl.XsltCompileException	Starting in .NET Core 2.0.4.
System.Xml.Xsl.XsltException	Starting in .NET Core 2.0.4.

See also

- [System.Runtime.Serialization](#)
Contains classes that can be used for serializing and deserializing objects.
- [XML and SOAP Serialization](#)
Describes the XML serialization mechanism that is included with the common language runtime.
- [Security and Serialization](#)
Describes the secure coding guidelines to follow when writing code that performs serialization.
- [.NET Remoting](#)
Describes the various methods in .NET Framework for remote communications.
- [XML Web Services Created Using ASP.NET and XML Web Service Clients](#)
Articles that describe and explain how to program XML Web services created using ASP.NET.

Deserialization risks in use of BinaryFormatter and related types

9/20/2022 • 5 minutes to read • [Edit Online](#)

This article applies to the following types:

- [BinaryFormatter](#)
- [SoapFormatter](#)
- [NetDataContractSerializer](#)
- [LosFormatter](#)
- [ObjectStateFormatter](#)

This article applies to the following .NET implementations:

- .NET Framework all versions
- .NET Core 2.1 - 3.1
- .NET 5 and later

WARNING

The `BinaryFormatter` type is dangerous and is *not* recommended for data processing. Applications should stop using

`BinaryFormatter` as soon as possible, even if they believe the data they're processing to be trustworthy.

`BinaryFormatter` is insecure and can't be made secure.

Deserialization vulnerabilities

Deserialization vulnerabilities are a threat category where request payloads are processed insecurely. An attacker who successfully leverages these vulnerabilities against an app can cause denial of service (DoS), information disclosure, or remote code execution inside the target app. This risk category consistently makes the [OWASP Top 10](#). Targets include apps written in a variety of languages, including C/C++, Java, and C#.

In .NET, the biggest risk target is apps that use the `BinaryFormatter` type to deserialize data. `BinaryFormatter` is widely used throughout the .NET ecosystem because of its power and its ease of use. However, this same power gives attackers the ability to influence control flow within the target app. Successful attacks can result in the attacker being able to run code within the context of the target process.

As a simpler analogy, assume that calling `BinaryFormatter.Deserialize` over a payload is the equivalent of interpreting that payload as a standalone executable and launching it.

BinaryFormatter security vulnerabilities

WARNING

The `BinaryFormatter.Deserialize` method is **never** safe when used with untrusted input. We strongly recommend that consumers instead consider using one of the alternatives outlined later in this article.

`BinaryFormatter` was implemented before deserialization vulnerabilities were a well-understood threat category. As a result, the code does not follow modern best practices. The `Deserialize` method can be used as a

vector for attackers to perform DoS attacks against consuming apps. These attacks might render the app unresponsive or result in unexpected process termination. This category of attack cannot be mitigated with a `SerializationBinder` or any other `BinaryFormatter` configuration switch. .NET considers this behavior to be *by design* and won't issue a code update to modify the behavior.

`BinaryFormatter.Deserialize` may be vulnerable to other attack categories, such as information disclosure or remote code execution. Utilizing features such as a custom `SerializationBinder` may be insufficient to properly mitigate these risks. The possibility exists that a novel vulnerability will be discovered for which .NET cannot practically publish a security update. Consumers should assess their individual scenarios and consider their potential exposure to these risks.

We recommend that `BinaryFormatter` consumers perform individual risk assessments on their apps. It is the consumer's sole responsibility to determine whether to utilize `BinaryFormatter`. Consumers should risk assess the security, technical, reputation, legal, and regulatory requirements of using `BinaryFormatter`.

Preferred alternatives

.NET offers several in-box serializers that can handle untrusted data safely:

- `XmlSerializer` and `DataContractSerializer` to serialize object graphs into and from XML. Do not confuse `DataContractSerializer` with `NetDataContractSerializer`.
- `BinaryReader` and `BinaryWriter` for XML and JSON.
- The `System.Text.Json` APIs to serialize object graphs into JSON.

Dangerous alternatives

Avoid the following serializers:

- `SoapFormatter`
- `LosFormatter`
- `NetDataContractSerializer`
- `ObjectStateFormatter`

The preceding serializers all perform unrestricted polymorphic deserialization and are dangerous, just like `BinaryFormatter`.

The risks of assuming data to be trustworthy

Frequently, an app developer might believe that they are processing only trusted input. The safe input case is true in some rare circumstances. But it's much more common that a payload crosses a trust boundary without the developer realizing it.

Consider an on-prem server where employees use a desktop client from their workstations to interact with the service. This scenario might be seen naïvely as a "safe" setup where utilizing `BinaryFormatter` is acceptable. However, this scenario presents a vector for malware that gains access to a single employee's machine to be able to spread throughout the enterprise. That malware can leverage the enterprise's use of `BinaryFormatter` to move laterally from the employee's workstation to the backend server. It can then exfiltrate the company's sensitive data. Such data could include trade secrets or customer data.

Consider also an app that uses `BinaryFormatter` to persist save state. This might at first seem to be a safe scenario, as reading and writing data on your own hard drive represents a minor threat. However, sharing documents across email or the internet is common, and most end users wouldn't perceive opening these downloaded files as risky behavior.

This scenario can be leveraged to nefarious effect. If the app is a game, users who share save files unknowingly

place themselves at risk. The developers themselves can also be targeted. The attacker might email the developers' tech support, attaching a malicious data file and asking the support staff to open it. This kind of attack could give the attacker a foothold in the enterprise.

Another scenario is where the data file is stored in cloud storage and automatically synced between the user's machines. An attacker who is able to gain access to the cloud storage account can poison the data file. This data file will be automatically synced to the user's machines. The next time the user opens the data file, the attacker's payload runs. Thus the attacker can leverage a cloud storage account compromise to gain full code execution permissions.

Consider an app that moves from a desktop-install model to a cloud-first model. This scenario includes apps that move from a desktop app or rich client model into a web-based model. Any threat models drawn for the desktop app aren't necessarily applicable to the cloud-based service. The threat model for the desktop app might dismiss a given threat as "not interesting for the client to attack itself." But that same threat might become interesting when it considers a remote user (the client) attacking the cloud service itself.

NOTE

In general terms, the intent of serialization is to transmit an object into or out of an app. A threat modeling exercise almost always marks this kind of data transfer as crossing a trust boundary.

See also

- [YSoSerial.Net](#) for research into how adversaries attack apps that utilize `BinaryFormatter`.
- General background on deserialization vulnerabilities:
 - [OWASP Top 10 - A8:2017-Insecure Deserialization](#)
 - [CWE-502: Deserialization of Untrusted Data](#)

BinaryFormatter event source

9/20/2022 • 5 minutes to read • [Edit Online](#)

Starting with .NET 5, `BinaryFormatter` includes a built-in `EventSource` that gives you visibility into when an object serialization or deserialization is occurring. Apps can use `EventListener`-derived types to listen for these notifications and log them.

This functionality is not a substitute for a `SerializationBinder` or an `ISerializationSurrogate` and can't be used to modify the data being serialized or deserialized. Rather, this eventing system is intended to provide insight into the types being serialized or deserialized. It can also be used to detect unintended calls into the `BinaryFormatter` infrastructure, such as calls originating from third-party library code.

Description of events

The `BinaryFormatter` event source has the well-known name

`System.Runtime.Serialization.Formatters.Binary.BinaryFormatterEventSource`. Listeners may subscribe to 6 events.

SerializationStart event (id = 10)

Raised when `BinaryFormatter.Serialize` has been called and has started the serialization process. This event is paired with the `SerializationEnd` event. The `serializationStart` event can be called recursively if an object calls `BinaryFormatter.Serialize` within its own serialization routine.

This event doesn't contain a payload.

SerializationEnd event (id = 11)

Raised when `BinaryFormatter.Serialize` has completed its work. Each occurrence of `SerializationEnd` denotes the completion of the last unpaired `SerializationStart` event.

This event doesn't contain a payload.

SerializingObject event (id = 12)

Raised when `BinaryFormatter.Serialize` is in the process of serializing a non-primitive type. The `BinaryFormatter` infrastructure special-cases certain types (such as `string` and `int`) and doesn't raise this event when these types are encountered. This event is raised for user-defined types and other types that `BinaryFormatter` doesn't natively understand.

This event may be raised zero or more times between `SerializationStart` and `SerializationEnd` events.

This event contains a payload with one argument:

- `typeName` (`string`): The assembly-qualified name (see `Type.AssemblyQualifiedName`) of the type being serialized.

DeserializationStart event (id = 20)

Raised when `BinaryFormatter.Deserialize` has been called and has started the deserialization process. This event is paired with the `DeserializationEnd` event. The `DeserializationStart` event can be called recursively if an object calls `BinaryFormatter.Deserialize` within its own deserialization routine.

This event doesn't contain a payload.

DeserializationEnd event (id = 21)

Raised when `BinaryFormatter.Deserialize` has completed its work. Each occurrence of `DeserializationEnd` denotes the completion of the last unpaired `DeserializationStart` event.

This event doesn't contain a payload.

DeserializingObject event (id = 22)

Raised when `BinaryFormatter.Deserialize` is in the process of deserializing a non-primitive type. The `BinaryFormatter` infrastructure special-cases certain types (such as `string` and `int`) and doesn't raise this event when these types are encountered. This event is raised for user-defined types and other types that `BinaryFormatter` doesn't natively understand.

This event may be raised zero or more times between `DeserializationStart` and `DeserializationEnd` events.

This event contains a payload with one argument.

- `typeName` (`string`): The assembly-qualified name (see [Type.AssemblyQualifiedName](#)) of the type being serialized.

[Advanced] Subscribing to a subset of notifications

Listeners who wish to subscribe to only a subset of notifications can choose which keywords to enable.

- `Serialization = (EventKeywords)1`: Raises the `SerializationStart`, `SerializationEnd`, and `SerializingObject` events.
- `Deserialization = (EventKeywords)2`: Raises the `DeserializationStart`, `DeserializationEnd`, and `DeserializingObject` events.

If no keyword filters are provided during `EventListener` registration, all events are raised.

For more information, see [System.Diagnostics.Tracing.EventKeywords](#).

Sample code

The following code:

- creates an `EventListener`-derived type that writes to `System.Console`,
- subscribes that listener to `BinaryFormatter`-produced notifications,
- serializes and deserializes a simple object graph using `BinaryFormatter`, and
- analyzes the events that have been raised.

```
using System;
using System.Diagnostics.Tracing;
using System.IO;
using System.Runtime.Serialization.Formatters.Binary;

namespace BinaryFormatterEventSample
{
    class Program
    {
        static EventListener _globalListener = null;

        static void Main(string[] args)
        {
            // First, set up the event listener.
            // Note: We assign it to a static field so that it doesn't get GCed.
            // We also provide a callback that subscribes this listener to all
            // events produced by the well-known BinaryFormatter source.

            _globalListener = new ConsoleEventListener();
            _globalListener.EventSourceCreated += (sender, args) =>
            {

```

```

        if (args.EventSource?.Name ==
            "System.Runtime.Serialization.Formatters.Binary.BinaryFormatterEventSource")
    {
        ((EventListener)sender)
            .EnableEvents(args.EventSource, EventLevel.LogAlways);
    }
};

// Next, create the Person object and serialize it.

Person originalPerson = new Person()
{
    FirstName = "Logan",
    LastName = "Edwards",
    FavoriteBook = new Book()
    {
        Title = "A Tale of Two Cities",
        Author = "Charles Dickens",
        Price = 10.25M
    }
};

byte[] serializedPerson = SerializePerson(originalPerson);

// Finally, deserialize the Person object.

Person rehydratedPerson = DeserializePerson(serializedPerson);

Console.WriteLine
    ($"Rehydrated person {rehydratedPerson.FirstName} {rehydratedPerson.LastName}");
Console.Write
    ($"Favorite book: {rehydratedPerson.FavoriteBook.Title} ");
Console.Write
    ($"by {rehydratedPerson.FavoriteBook.Author}, ");
Console.WriteLine
    ($"list price {rehydratedPerson.FavoriteBook.Price}");
}

private static byte[] SerializePerson(Person p)
{
    MemoryStream memStream = new MemoryStream();
    BinaryFormatter formatter = new BinaryFormatter();
#pragma warning disable SYSLIB0011 // BinaryFormatter.Serialize is obsolete
    formatter.Serialize(memStream, p);
#pragma warning restore SYSLIB0011

    return memStream.ToArray();
}

private static Person DeserializePerson(byte[] serializedData)
{
    MemoryStream memStream = new MemoryStream(serializedData);
    BinaryFormatter formatter = new BinaryFormatter();

#pragma warning disable SYSLIB0011 // Danger: BinaryFormatter.Deserialize is insecure for untrusted input
    return (Person)formatter.Deserialize(memStream);
#pragma warning restore SYSLIB0011
}

[Serializable]
public class Person
{
    public string FirstName;
    public string LastName;
    public Book FavoriteBook;
}

[Serializable]

```

```

[Serializable]
public class Book
{
    public string Title;
    public string Author;
    public decimal Price;
}

// A sample EventListener that writes data to System.Console.
public class ConsoleEventListener : EventListener
{
    protected override void OnEventWritten(EventWrittenEventArgs eventData)
    {
        base.OnEventWritten(eventData);

        Console.WriteLine($"Event {eventData.EventName} (id={eventData.EventId}) received.");
        if (eventData.PayloadNames != null)
        {
            for (int i = 0; i < eventData.PayloadNames.Count; i++)
            {
                Console.WriteLine($"{eventData.PayloadNames[i]} = {eventData.Payload[i]}");
            }
        }
    }
}

```

The preceding code produces output similar to the following example:

```

Event SerializationStart (id=10) received.
Event SerializingObject (id=12) received.
typeName = BinaryFormatterEventSample.Person, BinaryFormatterEventSample, Version=1.0.0.0, Culture=neutral,
PublicKeyToken=null
Event SerializingObject (id=12) received.
typeName = BinaryFormatterEventSample.Book, BinaryFormatterEventSample, Version=1.0.0.0, Culture=neutral,
PublicKeyToken=null
Event SerializationStop (id=11) received.
Event DeserializationStart (id=20) received.
Event DeserializingObject (id=22) received.
typeName = BinaryFormatterEventSample.Person, BinaryFormatterEventSample, Version=1.0.0.0, Culture=neutral,
PublicKeyToken=null
Event DeserializingObject (id=22) received.
typeName = BinaryFormatterEventSample.Book, BinaryFormatterEventSample, Version=1.0.0.0, Culture=neutral,
PublicKeyToken=null
Event DeserializationStop (id=21) received.
Rehydrated person Logan Edwards
Favorite book: A Tale of Two Cities by Charles Dickens, list price 10.25

```

In this sample, the console-based `EventListener` logs that serialization starts, instances of `Person` and `Book` are serialized, and then serialization completes. Similarly, once deserialization has started, instances of `Person` and `Book` are deserialized, and then deserialization completes.

The app then prints the values contained in the serialized `Person` to demonstrate that the object did in fact serialize and deserialize properly.

See also

For more information on using `EventListener` to receive `EventSource`-based notifications, see [the `EventListener` class](#).

Serialization concepts

9/20/2022 • 2 minutes to read • [Edit Online](#)

Why would you want to use serialization? The two most important reasons are to persist the state of an object to a storage medium so an exact copy can be re-created at a later stage, and to send the object by value from one application domain to another. For example, serialization is used to save session state in ASP.NET and to copy objects to the Clipboard in Windows Forms. It is also used by remoting to pass objects by value from one application domain to another.

WARNING

Binary serialization can be dangerous. For more information, see [BinaryFormatter security guide](#).

Persistent storage

It is often necessary to store the value of the fields of an object to disk and then, later, retrieve this data. Although this is easy to achieve without relying on serialization, this approach is often cumbersome and error prone, and becomes progressively more complex when you need to track a hierarchy of objects. Imagine writing a large business application, that contains thousands of objects, and having to write code to save and restore the fields and properties to and from disk for each object. Serialization provides a convenient mechanism for achieving this objective.

The common language runtime manages how objects are stored in memory and provides an automated serialization mechanism by using [reflection](#). When an object is serialized, the name of the class, the assembly, and all the data members of the class instance are written to storage. Objects often store references to other instances in member variables. When the class is serialized, the serialization engine tracks referenced objects, already serialized, to ensure that the same object is not serialized more than once. The serialization architecture provided by .NET correctly handles object graphs and circular references automatically. The only requirement placed on object graphs is that all objects, referenced by the serialized object, must also be marked as `Serializable` (for more information, see [Basic Serialization](#)). If this is not done, an exception will be thrown when the serializer attempts to serialize the unmarked object.

When the serialized class is deserialized, the class is recreated and the values of all the data members are automatically restored.

Marshal by value

Objects are valid only in the application domain where they are created. Any attempt to pass the object as a parameter or return it as a result will fail unless the object derives from `MarshalByRefObject` or is marked as `Serializable`. If the object is marked as `Serializable`, the object will automatically be serialized, transported from the one application domain to the other, and then deserialized to produce an exact copy of the object in the second application domain. This process is typically referred to as marshal-by-value.

When an object derives from `MarshalByRefObject`, an object reference is passed from one application domain to another, rather than the object itself. You can also mark an object that derives from `MarshalByRefObject` as `Serializable`. When this object is used with remoting, the formatter responsible for serialization, which has been preconfigured with a surrogate selector (`SurrogateSelector`), takes control of the serialization process, and replaces all objects derived from `MarshalByRefObject` with a proxy. Without the `SurrogateSelector` in place, the serialization architecture follows the standard serialization rules described in [Steps in the Serialization Process](#).

Related sections

[Binary Serialization](#)

Describes the binary serialization mechanism that is included with the common language runtime.

[XML and SOAP Serialization](#)

Describes the XML and SOAP serialization mechanism that is included with the common language runtime.

Basic serialization

9/20/2022 • 2 minutes to read • [Edit Online](#)

WARNING

Binary serialization can be dangerous. For more information, see [BinaryFormatter security guide](#).

The easiest way to make a class serializable is to mark it with the [SerializableAttribute](#) as follows.

```
[Serializable]
public class MyObject {
    public int n1 = 0;
    public int n2 = 0;
    public String str = null;
}
```

The following code example shows how an instance of this class can be serialized to a file.

```
MyObject obj = new MyObject();
obj.n1 = 1;
obj.n2 = 24;
obj.str = "Some String";
IFormatter formatter = new BinaryFormatter();
Stream stream = new FileStream("MyFile.bin", FileMode.Create, FileAccess.Write, FileShare.None);
formatter.Serialize(stream, obj);
stream.Close();
```

This example uses a binary formatter to do the serialization. All you need to do is create an instance of the stream and the formatter you intend to use, and then call the **Serialize** method on the formatter. The stream and the object to serialize are provided as parameters to this call. Although it is not explicitly demonstrated in this example, all member variables of a class will be serialized—even variables marked as private. In this aspect, binary serialization differs from the [XmlSerializer](#) class, which only serializes public fields. For information on excluding member variables from binary serialization, see [Selective Serialization](#).

Restoring the object back to its former state is just as easy. First, create a stream for reading and a [Formatter](#), and then instruct the formatter to deserialize the object. The code example below shows how this is done.

```
IFormatter formatter = new BinaryFormatter();
Stream stream = new FileStream("MyFile.bin", FileMode.Open, FileAccess.Read, FileShare.Read);
MyObject obj = (MyObject) formatter.Deserialize(stream);
stream.Close();

// Here's the proof.
Console.WriteLine("n1: {0}", obj.n1);
Console.WriteLine("n2: {0}", obj.n2);
Console.WriteLine("str: {0}", obj.str);
```

The [BinaryFormatter](#) used above is very efficient and produces a compact byte stream. All objects serialized with this formatter can also be deserialized with it, which makes it an ideal tool for serializing objects that will be deserialized on .NET. It is important to note that constructors are not called when an object is deserialized. This constraint is placed on deserialization for performance reasons. However, this violates some of the usual contracts the runtime makes with the object writer, and developers should ensure that they understand the

ramifications when marking an object as serializable.

If portability is a requirement, use the [SoapFormatter](#) instead. Simply replace the **BinaryFormatter** in the code above with **SapFormatter**, and call **Serialize** and **Deserialize** as before. This formatter produces the following output for the example used above.

```
<SOAP-ENV:Envelope
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:a1="http://schemas.microsoft.com/clr/assem/ToFile">

  <SOAP-ENV:Body>
    <a1:MyObject id="ref-1">
      <n1>1</n1>
      <n2>24</n2>
      <str id="ref-3">Some String</str>
    </a1:MyObject>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

It's important to note that the [Serializable](#) attribute cannot be inherited. If you derive a new class from `MyObject`, the new class must be marked with the attribute as well, or it cannot be serialized. For example, when you attempt to serialize an instance of the class below, you'll get a [SerializationException](#) informing you that the `MyStuff` type is not marked as serializable.

```
public class MyStuff : MyObject
{
  public int n3;
}
```

Using the [Serializable](#) attribute is convenient, but it has limitations as previously demonstrated. Refer to the [Serialization Guidelines](#) for information about when you should mark a class for serialization. Serialization cannot be added to a class after it has been compiled.

See also

- [Binary Serialization](#)
- [XML and SOAP Serialization](#)

Selective serialization

9/20/2022 • 2 minutes to read • [Edit Online](#)

A class often contains fields that shouldn't be serialized. For example, assume a class stores a thread ID in a member variable. When the class is deserialized, the thread stored the ID for when the class was serialized might no longer be running; so serializing this value doesn't make sense. You can prevent member variables from being serialized by marking them with the [NonSerialized](#) attribute as follows.

```
[Serializable]
public class MyObject
{
    public int n1;
    [NonSerialized] public int n2;
    public String str;
}
```

If possible, make an object that could contain security-sensitive data nonserializable. If the object must be serialized, apply the [NonSerialized](#) attribute to specific fields that store sensitive data. If you don't exclude these fields from serialization, be aware that the data they store are exposed to any code that has permission to serialize. For more information about writing secure serialization code, see [Security and Serialization](#).

WARNING

Binary serialization can be dangerous. For more information, see [BinaryFormatter security guide](#).

See also

- [Binary Serialization](#)
- [XML and SOAP Serialization](#)
- [Security and Serialization](#)

Custom serialization

9/20/2022 • 6 minutes to read • [Edit Online](#)

Custom serialization is the process of controlling the serialization and deserialization of a type. By controlling serialization, it's possible to ensure serialization compatibility, which is the ability to serialize and deserialize between versions of a type without breaking the core functionality of the type. For example, in the first version of a type, there may be only two fields. In the next version of a type, several more fields are added. Yet the second version of an application must be able to serialize and deserialize both types. The following sections describe how to control serialization.

WARNING

Binary serialization can be dangerous. For more information, see [BinaryFormatter security guide](#).

IMPORTANT

In versions previous to .NET Framework 4.0, serialization of custom user data in a partially trusted assembly was accomplished using `GetObjectData`. In .NET Framework version 4.0 - 4.8, that method is marked with the `SecurityCriticalAttribute` attribute, which prevents execution in partially trusted assemblies. To work around this condition, implement the `ISafeSerializationData` interface.

Running custom methods during and after serialization

The recommended way to run custom methods during and after serialization is to apply the following attributes to methods that are used to correct data during and after serialization:

- [OnDeserializedAttribute](#)
- [OnDeserializingAttribute](#)
- [OnSerializedAttribute](#)
- [OnSerializingAttribute](#)

These attributes allow the type to participate in any one of, or all four of the phases, of the serialization and deserialization processes. The attributes specify the methods of the type that should be invoked during each phase. The methods do not access the serialization stream but instead allow you to alter the object before and after serialization, or before and after deserialization. The attributes can be applied at all levels of the type inheritance hierarchy, and each method is called in the hierarchy from the base to the most derived. This mechanism avoids the complexity and any resulting issues of implementing the `ISerializable` interface by giving the responsibility for serialization and deserialization to the most derived implementation. Additionally, this mechanism allows the formatters to ignore the population of fields and retrieval from the serialization stream. For details and examples of controlling serialization and deserialization, click any of the previous links.

In addition, when adding a new field to an existing serializable type, apply the `OptionalFieldAttribute` attribute to the field. The `BinaryFormatter` and the `SoapFormatter` ignores the absence of the field when a stream that is missing the new field is processed.

Implementing the `ISerializable` interface

The other way to control serialization is achieved by implementing the [ISerializable](#) interface on an object. Note, however, that the method in the previous section supersedes this method to control serialization.

In addition, you should not use default serialization on a class that is marked with the [Serializable](#) attribute and has declarative or imperative security at the class level or on its constructors. Instead, these classes should always implement the [ISerializable](#) interface.

Implementing [ISerializable](#) involves implementing the `GetObjectData` method and a special constructor that's used when the object is deserialized. The following sample code shows how to implement [ISerializable](#) on the `MyObject` class from a previous section.

```
[Serializable]
public class MyObject : ISerializable
{
    public int n1;
    public int n2;
    public String str;

    public MyObject()
    {
    }

    protected MyObject(SerializationInfo info, StreamingContext context)
    {
        n1 = info.GetInt32("i");
        n2 = info.GetInt32("j");
        str = info.GetString("k");
    }

    public virtual void GetObjectData(SerializationInfo info, StreamingContext context)
    {
        info.AddValue("i", n1);
        info.AddValue("j", n2);
        info.AddValue("k", str);
    }
}
```

```
<Serializable()> _
Public Class MyObject
    Implements ISerializable
    Public n1 As Integer
    Public n2 As Integer
    Public str As String

    Public Sub New()
    End Sub

    Protected Sub New(ByVal info As SerializationInfo, ByVal context As StreamingContext)
        n1 = info.GetInt32("i")
        n2 = info.GetInt32("j")
        str = info.GetString("k")
    End Sub 'New

    Public Overridable Sub GetObjectData(ByVal info As SerializationInfo, ByVal context As StreamingContext)
        info.AddValue("i", n1)
        info.AddValue("j", n2)
        info.AddValue("k", str)
    End Sub
End Class
```

When `GetObjectData` is called during serialization, you are responsible for populating the [SerializationInfo](#) provided with the method call. Add the variables to be serialized as name and value pairs. Any text can be used

as the name. You have the freedom to decide which member variables are added to the [SerializationInfo](#), provided that sufficient data is serialized to restore the object during deserialization. Derived classes should call the [GetObjectData](#) method on the base object if the latter implements [ISerializable](#).

It's important to stress that when [ISerializable](#) is added to a class, you must implement both [GetObjectData](#) and the special constructor. The compiler warns you if [GetObjectData](#) is missing. However, because it is impossible to enforce the implementation of a constructor, no warning is provided if the constructor is absent, and an exception is thrown when an attempt is made to deserialize a class without the constructor.

The current design was favored above a [SetObjectData](#) method to get around potential security and versioning problems. For example, a [SetObjectData](#) method must be public if it is defined as part of an interface; thus users must write code to defend against having the [SetObjectData](#) method called multiple times. Otherwise, a malicious application that calls the [SetObjectData](#) method on an object in the process of executing an operation can cause potential problems.

During deserialization, [SerializationInfo](#) is passed to the class using the constructor provided for this purpose. Any visibility constraints placed on the constructor are ignored when the object is deserialized; so you can mark the class as public, protected, internal, or private. However, it is a best practice to make the constructor protected unless the class is sealed, in which case the constructor should be marked private. The constructor should also perform thorough input validation.

To restore the state of the object, simply retrieve the values of the variables from [SerializationInfo](#) using the names used during serialization. If the base class implements [ISerializable](#), the base constructor should be called to allow the base object to restore its variables.

When you derive a new class from one that implements [ISerializable](#), the derived class must implement both the constructor as well as the [GetObjectData](#) method if it has variables that need to be serialized. The following code example shows how this is done using the [MyObject](#) class shown previously.

```
[Serializable]
public class ObjectTwo : MyObject
{
    public int num;

    public ObjectTwo()
        : base()
    {
    }

    protected ObjectTwo(SerializationInfo si, StreamingContext context)
        : base(si, context)
    {
        num = si.GetInt32("num");
    }

    public override void GetObjectData(SerializationInfo si, StreamingContext context)
    {
        base.GetObjectData(si, context);
        si.AddValue("num", num);
    }
}
```

```

<Serializable()> _
Public Class ObjectTwo
    Inherits MyObject
    Public num As Integer

    Public Sub New()

        End Sub

        Protected Sub New(ByVal si As SerializationInfo, _
        ByVal context As StreamingContext)
            MyBase.New(si, context)
            num = si.GetInt32("num")
        End Sub

        Public Overrides Sub GetObjectData(ByVal si As SerializationInfo, ByVal context As StreamingContext)
            MyBase.GetObjectData(si, context)
            si.AddValue("num", num)
        End Sub
    End Class

```

Don't forget to call the base class in the deserialization constructor. If this isn't done, the constructor on the base class is never called, and the object is not fully constructed after deserialization.

Objects are reconstructed from the inside out; and calling methods during deserialization can have undesirable side effects, because the methods called might refer to object references that have not been deserialized by the time the call is made. If the class being deserialized implements the [IDeserializationCallback](#), the [OnDeserialization](#) method is automatically called when the entire object graph has been deserialized. At this point, all the child objects referenced have been fully restored. A hash table is a typical example of a class that is difficult to deserialize without using the event listener. It is easy to retrieve the key and value pairs during deserialization, but adding these objects back to the hash table can cause problems, because there is no guarantee that classes that derived from the hash table have been deserialized. Calling methods on a hash table at this stage is therefore not advisable.

See also

- [Binary Serialization](#)
- [XML and SOAP Serialization](#)
- [Security and Serialization](#)

Steps in the serialization process

9/20/2022 • 2 minutes to read • [Edit Online](#)

When the `Serialize` method is called on a `formatter`, object serialization proceeds according to the following sequence of rules:

- A check is made to determine whether the formatter has a surrogate selector. If the formatter does, check whether the surrogate selector handles objects of the given type. If the selector handles the object type, `ISerializable.GetObjectData` is called on the surrogate selector.
- If there is no surrogate selector or if it does not handle the object type, a check is made to determine whether the object is marked with the `Serializable` attribute. If the object is not, a `SerializationException` is thrown.
- If the object is marked appropriately, check whether the object implements the `ISerializable` interface. If the object does, `GetObjectData` is called on the object.
- If the object does not implement `ISerializable`, the default serialization policy is used, serializing all fields not marked as `NonSerialized`.

WARNING

Binary serialization can be dangerous. For more information, see [BinaryFormatter security guide](#).

See also

- [Binary Serialization](#)
- [XML and SOAP Serialization](#)

Version tolerant serialization

9/20/2022 • 6 minutes to read • [Edit Online](#)

In the earliest versions of .NET Framework, creating serializable types that would be reusable from one version of an application to the next was problematic. If a type was modified by adding extra fields, the following problems would occur:

- Older versions of an application would throw exceptions when asked to deserialize new versions of the old type.
- Newer versions of an application would throw exceptions when deserializing older versions of a type with missing data.

Version Tolerant Serialization (VTS) is a set of features that makes it easier, over time, to modify serializable types. Specifically, the VTS features are enabled for classes to which the [SerializableAttribute](#) attribute has been applied, including generic types. VTS makes it possible to add new fields to those classes without breaking compatibility with other versions of the type.

The VTS features are enabled when using the [BinaryFormatter](#). Additionally, all features except extraneous data tolerance are also enabled when using the [SoapFormatter](#). For more information about using these classes for serialization, see [Binary Serialization](#).

WARNING

Binary serialization can be dangerous. For more information, see [BinaryFormatter security guide](#).

Feature list

The set of features includes the following:

- Tolerance of extraneous or unexpected data. This enables newer versions of the type to send data to older versions.
- Tolerance of missing optional data. This enables older versions to send data to newer versions.
- Serialization callbacks. This enables intelligent default value setting in cases where data is missing.

In addition, there is a feature for declaring when a new optional field has been added. This is the [VersionAdded](#) property of the [OptionalFieldAttribute](#) attribute.

These features are discussed in greater detail in the following sections.

Tolerance of extraneous or unexpected data

In the past, during deserialization, any extraneous or unexpected data caused exceptions to be thrown. With VTS, in the same situation, any extraneous or unexpected data is ignored instead of causing exceptions to be thrown. This enables applications that use newer versions of a type (that is, a version that includes more fields) to send information to applications that expect older versions of the same type.

In the following example, the extra data contained in the `CountryField` of version 2.0 of the `Address` class is ignored when an older application deserializes the newer version.

```

// Version 1 of the Address class.
[Serializable]
public class Address
{
    public string Street;
    public string City;
}

// Version 2.0 of the Address class.
[Serializable]
public class Address
{
    public string Street;
    public string City;
    // The older application ignores this data.
    public string CountryField;
}

```

```

' Version 1 of the Address class.
<Serializable> _
Public Class Address
    Public Street As String
    Public City As String
End Class

' Version 2.0 of the Address class.
<Serializable> _
Public Class Address
    Public Street As String
    Public City As String
    ' The older application ignores this data.
    Public CountryField As String
End Class

```

Tolerance of missing data

Fields can be marked as optional by applying the [OptionalFieldAttribute](#) attribute to them. During deserialization, if the optional data is missing, the serialization engine ignores the absence and does not throw an exception. Thus, applications that expect older versions of a type can send data to applications that expect newer versions of the same type.

The following example shows version 2.0 of the `Address` class with the `CountryField` field marked as optional. If an older application sends version 1 to a newer application that expects version 2.0, the absence of the data is ignored.

```

[Serializable]
public class Address
{
    public string Street;
    public string City;

    [OptionalField]
    public string CountryField;
}

```

```

<Serializable>
Public Class Address
    Public Street As String
    Public City As String

    <OptionalField>
    Public CountryField As String
End Class

```

Serialization callbacks

Serialization callbacks are a mechanism that provides hooks into the serialization/deserialization process at four points.

ATTRIBUTE	WHEN THE ASSOCIATED METHOD IS CALLED	TYPICAL USE
OnDeserializingAttribute	Before deserialization.*	Initialize default values for optional fields.
OnDeserializedAttribute	After deserialization.	Fix optional field values based on contents of other fields.
OnSerializingAttribute	Before serialization.	Prepare for serialization. For example, create optional data structures.
OnSerializedAttribute	After serialization.	Log serialization events.

* This callback is invoked before the deserialization constructor, if one is present.

Using callbacks

To use callbacks, apply the appropriate attribute to a method that accepts a [StreamingContext](#) parameter. Only one method per class can be marked with each of these attributes. For example:

```

[OnDeserializing]
private void SetCountryRegionDefault(StreamingContext sc)
{
    CountryField = "Japan";
}

```

```

<OnDeserializing>
Private Sub SetCountryRegionDefault(sc As StreamingContext)
    CountryField = "Japan"
End Sub

```

The intended use of these methods is for versioning. During deserialization, an optional field may not be correctly initialized if the data for the field is missing. This can be corrected by creating the method that assigns the correct value, then applying either the **OnDeserializingAttribute** or **OnDeserializedAttribute** attribute to the method.

The following example shows the method in the context of a type. If an earlier version of an application sends an instance of the `Address` class to a later version of the application, the `CountryField` field data will be missing. But after deserialization, the field will be set to a default value "Japan".

```
[Serializable]
public class Address
{
    public string Street;
    public string City;
    [OptionalField]
    public string CountryField;

    [OnDeserializing]
    private void SetCountryRegionDefault(StreamingContext sc)
    {
        CountryField = "Japan";
    }
}
```

```
<Serializable> _
Public Class Address
    Public Street As String
    Public City As String
    <OptionalField> _
    Public CountryField As String

    <OnDeserializing> _
    Private Sub SetCountryRegionDefault(sc As StreamingContext)
        CountryField = "Japan"
    End Sub
End Class
```

The VersionAdded property

The **OptionalFieldAttribute** has the **VersionAdded** property. The property indicates which version of a type a given field has been added. It should be incremented by exactly one (starting at 2) every time the type is modified, as shown in the following example:

```

// Version 1.0
[Serializable]
public class Person
{
    public string FullName;
}

// Version 2.0
[Serializable]
public class Person
{
    public string FullName;

    [OptionalField(VersionAdded = 2)]
    public string NickName;
    [OptionalField(VersionAdded = 2)]
    public DateTime BirthDate;
}

// Version 3.0
[Serializable]
public class Person
{
    public string FullName;

    [OptionalField(VersionAdded=2)]
    public string NickName;
    [OptionalField(VersionAdded=2)]
    public DateTime BirthDate;

    [OptionalField(VersionAdded=3)]
    public int Weight;
}

```

```

' Version 1.0
<Serializable> _
Public Class Person
    Public FullName
End Class

' Version 2.0
<Serializable> _
Public Class Person
    Public FullName As String

    <OptionalField(VersionAdded := 2)> _
    Public NickName As String
    <OptionalField(VersionAdded := 2)> _
    Public BirthDate As DateTime
End Class

' Version 3.0
<Serializable> _
Public Class Person
    Public FullName As String

    <OptionalField(VersionAdded := 2)> _
    Public NickName As String
    <OptionalField(VersionAdded := 2)> _
    Public BirthDate As DateTime

    <OptionalField(VersionAdded := 3)> _
    Public Weight As Integer
End Class

```

SerializationBinder

Some users may need to control which class to serialize and deserialize because a different version of the class is required on the server and client. [SerializationBinder](#) is an abstract class used to control the actual types used during serialization and deserialization. To use this class, derive a class from [SerializationBinder](#) and override the [BindToName](#) and [BindToType](#) methods. For more information, see [Controlling Serialization and Deserialization with SerializationBinder](#).

Best practices

To ensure proper versioning behavior, follow these rules when modifying a type from version to version:

- Never remove a serialized field.
- Never apply the [NonSerializedAttribute](#) attribute to a field if the attribute was not applied to the field in the previous version.
- Never change the name or the type of a serialized field.
- When adding a new serialized field, apply the [OptionalFieldAttribute](#) attribute.
- When removing a [NonSerializedAttribute](#) attribute from a field (that was not serializable in a previous version), apply the [OptionalFieldAttribute](#) attribute.
- For all optional fields, set meaningful defaults using the serialization callbacks unless 0 or `null` as defaults are acceptable.

To ensure that a type will be compatible with future serialization engines, follow these guidelines:

- Always set the [VersionAdded](#) property on the [OptionalFieldAttribute](#) attribute correctly.
- Avoid branched versioning.

See also

- [SerializableAttribute](#)
- [BinaryFormatter](#)
- [SoapFormatter](#)
- [VersionAdded](#)
- [OptionalFieldAttribute](#)
- [OnDeserializingAttribute](#)
- [OnDeserializedAttribute](#)
- [OnSerializingAttribute](#)
- [OnSerializedAttribute](#)
- [StreamingContext](#)
- [NonSerializedAttribute](#)
- [Binary Serialization](#)

Serialization guidelines

9/20/2022 • 11 minutes to read • [Edit Online](#)

This article lists the guidelines to consider when designing an API to be serialized.

WARNING

Binary serialization can be dangerous. For more information, see [BinaryFormatter security guide](#).

.NET offers three main serialization technologies that are optimized for various serialization scenarios. The following table lists these technologies and the main .NET types related to these technologies.

TECHNOLOGY	RELEVANT CLASSES	NOTES
Data Contract Serialization	DataContractAttribute DataMemberAttribute DataContractSerializer NetDataContractSerializer DataContractJsonSerializer ISerializable	General persistence Web Services JSON
XML Serialization	XmlSerializer	XML format with full control
Runtime -Serialization (Binary and SOAP)	SerializableAttribute ISerializable BinaryFormatter SoapFormatter	.NET Remoting

When you design new types, you should decide which, if any, of these technologies those types need to support. The following guidelines describe how to make that choice and how to provide such support. These guidelines are not meant to help you choose which serialization technology you should use in the implementation of your application or library. Such guidelines are not directly related to API design and thus are not within the scope of this topic.

Guidelines

- DO think about serialization when you design new types.

Serialization is an important design consideration for any type, because programs might need to persist or transmit instances of the type.

Choosing the right serialization technology to support

Any given type can support none, one, or more of the serialization technologies.

- CONSIDER supporting *data contract serialization* if instances of your type might need to be persisted or used in Web Services.
- CONSIDER supporting the *XML serialization* instead of or in addition to data contract serialization if you need more control over the XML format that is produced when the type is serialized.

This may be necessary in some interoperability scenarios where you need to use an XML construct that is not supported by data contract serialization, for example, to produce XML attributes.

- CONSIDER supporting *runtime serialization* if instances of your type need to travel across .NET Remoting boundaries.
- AVOID supporting runtime serialization or XML serialization just for general persistence reasons. Prefer data contract serialization instead

Data contract serialization

Types can support data contract serialization by applying the [DataContractAttribute](#) to the type and the [DataMemberAttribute](#) to the members (fields and properties) of the type.

```
[DataContract]
class Person
{
    [DataMember]
    string LastName { get; set; }
    [DataMember]
    string FirstName { get; set; }

    public Person(string firstNameValue, string lastNameValue)
    {
        FirstName = firstNameValue;
        LastName = lastNameValue;
    }
}
```

```
<DataContract()> Public Class Person
    <DataMember()> Public Property LastName As String
    <DataMember()> Public Property FirstName As String

    Public Sub New(ByVal firstNameValue As String, ByVal lastNameValue As String)
        FirstName = firstNameValue
        LastName = lastNameValue
    End Sub

End Class
```

1. CONSIDER marking data members of your type public if the type can be used in partial trust. In full trust, data contract serializers can serialize and deserialize nonpublic types and members, but only public members can be serialized and deserialized in partial trust.
2. DO implement a getter and setter on all properties that have Data-MemberAttribute. Data contract serializers require both the getter and the setter for the type to be considered serializable. If the type won't be used in partial trust, one or both of the property accessors can be nonpublic.

```

[DataContract]
class Person2
{
    string lastName;
    string firstName;

    public Person2(string firstName, string lastName)
    {
        this.lastName = lastName;
        this.firstName = firstName;
    }

    [DataMember]
    public string LastName
    {
        // Implement get and set.
        get { return lastName; }
        private set { lastName = value; }
    }

    [DataMember]
    public string FirstName
    {
        // Implement get and set.
        get { return firstName; }
        private set { firstName = value; }
    }
}

```

```

<DataContract()> Class Person2
    Private lastNameValue As String
    Private firstNameValue As String

    Public Sub New(ByVal firstName As String, ByVal lastName As String)
        Me.lastNameValue = lastName
        Me.firstNameValue = firstName
    End Sub

    <DataMember()> Property LastName As String
        Get
            Return lastNameValue
        End Get

        Set(ByVal value As String)
            lastNameValue = value
        End Set

    End Property

    <DataMember()> Property FirstName As String
        Get
            Return firstNameValue
        End Get

        Set(ByVal value As String)
            firstNameValue = value
        End Set
    End Property

End Class

```

3. CONSIDER using the serialization callbacks for initialization of deserialized instances.

Constructors are not called when objects are deserialized. Therefore, any logic that executes during

normal construction needs to be implemented as one of the *serialization callbacks*.

```
[DataContract]
class Person3
{
    [DataMember]
    string lastName;
    [DataMember]
    string firstName;
    string fullName;

    public Person3(string firstName, string lastName)
    {
        // This constructor is not called during deserialization.
        this.lastName = lastName;
        this.firstName = firstName;
        fullName = firstName + " " + lastName;
    }

    public string FullName
    {
        get { return fullName; }
    }

    // This method is called after the object
    // is completely deserialized. Use it instead of the
    // constructor.
    [OnDeserialized]
    void OnDeserialized(StreamingContext context)
    {
        fullName = firstName + " " + lastName;
    }
}
```

```
<DataContract()> _
Class Person3
    <DataMember()> Private lastNameValue As String
    <DataMember()> Private firstNameValue As String
    Dim fullNameValue As String

    Public Sub New(ByVal firstName As String, ByVal lastName As String)
        lastNameValue = lastName
        firstNameValue = firstName
        fullNameValue = firstName & " " & lastName
    End Sub

    Public ReadOnly Property FullName As String
        Get
            Return fullNameValue
        End Get
    End Property

    <OnDeserialized()> Sub OnDeserialized(ByVal context As StreamingContext)
        fullNameValue = firstNameValue & " " & lastNameValue
    End Sub
End Class
```

The [OnDeserializedAttribute](#) attribute is the most commonly used callback attribute. The other attributes in the family are [OnDeserializingAttribute](#), [OnSerializingAttribute](#), and [OnSerializedAttribute](#). They can be used to mark callbacks that get executed before deserialization, before serialization, and finally, after serialization, respectively.

4. CONSIDER using the [KnownTypeAttribute](#) to indicate concrete types that should be used when

deserializing a complex object graph.

For example, if a type of a deserialized data member is represented by an abstract class, the serializer will need the *known type* information to decide what concrete type to instantiate and assign to the member. If the known type is not specified using the attribute, it will need to be passed to the serializer explicitly (you can do it by passing known types to the serializer constructor) or it will need to be specified in the configuration file.

```
// The KnownTypeAttribute specifies types to be
// used during serialization.
[KnownType(typeof(USAddress))]
[DataContract]
class Person4
{
    [DataMember]
    string fullNameValue;
    [DataMember]
    Address address; // Address is abstract

    public Person4(string fullName, Address address)
    {
        this.fullNameValue = fullName;
        this.address = address;
    }

    public string FullName
    {
        get { return fullNameValue; }
    }
}

[DataContract]
public abstract class Address
{
    public abstract string FullAddress { get; }
}

[DataContract]
public class USAddress : Address
{
    [DataMember]
    public string Street { get; set; }
    [DataMember]
    public string City { get; set; }
    [DataMember]
    public string State { get; set; }
    [DataMember]
    public string ZipCode { get; set; }

    public override string FullAddress
    {
        get
        {
            return Street + "\n" + City + ", " + State + " " + ZipCode;
        }
    }
}
```

```

<KnownType(GetType(USAddress)), _
DataContract()> Class Person4
    <DataMember()> Property fullNameValue As String
    <DataMember()> Property addressValue As USAddress ' Address is abstract

    Public Sub New(ByVal fullName As String, ByVal address As Address)
        fullNameValue = fullName
        addressValue = address
    End Sub

    Public ReadOnly Property FullName() As String
        Get
            Return fullNameValue
        End Get

        End Property
    End Class

    <DataContract()> Public MustInherit Class Address
        Public MustOverride Function FullAddress() As String
    End Class

    <DataContract()> Public Class USAddress
        Inherits Address
        <DataMember()> Public Property Street As String
        <DataMember()> Public City As String
        <DataMember()> Public State As String
        <DataMember()> Public ZipCode As String

        Public Overrides Function FullAddress() As String
            Return Street & "\n" & City & ", " & State & " " & ZipCode
        End Function
    End Class

```

In cases where the list of known types is not known statically (when the **Person** class is compiled), the **KnownTypeAttribute** can also point to a method that returns a list of known types at run time.

5. DO consider backward and forward compatibility when creating or changing serializable types.

Keep in mind that serialized streams of future versions of your type can be deserialized into the current version of the type, and vice versa. Make sure you understand that data members, even private and internal, cannot change their names, types, or even their order in future versions of the type unless special care is taken to preserve the contract using explicit parameters to the data contract attributes. Test compatibility of serialization when making changes to serializable types. Try deserializing the new version into an old version, and vice versa.

6. CONSIDER implementing [IExtensibleDataObject](#) interface to allow round-tripping between different versions of the type.

The interface allows the serializer to ensure that no data is lost during round-tripping. The [ExtensionData](#) property stores any data from the future version of the type that is unknown to the current version. When the current version is subsequently serialized and deserialized into a future version, the additional data will be available in the serialized stream through the [ExtensionData](#) property value.

```
// Implement the IExtensibleDataObject interface.
[DataContract]
class Person5 : IExtensibleDataObject
{
    ExtensionDataObject serializationData;
    [DataMember]
    string fullNameValue;

    public Person5(string fullName)
    {
        this.fullNameValue = fullName;
    }

    public string FullName
    {
        get { return fullNameValue; }
    }

    ExtensionDataObject IExtensibleDataObject.ExtensionData
    {
        get
        {
            return serializationData;
        }
        set { serializationData = value; }
    }
}
```

```
<DataContract()> Class Person5
    Implements IExtensibleDataObject
    <DataMember()> Dim fullNameValue As String

    Public Sub New(ByVal fullName As String)
        fullName = fullName
    End Sub

    Public ReadOnly Property FullName
        Get
            Return fullNameValue
        End Get
    End Property
    Private serializationData As ExtensionDataObject
    Public Property ExtensionData As ExtensionDataObject Implements
        IExtensibleDataObject.ExtensionData
        Get
            Return serializationData
        End Get
        Set(ByVal value As ExtensionDataObject)
            serializationData = value
        End Set
    End Property
End Class
```

For more information, see [Forward-Compatible Data Contracts](#).

XML serialization

Data contract serialization is the main (default) serialization technology in .NET Framework, but there are serialization scenarios that data contract serialization does not support. For example, it does not give you full control over the shape of XML produced or consumed by the serializer. If such fine control is required, *XML serialization* has to be used, and you need to design your types to support this serialization technology.

1. AVOID designing your types specifically for XML Serialization, unless you have a very strong reason to control the shape of the XML produced. This serialization technology has been superseded by the Data

Contract Serialization discussed in the previous section.

In other words, don't apply attributes from the [System.Xml.Serialization](#) namespace to new types, unless you know that the type will be used with XML Serialization. The following example shows how [System.Xml.Serialization](#) can be used to control the shape of the XML -produced.

```
public class Address2
{
    [System.Xml.Serialization.XmlAttribute] // Serialize as XML attribute, instead of an element.
    public string Name { get { return "Poe, Toni"; } set { } }

    [System.Xml.Serialization.XmlElement(ElementName = "StreetLine")] // Explicitly name the element.
    public string Street = "1 Main Street";
}
```

```
Public Class Address2
    ' Supports XML Serialization.
    <System.Xml.Serialization.XmlAttribute()> _
    Public ReadOnly Property Name As String ' Serialize as XML attribute, instead of an element.
        Get
            Return "Poe, Toni"
        End Get
    End Property
    <System.Xml.Serialization.XmlElement(ElementName:="StreetLine")> _
    Public Street As String = "1 Main Street" ' Explicitly names the element 'StreetLine'.
End Class
```

2. CONSIDER implementing the [IXmlSerializable](#) interface if you want even more control over the shape of the serialized XML than what's offered by applying the XML Serialization attributes. Two methods of the interface, [ReadXml](#) and [WriteXml](#), allow you to fully control the serialized XML stream. You can also control the XML schema that gets generated for the type by applying the [XmlSchemaProviderAttribute](#) attribute.

Runtime serialization

Runtime serialization is a technology used by .NET Remoting. If you think your types will be transported using .NET Remoting, make sure they support runtime serialization.

The basic support for *runtime serialization* can be provided by applying the [SerializableAttribute](#) attribute, and more advanced scenarios involve implementing a simple *runtime serializable pattern* (implement [ISerializable](#) and provide a serialization constructor).

1. CONSIDER supporting runtime serialization if your types will be used with .NET Remoting. For example, the [System.AddIn](#) namespace uses .NET Remoting, and so all types exchanged between [System.AddIn](#) add-ins need to support runtime serialization.

```
// Apply SerializableAttribute to support runtime serialization.
[Serializable]
public class Person6
{
    // Code not shown.
}
```

```
<Serializable()> Public Class Person6 ' Support runtime serialization with the SerializableAttribute.

    ' Code not shown.
End Class
```

2. CONSIDER implementing the *runtime serializable pattern* if you want complete control over the

serialization process. For example, if you want to transform data as it gets serialized or deserialized.

The pattern is very simple. All you need to do is implement the [ISerializable](#) interface and provide a special constructor that is used when the object is deserialized.

```
// Implement the ISerializable interface for more control.
[Serializable]
public class Person_Runtime_Serializable : ISerializable
{
    string fullName;

    public Person_Runtime_Serializable() { }

    protected Person_Runtime_Serializable(SerializationInfo info, StreamingContext context){
        if (info == null) throw new System.ArgumentNullException("info");
        fullName = (string)info.GetValue("name", typeof(string));
    }

    void ISerializable.GetObjectData(SerializationInfo info,
        StreamingContext context) {
        if (info == null) throw new System.ArgumentNullException("info");
        info.AddValue("name", fullName);
    }

    public string FullName
    {
        get { return fullName; }
        set { fullName = value; }
    }
}
```

```

' Implement the ISerializable interface for more control.
<Serializable()> Public Class Person_Runtime_Serializable
    Implements ISerializable

    Private fullNameValue As String

    Public Sub New()
        ' empty constructor.
    End Sub
    Protected Sub New(ByVal info As SerializationInfo, _
                    ByVal context As StreamingContext)
        If info Is Nothing Then
            Throw New System.ArgumentNullException("info")
            FullName = CType(info.GetValue("name", GetType(String)), String)
        End If
    End Sub

    Private Sub GetObjectData(ByVal info As SerializationInfo, _
                            ByVal context As StreamingContext) _
                            Implements ISerializable.GetObjectData
        If info Is Nothing Then
            Throw New System.ArgumentNullException("info")
        End If
        info.AddValue("name", FullName)
    End Sub

    Public Property FullName As String
        Get
            Return fullNameValue
        End Get
        Set(ByVal value As String)
            fullNameValue = value
        End Set
    End Property
End Class

```

3. DO make the serialization constructor protected and provide two parameters typed and named exactly as shown in the sample here.

```
protected Person_Runtime_Serializable(SerializationInfo info, StreamingContext context){
```

```
Protected Sub New(ByVal info As SerializationInfo, _
                 ByVal context As StreamingContext)
```

4. DO implement the ISerializable members explicitly.

```
void ISerializable.GetObjectData(SerializationInfo info,
                                StreamingContext context) {
```

```
Private Sub GetObjectData(ByVal info As SerializationInfo, _
                        ByVal context As StreamingContext) _
                        Implements ISerializable.GetObjectData
```

See also

- [Using Data Contracts](#)
- [Data Contract Serializer](#)
- [Types Supported by the Data Contract Serializer](#)
- [Binary Serialization](#)
- [.NET Remoting](#)
- [XML and SOAP Serialization](#)
- [Security and Serialization](#)

How to: chunk serialized data

9/20/2022 • 5 minutes to read • [Edit Online](#)

WARNING

Binary serialization can be dangerous. For more information, see [BinaryFormatter security guide](#).

Two issues that occur when sending large data sets in Web service messages are:

1. A large working set (memory) due to buffering by the serialization engine.
2. Inordinate bandwidth consumption due to 33 percent inflation after Base64 encoding.

To solve these problems, implement the [IXmlSerializable](#) interface to control the serialization and deserialization. Specifically, implement the [WriteXml](#) and [ReadXml](#) methods to chunk the data.

To implement server-side chunking

1. On the server machine, the Web method must turn off ASP.NET buffering and return a type that implements [IXmlSerializable](#).
2. The type that implements [IXmlSerializable](#) chunks the data in the [WriteXml](#) method.

To implement client-side processing

1. Alter the Web method on the client proxy to return the type that implements [IXmlSerializable](#). You can use a [SchemaImporterExtension](#) to do this automatically, but this isn't shown here.
2. Implement the [ReadXml](#) method to read the chunked data stream and write the bytes to disk. This implementation also raises progress events that can be used by a graphic control, such as a progress bar.

Example

The following code example shows the Web method on the client that turns off ASP.NET buffering. It also shows the client-side implementation of the [IXmlSerializable](#) interface that chunks the data in the [WriteXml](#) method.

```
[WebMethod]
[SoapDocumentMethod(ParameterStyle = SoapParameterStyle.Bare)]
public SongStream DownloadSong(DownloadAuthorization Authorization, string filePath)
{
    // Turn off response buffering.
    System.Web.HttpContext.Current.Response.Buffer = false;
    // Return a song.
    SongStream song = new SongStream(filePath);
    return song;
}
```

```

<WebMethod(), SoapDocumentMethodAttribute(ParameterStyle:=SoapParameterStyle.Bare)>
Public Function DownloadSong(ByVal Authorization As DownloadAuthorization, ByVal filePath As String) As
SongStream

    ' Turn off response buffering.
    System.Web.HttpContext.Current.Response.Buffer = False
    ' Return a song.
    Dim song As New SongStream(filePath)
    Return song

End Function
End Class

```

```

[XmlSchemaProvider("MySchema")]
public class SongStream : IXmlSerializable
{
    private const string ns = "http://demos.Contoso.com/webservices";
    private string filePath;

    public SongStream() { }

    public SongStream(string filePath)
    {
        this.filePath = filePath;
    }

    // This is the method named by the XmlSchemaProviderAttribute applied to the type.
    public static XmlQualifiedName MySchema(XmlSchemaSet xs)
    {
        // This method is called by the framework to get the schema for this type.
        // We return an existing schema from disk.

        XmlSerializer schemaSerializer = new XmlSerializer(typeof(XmlSchema));
        string xsdPath = null;
        // NOTE: replace the string with your own path.
        xsdPath = System.Web.HttpContext.Current.Server.MapPath("SongStream.xsd");
        XmlSchema s = (XmlSchema)schemaSerializer.Deserialize(
            new XmlTextReader(xsdPath), null);
        xs.XmlResolver = new XmlUrlResolver();
        xs.Add(s);

        return new XmlQualifiedName("songStream", ns);
    }

    void IXmlSerializable.WriteXml(System.Xml.XmlWriter writer)
    {
        // This is the chunking code.
        // ASP.NET buffering must be turned off for this to work.

        int bufferSize = 4096;
        char[] songBytes = new char[bufferSize];
        FileStream inFile = File.Open(this.filePath, FileMode.Open, FileAccess.Read);

        long length = inFile.Length;

        // Write the file name.
        writer.WriteElementString("fileName", ns, Path.GetFileNameWithoutExtension(this.filePath));

        // Write the size.
        writer.WriteElementString("size", ns, length.ToString());

        // Write the song bytes.
        writer.WriteStartElement("song", ns);

        StreamReader sr = new StreamReader(inFile, true);
        int readLen = sr.Read(songBytes, 0, bufferSize);
    }
}

```

```

        while (readLen > 0)
    {
        writer.WriteStartElement("chunk", ns);
        writer.WriteChars(songBytes, 0, readLen);
        writer.WriteEndElement();

        writer.Flush();
        readLen = sr.Read(songBytes, 0, bufferSize);
    }

    writer.WriteEndElement();
    inFile.Close();
}

XmlSchema IXmlSerializable.GetSchema()
{
    throw new NotImplementedException();
}

void IXmlSerializable.ReadXml(System.Xml.XmlReader reader)
{
    throw new NotImplementedException();
}
}

```

```

<XmlSchemaProvider("MySchema")>
Public Class SongStream
    Implements IXmlSerializable

    Private Const ns As String = "http://demos.Contoso.com/webservices"
    Private filePath As String

    Public Sub New()

    End Sub

    Public Sub New(ByVal filePath As String)
        Me.filePath = filePath
    End Sub

    ' This is the method named by the XmlSchemaProviderAttribute applied to the type.
    Public Shared Function MySchema(ByVal xs As XmlSchemaSet) As XmlQualifiedName
        ' This method is called by the framework to get the schema for this type.
        ' We return an existing schema from disk.
        Dim schemaSerializer As New XmlSerializer(GetType(XmlSchema))
        Dim xsdPath As String = Nothing
        ' NOTE: replace SongStream.xsd with your own schema file.
        xsdPath = System.Web.HttpContext.Current.Server.MapPath("SongStream.xsd")
        Dim s As XmlSchema = CType(schemaSerializer.Deserialize(New XmlTextReader(xsdPath)), XmlSchema)
        xs.XmlResolver = New XmlUrlResolver()
        xs.Add(s)

        Return New XmlQualifiedName("songStream", ns)
    End Function

    Sub WriteXml(ByVal writer As System.Xml.XmlWriter) Implements IXmlSerializable.WriteXml
        ' This is the chunking code.
        ' ASP.NET buffering must be turned off for this to work.

        Dim bufferSize As Integer = 4096
        Dim songBytes(bufferSize) As Char
        Dim inFile As FileStream = File.Open(Me.filePath, FileMode.Open, FileAccess.Read)

        Dim length As Long = inFile.Length

```

```

' Write the file name.
writer.WriteElementString("fileName", ns, Path.GetFileNameWithoutExtension(Me.filePath))

' Write the size.
writer.WriteElementString("size", ns, length.ToString())

' Write the song bytes.
writer.WriteStartElement("song", ns)

Dim sr As New StreamReader(inFile, True)
Dim readLen As Integer = sr.Read(songBytes, 0, bufferSize)

While readLen > 0
    writer.WriteStartElement("chunk", ns)
    writer.WriteChars(songBytes, 0, readLen)
    writer.WriteEndElement()

    writer.Flush()
    readLen = sr.Read(songBytes, 0, bufferSize)
End While

writer.WriteEndElement()
inFile.Close()
End Sub

Function GetSchema() As System.Xml.Schema.XmlSchema Implements IXmlSerializable.GetSchema
    Throw New System.NotImplementedException()
End Function

Sub ReadXml(ByVal reader As System.Xml.XmlReader) Implements IXmlSerializable.ReadXml
    Throw New System.NotImplementedException()
End Sub
End Class

```

```

public class SongFile : IXmlSerializable
{
    public static event ProgressMade OnProgress;

    public SongFile()
    { }

    private const string ns = "http://demos.teched2004.com/webservices";
    public static string MusicPath;
    private string filePath;
    private double size;

    void IXmlSerializable.ReadXml(System.Xml.XmlReader reader)
    {
        reader.ReadStartElement("DownloadSongResult", ns);
        ReadFileName(reader);
        ReadSongSize(reader);
        ReadAndSaveSong(reader);
        reader.ReadEndElement();
    }

    void ReadFileName(XmlReader reader)
    {
        string fileName = reader.ReadElementString("fileName", ns);
        this.filePath =
            Path.Combine(MusicPath, Path.ChangeExtension(fileName, ".mp3"));
    }

    void ReadSongSize(XmlReader reader)
    {
        this.size = Convert.ToDouble(reader.ReadElementString("size", ns));
    }
}

```

```

void ReadAndSaveSong(XmlReader reader)
{
    FileStream outFile = File.Open(
        this.filePath, FileMode.Create, FileAccess.Write);

    string songBase64;
    byte[] songBytes;
    reader.ReadStartElement("song", ns);
    double totalRead = 0;
    while (true)
    {
        if (reader.IsStartElement("chunk", ns))
        {
            songBase64 = reader.ReadElementString();
            totalRead += songBase64.Length;
            songBytes = Convert.FromBase64String(songBase64);
            outFile.Write(songBytes, 0, songBytes.Length);
            outFile.Flush();

            if (OnProgress != null)
            {
                OnProgress(100 * (totalRead / size));
            }
        }

        else
        {
            break;
        }
    }

    outFile.Close();
    reader.ReadEndElement();
}

public void Play()
{
    System.Diagnostics.Process.Start(this.filePath);
}

XmlSchema IXmlSerializable.GetSchema()
{
    throw new NotImplementedException();
}

public void WriteXml(XmlWriter writer)
{
    throw new NotImplementedException();
}
}

```

```

Public Class SongFile
    Implements IXmlSerializable
    Public Shared Event OnProgress As ProgressMade

    Public Sub New()

    End Sub

    Private Const ns As String = "http://demos.teched2004.com/webservices"
    Public Shared MusicPath As String
    Private filePath As String
    Private size As Double

    Sub ReadXml(ByVal reader As System.Xml.XmlReader) Implements IXmlSerializable.ReadXml
        reader.ReadStartElement("DownloadSongResult", ns)
        Dim song As Song
        song = New Song()
        song.Title = reader.ReadElementString("Title")
        song.Artist = reader.ReadElementString("Artist")
        song.Genre = reader.ReadElementString("Genre")
        song.Length = reader.ReadElementString("Length")
        song.BPM = reader.ReadElementString("BPM")
        song.KeySignature = reader.ReadElementString("KeySignature")
        song.Lyrics = reader.ReadElementString("Lyrics")
        song.Base64 = reader.ReadElementString("Base64")
        songBytes = Convert.FromBase64String(song.Base64)
        song.FilePath = filePath & song.Title & ".mp3"
        songBytes = songBytes
        File.WriteAllBytes(song.FilePath, songBytes)
        OnProgress(100 * (size / song.Length))
    End Sub

```

```

    ReadFileName(reader)
    ReadSongSize(reader)
    ReadAndSaveSong(reader)
    reader.ReadEndElement()
End Sub

Sub ReadFileName(ByVal reader As XmlReader)
    Dim fileName As String = reader.ReadElementString("fileName", ns)
    Me.filePath = Path.Combine(MusicPath, Path.ChangeExtension(fileName, ".mp3"))

End Sub

Sub ReadSongSize(ByVal reader As XmlReader)
    Me.size = Convert.ToDouble(reader.ReadElementString("size", ns))

End Sub

Sub ReadAndSaveSong(ByVal reader As XmlReader)
    Dim outFile As FileStream = File.Open(Me.filePath, FileMode.Create, FileAccess.Write)

    Dim songBase64 As String
    Dim songBytes() As Byte
    reader.ReadStartElement("song", ns)
    Dim totalRead As Double = 0
    While True
        If reader.IsStartElement("chunk", ns) Then
            songBase64 = reader.ReadElementString()
            totalRead += songBase64.Length
            songBytes = Convert.FromBase64String(songBase64)
            outFile.Write(songBytes, 0, songBytes.Length)
            outFile.Flush()
            RaiseEvent OnProgress((100 * (totalRead / size)))
        Else
            Exit While
        End If
    End While

    outFile.Close()
    reader.ReadEndElement()
End Sub

Public Sub Play()
    System.Diagnostics.Process.Start(Me.filePath)
End Sub

Function GetSchema() As System.Xml.Schema.XmlSchema Implements IXmlSerializable.GetSchema
    Throw New System.NotImplementedException()
End Function

Public Sub WriteXml(ByVal writer As XmlWriter) Implements IXmlSerializable.WriteXml
    Throw New System.NotImplementedException()
End Sub
End Class

```

Compiling the code

- The code uses the following namespaces: [System](#), [System.Runtime.Serialization](#), [System.Web.Services](#), [System.Web.Services.Protocols](#), [System.Xml](#), [System.Xml.Serialization](#), and [System.Xml.Schema](#).

See also

- [Custom Serialization](#)

How to determine if a .NET Standard object is serializable

9/20/2022 • 2 minutes to read • [Edit Online](#)

.NET Standard is a specification that defines the types and members that must be present on specific .NET implementations that conform to that version of the standard. However, .NET Standard does not define whether a type is serializable. The types defined in the .NET Standard Library are not marked with the `SerializableAttribute` attribute. Instead, specific .NET implementations, such as .NET Framework and .NET Core, are free to determine whether a particular type is serializable.

If you've developed a library that targets .NET Standard, your library can be consumed by any .NET implementation that supports .NET Standard. This means that you cannot know in advance whether a particular type is serializable; you can only determine whether it is serializable at run time.

You can determine whether an object is serializable at run time by retrieving the value of the `IsSerializable` property of a `Type` object that represents that object's type. The following example provides one implementation. It defines an `IsSerializable(Object)` extension method that indicates whether any `Object` instance can be serialized.

```
namespace Libraries
{
    using System;

    public static class UtilityLibrary
    {
        public static bool IsSerializable(this object obj)
        {
            if (obj == null)
                return false;

            Type t = obj.GetType();
            return t.IsSerializable;
        }
    }
}
```

```
Imports System.Runtime.CompilerServices

Namespace Global.Libraries

    Public Module UtilityLibrary
        <Extension>
        Public Function IsSerializable(obj As Object) As Boolean
            If obj Is Nothing Then Return False

            Dim t As Type = obj.GetType()
            Return t.IsSerializable
        End Function
    End Module
End Namespace
```

You can then pass any object to the method to determine whether it can be serialized and deserialized on the current .NET implementation, as the following example shows:

```

using System;
using System.IO;
using System.Runtime.Serialization.Formatters.Binary;
using Libraries;

namespace test_serialization
{
    class Program
    {
        static void Main()
        {
            var value = ValueTuple.Create("03244562", DateTime.Now, 13452.50m);
            if (value.IsSerializable())
            {
                // Serialize the value tuple.
                var formatter = new BinaryFormatter();
                using (var stream = new FileStream("data.bin", FileMode.Create,
                    FileAccess.Write, FileShare.None))
                {
                    formatter.Serialize(stream, value);
                }
                // Deserialize the value tuple.
                using (var readStream = new FileStream("data.bin", FileMode.Open))
                {
                    object restoredValue = formatter.Deserialize(readStream);
                    Console.WriteLine($"{restoredValue.GetType().Name}: {restoredValue}");
                }
            }
            else
            {
                Console.WriteLine($"{nameof(value)} is not serializable");
            }
        }
    }
}

// The example displays output like the following:
//    ValueTuple`3: (03244562, 10/18/2017 5:25:22 PM, 13452.50)

```

```

Imports System.IO
Imports System.Runtime.Serialization.Formatters.Binary
Imports Libraries

Module Program
    Sub Main()
        Dim value = ValueTuple.Create("03244562", DateTime.Now, 13452.50d)
        If value.IsSerializable() Then
            Dim formatter As New BinaryFormatter()
            ' Serialize the value tuple.
            Using stream As New FileStream("data.bin", FileMode.Create,
                FileAccess.Write, FileShare.None)
                formatter.Serialize(stream, value)
            End Using
            ' Deserialize the value tuple.
            Using readStream As New FileStream("data.bin", FileMode.Open)
                Dim restoredValue = formatter.Deserialize(readStream)
                Console.WriteLine($"{restoredValue.GetType().Name}: {restoredValue}")
            End Using
        Else
            Console.WriteLine($"{nameof(value)} is not serializable")
        End If
    End Sub
End Module

' The example displays output like the following:
'    ValueTuple`3: (03244562, 10/18/2017 5:25:22 PM, 13452.50)

```

See also

- [Binary serialization](#)
- [System.SerializableAttribute](#)
- [Type.IsSerializable](#)

XML and SOAP serialization

9/20/2022 • 2 minutes to read • [Edit Online](#)

XML serialization converts (serializes) the public fields and properties of an object, and the parameters and return values of methods, into an XML stream that conforms to a specific XML Schema definition language (XSD) document. XML serialization results in strongly typed classes with public properties and fields that are converted to a serial format (in this case, XML) for storage or transport.

Because XML is an open standard, the XML stream can be processed by any application, as needed, regardless of platform. For example, XML Web services created using ASP.NET use the [XmlSerializer](#) class to create XML streams that pass data between XML Web service applications throughout the Internet or on intranets. Conversely, deserialization takes such an XML stream and reconstructs the object.

XML serialization can also be used to serialize objects into XML streams that conform to the SOAP specification. SOAP is a protocol based on XML, designed specifically to transport procedure calls using XML.

To serialize or deserialize objects, use the [XmlSerializer](#) class. To create the classes to be serialized, use the XML Schema Definition tool.

See also

- [Binary Serialization](#)
- [XML Web Services created using ASP.NET and XML Web Service clients](#)

XML serialization

9/20/2022 • 9 minutes to read • [Edit Online](#)

Serialization is the process of converting an object into a form that can be readily transported. For example, you can serialize an object and transport it over the Internet using HTTP between a client and a server. On the other end, deserialization reconstructs the object from the stream.

XML serialization serializes only the public fields and property values of an object into an XML stream. XML serialization does not include type information. For example, if you have a **Book** object that exists in the **Library** namespace, there is no guarantee that it is deserialized into an object of the same type.

NOTE

XML serialization does not convert methods, indexers, private fields, or read-only properties (except read-only collections). To serialize all an object's fields and properties, both public and private, use the [DataContractSerializer](#) instead of XML serialization.

The central class in XML serialization is the [XmlSerializer](#) class, and the most important methods in this class are the **Serialize** and **Deserialize** methods. The [XmlSerializer](#) creates C# files and compiles them into .dll files to perform this serialization. The [XML Serializer Generator Tool \(Sgen.exe\)](#) is designed to generate these serialization assemblies in advance to be deployed with your application and improve startup performance. The XML stream generated by the [XmlSerializer](#) is compliant with the World Wide Web Consortium (W3C) [XML Schema definition language \(XSD\) 1.0 recommendation](#). Furthermore, the data types generated are compliant with the document titled "XML Schema Part 2: Datatypes."

The data in your objects is described using programming language constructs like classes, fields, properties, primitive types, arrays, and even embedded XML in the form of [XmlElement](#) or [XmlAttribute](#) objects. You have the option of creating your own classes, annotated with attributes, or using the XML Schema Definition tool to generate the classes based on an existing XML Schema.

If you have an XML Schema, you can run the XML Schema Definition tool to produce a set of classes that are strongly typed to the schema and annotated with attributes. When an instance of such a class is serialized, the generated XML adheres to the XML Schema. Provided with such a class, you can program against an easily manipulated object model while being assured that the generated XML conforms to the XML schema. This is an alternative to using other classes in .NET, such as the [XmlReader](#) and [XmlWriter](#) classes, to parse and write an XML stream. For more information, see [XML Documents and Data](#). These classes allow you to parse any XML stream. In contrast, use the [XmlSerializer](#) when the XML stream is expected to conform to a known XML Schema.

Attributes control the XML stream generated by the [XmlSerializer](#) class, allowing you to set the XML namespace, element name, attribute name, and so on, of the XML stream. For more information about these attributes and how they control XML serialization, see [Controlling XML Serialization Using Attributes](#). For a table of those attributes that are used to control the generated XML, see [Attributes That Control XML Serialization](#).

The [XmlSerializer](#) class can further serialize an object and generate an encoded SOAP XML stream. The generated XML adheres to section 5 of the World Wide Web Consortium document titled "Simple Object Access Protocol (SOAP) 1.1." For more information about this process, see [How to: Serialize an Object as a SOAP-Encoded XML Stream](#). For a table of the attributes that control the generated XML, see [Attributes That Control Encoded SOAP Serialization](#).

The [XmlSerializer](#) class generates the SOAP messages created by, and passed to, XML Web services. To control

the SOAP messages, you can apply attributes to the classes, return values, parameters, and fields found in an XML Web service file (.asmx). You can use both the attributes listed in "Attributes That Control XML Serialization" and "Attributes That Control Encoded SOAP Serialization" because an XML Web service can use either the literal or encoded SOAP style. For more information about using attributes to control the XML generated by an XML Web service, see [XML Serialization with XML Web Services](#). For more information about SOAP and XML Web services, see [Customizing SOAP Message Formatting](#).

Security Considerations for XmlSerializer Applications

When creating an application that uses the **XmlSerializer**, be aware of the following items and their implications:

- The **XmlSerializer** creates C# (.cs) files and compiles them into .dll files in the directory named by the TEMP environment variable; serialization occurs with those DLLs.

NOTE

These serialization assemblies can be generated in advance and signed by using the SGGen.exe tool. This does not work on a server of Web services. In other words, it is only for client use and for manual serialization.

The code and the DLLs are vulnerable to a malicious process at the time of creation and compilation. It might be possible for two or more users to share the TEMP directory. Sharing a TEMP directory is dangerous if the two accounts have different security privileges and the higher-privilege account runs an application using the **XmlSerializer**. In this case, one user can breach the computer's security by replacing either the .cs or .dll file that is compiled. To eliminate this concern, always be sure that each account on the computer has its own profile. By default, the TEMP environment variable points to a different directory for each account.

- If a malicious user sends a continuous stream of XML data to a Web server (a denial of service attack), then the **XmlSerializer** continues to process the data until the computer runs low on resources.

This kind of attack is eliminated if you are using a computer running Internet Information Services (IIS), and your application is running within IIS. IIS features a gate that does not process streams longer than a set amount (the default is 4 KB). If you create an application that does not use IIS and deserializes with the **XmlSerializer**, you should implement a similar gate that prevents a denial of service attack.

- The **XmlSerializer** serializes data and runs any code using any type given to it.

There are two ways in which a malicious object presents a threat. It could run malicious code or it could inject malicious code into the C# file created by the **XmlSerializer**. In the second case, there is a theoretical possibility that a malicious object may somehow inject code into the C# file created by the **XmlSerializer**. Although this issue has been examined thoroughly, and such an attack is considered unlikely, you should take the precaution of never serializing data with an unknown and untrusted type.

- Serialized sensitive data might be vulnerable.

After the **XmlSerializer** has serialized data, it can be stored as an XML file or other data store. If your data store is available to other processes, or is visible on an intranet or the Internet, the data can be stolen and used maliciously. For example, if you create an application that serializes orders that include credit card numbers, the data is highly sensitive. To help prevent this, always protect the store for your data and take steps to keep it private.

Serialization of a Simple Class

The following code example shows a basic class with a public field.

```
Public Class OrderForm
    Public OrderDate As DateTime
End Class
```

```
public class OrderForm
{
    public DateTime OrderDate;
}
```

When an instance of this class is serialized, it might resemble the following.

```
<OrderForm>
    <OrderDate>12/12/01</OrderDate>
</OrderForm>
```

For more examples of serialization, see [Examples of XML Serialization](#).

Items That Can Be Serialized

The following items can be serialized using the **XmlSerializer** class:

- Public read/write properties and fields of public classes.
- Classes that implement **ICollection** or **IEnumerable**.

NOTE

Only collections are serialized, not public properties.

- **XmlElement** objects.
- **XmlNode** objects.
- **DataSet** objects.

For more information about serializing or deserializing objects, see [How to: Serialize an Object](#) and [How to: Deserialize an Object](#).

Advantages of Using XML Serialization

The **XmlSerializer** class gives you complete and flexible control when you serialize an object as XML. If you are creating an XML Web service, you can apply attributes that control serialization to classes and members to ensure that the XML output conforms to a specific schema.

For example, **XmlSerializer** enables you to:

- Specify whether a field or property should be encoded as an attribute or an element.
- Specify an XML namespace to use.
- Specify the name of an element or attribute if a field or property name is inappropriate.

Another advantage of XML serialization is that you have no constraints on the applications you develop, as long as the XML stream that is generated conforms to a given schema. Imagine a schema that is used to describe books. It features a title, author, publisher, and ISBN number element. You can develop an application that processes the XML data in any way you want, for example, as a book order, or as an inventory of books. In either

case, the only requirement is that the XML stream conforms to the specified XML Schema definition language (XSD) schema.

XML Serialization Considerations

The following should be considered when using the **XmlSerializer** class:

- The Sgen.exe tool is expressly designed to generate serialization assemblies for optimum performance.
- The serialized data contains only the data itself and the structure of your classes. Type identity and assembly information are not included.
- Only public properties and fields can be serialized. Properties must have public accessors (get and set methods). If you must serialize non-public data, use the **DataContractSerializer** class rather than XML serialization.
- A class must have a parameterless constructor to be serialized by **XmlSerializer**.
- Methods cannot be serialized.
- **XmlSerializer** can process classes that implement **IEnumerable** or **ICollection** differently if they meet certain requirements, as follows.

A class that implements **IEnumerable** must implement a public **Add** method that takes a single parameter. The **Add** method's parameter must be consistent (polymorphic) with the type returned from the **IEnumerator.Current** property returned from the **GetEnumerator** method.

A class that implements **ICollection** in addition to **IEnumerable** (such as **CollectionBase**) must have a public **Item** indexed property (an indexer in C#) that takes an integer and it must have a public **Count** property of type **integer**. The parameter passed to the **Add** method must be the same type as that returned from the **Item** property, or one of that type's bases.

For classes that implement **ICollection**, values to be serialized are retrieved from the indexed **Item** property rather than by calling **GetEnumerator**. Also, public fields and properties are not serialized, with the exception of public fields that return another collection class (one that implements **ICollection**). For an example, see [Examples of XML Serialization](#).

XSD Data Type Mapping

The W3C document titled [XML Schema Part 2: Datatypes](#) specifies the simple data types that are allowed in an XML Schema definition language (XSD) schema. For many of these (for example, **int** and **decimal**), there is a corresponding data type in .NET. However, some XML data types do not have a corresponding .NET data type, for example, the **NMTOKEN** data type. In such cases, if you use the XML Schema Definition tool ([XML Schema Definition Tool \(Xsd.exe\)](#)) to generate classes from a schema, an appropriate attribute is applied to a member of type string, and its **DataType** property is set to the XML data type name. For example, if a schema contains an element named "MyToken" with the XML data type **NMTOKEN**, the generated class might contain a member as shown in the following example.

```
<XmlElement(DataType:="NMTOKEN")> _
Public MyToken As String
```

```
[XmlElement(DataType = "NMTOKEN")]
public string MyToken;
```

Similarly, if you are creating a class that must conform to a specific XML Schema (XSD), you should apply the appropriate attribute and set its **DataType** property to the desired XML data type name.

For a complete list of type mappings, see the [DataType](#) property for any of the following attribute classes:

- [SoapAttributeAttribute](#)
- [SoapElementAttribute](#)
- [XmlAttributeItemAttribute](#)
- [XmlAttributeAttribute](#)
- [XmlElementAttribute](#)
- [XmlRootAttribute](#)

See also

- [XmlSerializer](#)
- [DataContractSerializer](#)
- [FileStream](#)
- [XML and SOAP Serialization](#)
- [Binary Serialization](#)
- [Serialization](#)
- [XmlSerializer](#)
- [Examples of XML Serialization](#)
- [How to: Serialize an Object](#)
- [How to: Deserialize an Object](#)

Examples of XML Serialization

9/20/2022 • 15 minutes to read • [Edit Online](#)

XML serialization can take more than one form, from simple to complex. For example, you can serialize a class that simply consists of public fields and properties, as shown in [Introducing XML Serialization](#). The following code examples address various advanced scenarios, including how to use XML serialization to generate an XML stream that conforms to a specific XML Schema (XSD) document.

Serializing a DataSet

Besides serializing an instance of a public class, an instance of a [DataSet](#) can also be serialized, as shown in the following code example.

```
Private Sub SerializeDataSet(filename As String)
    Dim ser As XmlSerializer = new XmlSerializer(GetType(DataSet))
    ' Creates a DataSet; adds a table, column, and ten rows.
    Dim ds As DataSet = new DataSet("myDataSet")
    Dim t As DataTable = new DataTable("table1")
    Dim c As DataColumn = new DataColumn("thing")
    t.Columns.Add(c)
    ds.Tables.Add(t)
    Dim r As DataRow
    Dim i As Integer
    for i = 0 to 10
        r = t.NewRow()
        r(0) = "Thing " & i
        t.Rows.Add(r)
    Next
    Dim writer As StreamWriter = new StreamWriter(filename)
    ser.Serialize(writer, ds)
    writer.Close()
End Sub
```

```
private void SerializeDataSet(string filename){
    XmlSerializer ser = new XmlSerializer(typeof(DataSet));

    // Creates a DataSet; adds a table, column, and ten rows.
    DataSet ds = new DataSet("myDataSet");
    DataTable t = new DataTable("table1");
    DataColumn c = new DataColumn("thing");
    t.Columns.Add(c);
    ds.Tables.Add(t);
    DataRow r;
    for(int i = 0; i<10;i++){
        r = t.NewRow();
        r[0] = "Thing " + i;
        t.Rows.Add(r);
    }
    StreamWriter writer = new StreamWriter(filename);
    ser.Serialize(writer, ds);
    writer.Close();
}
```

Serializing an XElement and XmlNode

You can also serialize instances of an [XmlElement](#) or [XmlNode](#) class, as shown in the following code example.

```

private Sub SerializeElement(filename As String)
    Dim ser As XmlSerializer = new XmlSerializer(GetType(XmlElement))
    Dim myElement As XmlElement = _
        new XmlDocument().CreateElement("MyElement", "ns")
    myElement.InnerText = "Hello World"
    Dim writer As TextWriter = new StreamWriter(filename)
    ser.Serialize(writer, myElement)
    writer.Close()
End Sub

Private Sub SerializeNode(filename As String)
    Dim ser As XmlSerializer = _
        new XmlSerializer(GetType(XmlNode))
    Dim myNode As XmlNode = new XmlDocument()._
        CreateNode(XmlNodeType.Element, "MyNode", "ns")
    myNode.InnerText = "Hello Node"
    Dim writer As TextWriter = new StreamWriter(filename)
    ser.Serialize(writer, myNode)
    writer.Close()
End Sub

```

```

private void SerializeElement(string filename){
    XmlSerializer ser = new XmlSerializer(typeof(XmlElement));
    XmlElement myElement=
    new XmlDocument().CreateElement("MyElement", "ns");
    myElement.InnerText = "Hello World";
    TextWriter writer = new StreamWriter(filename);
    ser.Serialize(writer, myElement);
    writer.Close();
}

private void SerializeNode(string filename){
    XmlSerializer ser = new XmlSerializer(typeof(XmlNode));
    XmlNode myNode= new XmlDocument().
    CreateNode(XmlNodeType.Element, "MyNode", "ns");
    myNode.InnerText = "Hello Node";
    TextWriter writer = new StreamWriter(filename);
    ser.Serialize(writer, myNode);
    writer.Close();
}

```

Serializing a Class that Contains a Field Returning a Complex Object

If a property or field returns a complex object (such as an array or a class instance), the [XmlSerializer](#) converts it to an element nested within the main XML document. For example, the first class in the following code example returns an instance of the second class.

```

Public Class PurchaseOrder
    Public MyAddress As Address
End Class

Public Class Address
    Public FirstName As String
End Class

```

```
public class PurchaseOrder
{
    public Address MyAddress;
}
public class Address
{
    public string FirstName;
}
```

The serialized XML output might resemble the following.

```
<PurchaseOrder>
    <MyAddress>
        <FirstName>George</FirstName>
    </MyAddress>
</PurchaseOrder>
```

Serializing an Array of Objects

You can also serialize a field that returns an array of objects, as shown in the following code example.

```
Public Class PurchaseOrder
    public ItemsOrders () As Item
End Class

Public Class Item
    Public ItemID As String
    Public ItemPrice As decimal
End Class
```

```
public class PurchaseOrder
{
    public Item [] ItemsOrders;
}

public class Item
{
    public string ItemID;
    public decimal ItemPrice;
}
```

The serialized class instance might resemble the following, if two items are ordered.

```
<PurchaseOrder xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <ItemsOrders>
        <Item>
            <ItemID>aaa111</ItemID>
            <ItemPrice>34.22</ItemPrice>
        </Item>
        <Item>
            <ItemID>bbb222</ItemID>
            <ItemPrice>2.89</ItemPrice>
        </Item>
    </ItemsOrders>
</PurchaseOrder>
```

Serializing a Class that Implements the ICollection Interface

You can create your own collection classes by implementing the [ICollection](#) interface, and use the [XmlSerializer](#) to serialize instances of these classes. Note that when a class implements the [ICollection](#) interface, only the collection contained by the class is serialized. Any public properties or fields added to the class will not be serialized. The class must include an **Add** method and an **Item** property (C# indexer) to be serialized.

```
Imports System.Collections
Imports System.IO
Imports System.Xml.Serialization

Public Class Test
    Shared Sub Main()
        Dim t As Test= new Test()
        t.SerializeCollection("coll.xml")
    End Sub

    Private Sub SerializeCollection(filename As String)
        Dim Emps As Employees = new Employees()
        ' Note that only the collection is serialized -- not the
        ' CollectionName or any other public property of the class.
        Emps.CollectionName = "Employees"
        Dim John100 As Employee = new Employee("John", "100xxx")
        Emps.Add(John100)
        Dim x As XmlSerializer = new XmlSerializer(GetType(Employees))
        Dim writer As TextWriter = new StreamWriter(filename)
        x.Serialize(writer, Emps)
        writer.Close()
    End Sub
End Class

Public Class Employees
    Implements ICollection
    Public CollectionName As String
    Private empArray As ArrayList = new ArrayList()

    Public ReadOnly Default Overloads _
    Property Item(index As Integer) As Employee
        get
            return CType (empArray(index), Employee)
        End Get
    End Property

    Public Sub CopyTo(a As Array, index As Integer) _
    Implements ICollection.CopyTo
        empArray.CopyTo(a, index)
    End Sub

    Public ReadOnly Property Count () As integer Implements _
    ICollection.Count
        get
            Count = empArray.Count
        End Get
    End Property

    Public ReadOnly Property SyncRoot ()As Object _
    Implements ICollection.SyncRoot
        get
            return me
        End Get
    End Property

    Public ReadOnly Property IsSynchronized () As Boolean _
    Implements ICollection.IsSynchronized
        get
    End Get
End Class
```

```
    return false
End Get
End Property

Public Function GetEnumerator() As IEnumerator _
Implements IEnumerable.GetEnumerator

    return empArray.GetEnumerator()
End Function

Public Function Add(newEmployee As Employee) As Integer
    empArray.Add(newEmployee)
    return empArray.Count
End Function
End Class

Public Class Employee
    Public EmpName As String
    Public EmpID As String

    Public Sub New ()
    End Sub

    Public Sub New (newName As String , newID As String )
        EmpName = newName
        EmpID = newID
    End Sub
End Class
```

```

using System;
using System.Collections;
using System.IO;
using System.Xml.Serialization;

public class Test {
    static void Main(){
        Test t = new Test();
        t.SerializeCollection("coll.xml");
    }

    private void SerializeCollection(string filename){
        Employees Emps = new Employees();
        // Note that only the collection is serialized -- not the
        // CollectionName or any other public property of the class.
        Emps.CollectionName = "Employees";
        Employee John100 = new Employee("John", "100xxx");
        Emps.Add(John100);
        XmlSerializer x = new XmlSerializer(typeof(Employees));
        TextWriter writer = new StreamWriter(filename);
        x.Serialize(writer, Emps);
    }
}

public class Employees:ICollection {
    public string CollectionName;
    private ArrayList empArray = new ArrayList();

    public Employee this[int index]{
        get{return (Employee) empArray[index];}
    }

    public void CopyTo(Array a, int index){
        empArray.CopyTo(a, index);
    }
    public int Count{
        get{return empArray.Count;}
    }
    public object SyncRoot{
        get{return this;}
    }
    public bool IsSynchronized{
        get{return false;}
    }
    public IEnumator GetEnumator(){
        return empArray.GetEnumerator();
    }

    public void Add(Employee newEmployee){
        empArray.Add(newEmployee);
    }
}

public class Employee {
    public string EmpName;
    public string EmpID;
    public Employee(){}
    public Employee(string empName, string empID){
        EmpName = empName;
        EmpID = empID;
    }
}

```

Purchase Order Example

You can cut and paste the following example code into a text file renamed with a .cs or .vb file name extension.

Use the C# or Visual Basic compiler to compile the file. Then run it using the name of the executable.

This example uses a simple scenario to demonstrate how an instance of an object is created and serialized into a file stream using the [Serialize](#) method. The XML stream is saved to a file, and the same file is then read back and reconstructed into a copy of the original object using the [Deserialize](#) method.

In this example, a class named `PurchaseOrder` is serialized and then deserialized. A second class named `Address` is also included because the public field named `ShipTo` must be set to an `Address`. Similarly, an `OrderedItem` class is included because an array of `OrderedItem` objects must be set to the `OrderedItems` field. Finally, a class named `Test` contains the code that serializes and deserializes the classes.

The `CreatePO` method creates the `PurchaseOrder`, `Address`, and `OrderedItem` class objects, and sets the public field values. The method also constructs an instance of the [XmlSerializer](#) class that is used to serialize and deserialize the `PurchaseOrder`. Note that the code passes the type of the class that will be serialized to the constructor. The code also creates a `FileStream` that is used to write the XML stream to an XML document.

The `ReadPo` method is a little simpler. It just creates objects to deserialize and reads out their values. As with the `CreatePo` method, you must first construct an [XmlSerializer](#), passing the type of the class to be deserialized to the constructor. Also, a `FileStream` is required to read the XML document. To deserialize the objects, call the [Deserialize](#) method with the `FileStream` as an argument. The deserialized object must be cast to an object variable of type `PurchaseOrder`. The code then reads the values of the deserialized `PurchaseOrder`. Note that you can also read the `PO.xml` file that is created to see the actual XML output.

```
Imports System.IO
Imports System.Xml
Imports System.Xml.Serialization
Imports Microsoft.VisualBasic

' The XmlRoot attribute allows you to set an alternate name
' (PurchaseOrder) for the XML element and its namespace. By
' default, the XmlSerializer uses the class name. The attribute
' also allows you to set the XML namespace for the element. Lastly,
' the attribute sets the IsNullable property, which specifies whether
' the xsi:null attribute appears if the class instance is set to
' a null reference.
<XmlRoot("PurchaseOrder", _
Namespace := "http://www.cpandl.com", IsNullable := False)> _
Public Class PurchaseOrder
    Public ShipTo As Address
    Public OrderDate As String
    ' The XmlArrayAttribute changes the XML element name
    ' from the default of "OrderedItems" to "Items".
    <XmlArray("Items")> _
    Public OrderedItems() As OrderedItem
    Public SubTotal As Decimal
    Public ShipCost As Decimal
    Public TotalCost As Decimal
End Class

Public Class Address
    ' The XmlAttribute attribute instructs the XmlSerializer to serialize the
    ' Name field as an XML attribute instead of an XML element (XML element is
    ' the default behavior).
    <XmlAttribute()> _
    Public Name As String
    Public Line1 As String

    ' Setting the IsNullable property to false instructs the
    ' XmlSerializer that the XML attribute will not appear if
    ' the City field is set to a null reference.
    <XmlElement(IsNullable := False)> _
    Public City As String
    Public State As String

```

```

    Public Zip As String
End Class

Public Class OrderedItem
    Public ItemName As String
    Public Description As String
    Public UnitPrice As Decimal
    Public Quantity As Integer
    Public LineTotal As Decimal

    ' Calculate is a custom method that calculates the price per item
    ' and stores the value in a field.
    Public Sub Calculate()
        LineTotal = UnitPrice * Quantity
    End Sub
End Class

Public Class Test
    Public Shared Sub Main()
        ' Read and write purchase orders.
        Dim t As New Test()
        t.CreatePO("po.xml")
        t.ReadPO("po.xml")
    End Sub

    Private Sub CreatePO(filename As String)
        ' Creates an instance of the XmlSerializer class;
        ' specifies the type of object to serialize.
        Dim serializer As New XmlSerializer(GetType(PurchaseOrder))
        Dim writer As New StreamWriter(filename)
        Dim po As New PurchaseOrder()

        ' Creates an address to ship and bill to.
        Dim billAddress As New Address()
        billAddress.Name = "Teresa Atkinson"
        billAddress.Line1 = "1 Main St."
        billAddress.City = "AnyTown"
        billAddress.State = "WA"
        billAddress.Zip = "00000"
        ' Set ShipTo and BillTo to the same addressee.
        po.ShipTo = billAddress
        po.OrderDate = System.DateTime.Now.ToString("yyyy-MM-dd")

        ' Creates an OrderedItem.
        Dim i1 As New OrderedItem()
        i1.ItemName = "Widget S"
        i1.Description = "Small widget"
        i1.UnitPrice = CDec(5.23)
        i1.Quantity = 3
        i1.Calculate()

        ' Inserts the item into the array.
        Dim items(0) As OrderedItem
        items(0) = i1
        po.OrderedItems = items
        ' Calculates the total cost.
        Dim subTotal As New Decimal()
        Dim oi As OrderedItem
        For Each oi In items
            subTotal += oi.LineTotal
        Next oi
        po.SubTotal = subTotal
        po.ShipCost = CDec(12.51)
        po.TotalCost = po.SubTotal + po.ShipCost
        ' Serializes the purchase order, and close the TextWriter.
        serializer.Serialize(writer, po)
        writer.Close()
    End Sub

```

```

Protected Sub ReadPO(filename As String)
    ' Creates an instance of the XmlSerializer class;
    ' specifies the type of object to be deserialized.
    Dim serializer As New XmlSerializer(GetType(PurchaseOrder))
    ' If the XML document has been altered with unknown
    ' nodes or attributes, handles them with the
    ' UnknownNode and UnknownAttribute events.
    AddHandler serializer.UnknownNode, AddressOf serializer_UnknownNode
    AddHandler serializer.UnknownAttribute, AddressOf _
        serializer_UnknownAttribute

    ' A FileStream is needed to read the XML document.
    Dim fs As New FileStream(filename, FileMode.Open)
    ' Declare an object variable of the type to be deserialized.
    Dim po As PurchaseOrder
    ' Uses the Deserialize method to restore the object's state
    ' with data from the XML document.
    po = CType(serializer.Deserialize(fs), PurchaseOrder)
    ' Reads the order date.
    Console.WriteLine("OrderDate: " & po.OrderDate)

    ' Reads the shipping address.
    Dim shipTo As Address = po.ShipTo
    ReadAddress(shipTo, "Ship To:")
    ' Reads the list of ordered items.
    Dim items As OrderedItem() = po.OrderedItems
    Console.WriteLine("Items to be shipped:")
    Dim oi As OrderedItem
    For Each oi In items
        Console.WriteLine((ControlChars.Tab & oi.ItemName & _
            ControlChars.Tab & _
            oi.Description & ControlChars.Tab & oi.UnitPrice & _
            ControlChars.Tab & _
            oi.Quantity & ControlChars.Tab & oi.LineTotal))
    Next oi
    ' Reads the subtotal, shipping cost, and total cost.
    Console.WriteLine((ControlChars.Cr & New String _ 
        (ControlChars.Tab, 5) & _
        " Subtotal" & ControlChars.Tab & po.SubTotal & ControlChars.Cr & _
        New String(ControlChars.Tab, 5) & " Shipping" & ControlChars.Tab & _
        po.ShipCost & ControlChars.Cr & New String(ControlChars.Tab, 5) & _
        " Total" & New String(ControlChars.Tab, 2) & po.TotalCost))
End Sub

Protected Sub ReadAddress(a As Address, label As String)
    ' Reads the fields of the Address.
    Console.WriteLine(label)
    Console.WriteLine((ControlChars.Tab & a.Name & ControlChars.Cr & _
        ControlChars.Tab & a.Line1 & ControlChars.Cr & ControlChars.Tab & _
        a.City & ControlChars.Tab & a.State & ControlChars.Cr & _
        ControlChars.Tab & a.Zip & ControlChars.Cr))
End Sub

Protected Sub serializer_UnknownNode(sender As Object, e As _
    XmlNodeEventArgs)
    Console.WriteLine(("Unknown Node:" & e.Name & _
        ControlChars.Tab & e.Text))
End Sub

Protected Sub serializer_UnknownAttribute(sender As Object, _
    e As XmlAttributeEventArgs)
    Dim attr As System.Xml.XmlAttribute = e.Attr
    Console.WriteLine(("Unknown attribute " & attr.Name & "=" & _
        attr.Value & ""))
End Sub 'serializer_UnknownAttribute
End Class 'Test

```

```

using System;
using System.IO;
using System.Xml;
using System.Xml.Serialization;

// The XmlRoot attribute allows you to set an alternate name
// (PurchaseOrder) for the XML element and its namespace. By
// default, the XmlSerializer uses the class name. The attribute
// also allows you to set the XML namespace for the element. Lastly,
// the attribute sets the IsNullable property, which specifies whether
// the xsi:null attribute appears if the class instance is set to
// a null reference.
[XmlRoot("PurchaseOrder", Namespace="http://www.cpandl.com",
IsNullable = false)]
public class PurchaseOrder
{
    public Address ShipTo;
    public string OrderDate;
    // The XmlArray attribute changes the XML element name
    // from the default of "OrderedItems" to "Items".
    [XmlArray("Items")]
    public OrderedItem[] OrderedItems;
    public decimal SubTotal;
    public decimal ShipCost;
    public decimal TotalCost;
}

public class Address
{
    // The XmlAttribute attribute instructs the XmlSerializer to serialize the
    // Name field as an XML attribute instead of an XML element (XML element is
    // the default behavior).
    [XmlAttribute]
    public string Name;
    public string Line1;

    // Setting the IsNullable property to false instructs the
    // XmlSerializer that the XML attribute will not appear if
    // the City field is set to a null reference.
    [XmlElement(IsNullable = false)]
    public string City;
    public string State;
    public string Zip;
}

public class OrderedItem
{
    public string ItemName;
    public string Description;
    public decimal UnitPrice;
    public int Quantity;
    public decimal LineTotal;

    // Calculate is a custom method that calculates the price per item
    // and stores the value in a field.
    public void Calculate()
    {
        LineTotal = UnitPrice * Quantity;
    }
}

public class Test
{
    public static void Main()
    {
        // Read and write purchase orders.
        Test t = new Test();
        t.CreatePO("po.xml");
        t.ReadPO("po.xml");
    }
}

```

```

}

private void CreatePO(string filename)
{
    // Creates an instance of the XmlSerializer class;
    // specifies the type of object to serialize.
    XmlSerializer serializer =
    new XmlSerializer(typeof(PurchaseOrder));
    TextWriter writer = new StreamWriter(filename);
    PurchaseOrder po=new PurchaseOrder();

    // Creates an address to ship and bill to.
    Address billAddress = new Address();
    billAddress.Name = "Teresa Atkinson";
    billAddress.Line1 = "1 Main St.";
    billAddress.City = "AnyTown";
    billAddress.State = "WA";
    billAddress.Zip = "00000";
    // Sets ShipTo and BillTo to the same addressee.
    po.ShipTo = billAddress;
    po.OrderDate = System.DateTime.Now.ToString("yyyy-MM-dd");

    // Creates an OrderedItem.
    OrderedItem i1 = new OrderedItem();
    i1.ItemName = "Widget S";
    i1.Description = "Small widget";
    i1.UnitPrice = (decimal) 5.23;
    i1.Quantity = 3;
    i1.Calculate();

    // Inserts the item into the array.
    OrderedItem [] items = {i1};
    po.OrderedItems = items;
    // Calculate the total cost.
    decimal subTotal = new decimal();
    foreach(OrderedItem oi in items)
    {
        subTotal += oi.LineTotal;
    }
    po.SubTotal = subTotal;
    po.ShipCost = (decimal) 12.51;
    po.TotalCost = po.SubTotal + po.ShipCost;
    // Serializes the purchase order, and closes the TextWriter.
    serializer.Serialize(writer, po);
    writer.Close();
}

protected void ReadPO(string filename)
{
    // Creates an instance of the XmlSerializer class;
    // specifies the type of object to be deserialized.
    XmlSerializer serializer = new XmlSerializer(typeof(PurchaseOrder));
    // If the XML document has been altered with unknown
    // nodes or attributes, handles them with the
    // UnknownNode and UnknownAttribute events.
    serializer.UnknownNode+= new
    XmlNodeEventHandler(serializer_UncnownNode);
    serializer.UnknownAttribute+= new
    XmlAttributeEventHandler(serializer_UncnownAttribute);

    // A FileStream is needed to read the XML document.
    FileStream fs = new FileStream(filename, FileMode.Open);
    // Declares an object variable of the type to be deserialized.
    PurchaseOrder po;
    // Uses the Deserialize method to restore the object's state
    // with data from the XML document. */
    po = (PurchaseOrder) serializer.Deserialize(fs);
    // Reads the order date.
    Console.WriteLine ("OrderDate: " + po.OrderDate);
}

```

```

    Console.WriteLine("\tOrder Date: " + po.OrderDate);

    // Reads the shipping address.
    Address shipTo = po.ShipTo;
    ReadAddress(shipTo, "Ship To:");
    // Reads the list of ordered items.
    OrderedItem [] items = po.OrderedItems;
    Console.WriteLine("Items to be shipped:");
    foreach(OrderedItem oi in items)
    {
        Console.WriteLine("\t" +
            oi.ItemName + "\t" +
            oi.Description + "\t" +
            oi.UnitPrice + "\t" +
            oi.Quantity + "\t" +
            oi.LineTotal);
    }
    // Reads the subtotal, shipping cost, and total cost.
    Console.WriteLine(
        "\n\t\t\tSubtotal\t" + po.SubTotal +
        "\n\t\t\tShipping\t" + po.ShipCost +
        "\n\t\t\tTotal\t" + po.TotalCost
    );
}

protected void ReadAddress(Address a, string label)
{
    // Reads the fields of the Address.
    Console.WriteLine(label);
    Console.Write("\t" +
        a.Name + "\n\t" +
        a.Line1 + "\n\t" +
        a.City + "\t" +
        a.State + "\n\t" +
        a.Zip + "\n");
}

protected void serializer_UnknownNode
(object sender, XmlNodeEventArgs e)
{
    Console.WriteLine("Unknown Node:" + e.Name + "\t" + e.Text);
}

protected void serializer_UnknownAttribute
(object sender, XmlAttributeEventArgs e)
{
    System.Xml.XmlAttribute attr = e.Attr;
    Console.WriteLine("Unknown attribute " +
        attr.Name + "=" + attr.Value + "'");
}
}

```

The XML output might resemble the following.

```
<?xml version="1.0" encoding="utf-8"?>
<PurchaseOrder xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://www.cpandl.com">
    <ShipTo Name="Teresa Atkinson">
        <Line1>1 Main St.</Line1>
        <City>AnyTown</City>
        <State>WA</State>
        <Zip>00000</Zip>
    </ShipTo>
    <OrderDate>Wednesday, June 27, 2001</OrderDate>
    <Items>
        <OrderedItem>
            <ItemName>Widget S</ItemName>
            <Description>Small widget</Description>
            <UnitPrice>5.23</UnitPrice>
            <Quantity>3</Quantity>
            <LineTotal>15.69</LineTotal>
        </OrderedItem>
    </Items>
    <SubTotal>15.69</SubTotal>
    <ShipCost>12.51</ShipCost>
    <TotalCost>28.2</TotalCost>
</PurchaseOrder>
```

See also

- [Introducing XML Serialization](#)
- [Controlling XML Serialization Using Attributes](#)
- [Attributes That Control XML Serialization](#)
- [XmlSerializer Class](#)
- [How to: Serialize an Object](#)
- [How to: Deserialize an Object](#)

The XML Schema Definition Tool and XML Serialization

9/20/2022 • 2 minutes to read • [Edit Online](#)

The XML Schema Definition tool ([XML Schema Definition Tool \(Xsd.exe\)](#)) is installed along with the .NET Framework tools as part of the Windows® Software Development Kit (SDK). The tool is designed primarily for two purposes:

- To generate either C# or Visual Basic class files that conform to a specific XML Schema definition language (XSD) schema. The tool takes an XML Schema as an argument and outputs a file that contains a number of classes that, when serialized with the [XmlSerializer](#), conform to the schema. For information about how to use the tool to generate classes that conform to a specific schema, see [How to: Use the XML Schema Definition Tool to Generate Classes and XML Schema Documents](#).
- To generate an XML Schema document from a .dll file or .exe file. To see the schema of a set of files that you have either created or one that has been modified with attributes, pass the DLL or EXE as an argument to the tool to generate the XML schema. For information about how to use the tool to generate an XML Schema Document from a set of classes, see [How to: Use the XML Schema Definition Tool to Generate Classes and XML Schema Documents](#).

For more information about using the tool, see [XML Schema Definition Tool \(Xsd.exe\)](#).

See also

- [DataSet](#)
- [Introducing XML Serialization](#)
- [XML Schema Definition Tool \(Xsd.exe\)](#)
- [XmlSerializer](#)
- [How to: Serialize an Object](#)
- [How to: Deserialize an Object](#)
- [How to: Use the XML Schema Definition Tool to Generate Classes and XML Schema Documents](#)
- [XML Schema Binding Support](#)

Control XML serialization using attributes

9/20/2022 • 6 minutes to read • [Edit Online](#)

Attributes can be used to control the XML serialization of an object or to create an alternate XML stream from the same set of classes. For more details about creating an alternate XML stream, see [How to: Specify an Alternate Element Name for an XML Stream](#).

NOTE

If the XML generated must conform to section 5 of the World Wide Web Consortium (W3C) document titled [Simple Object Access Protocol \(SOAP\) 1.1](#), use the attributes listed in [Attributes That Control Encoded SOAP Serialization](#).

By default, an XML element name is determined by the class or member name. In a simple class named `Book`, a field named `ISBN` will produce an XML element tag `<ISBN>`, as shown in the following example.

```
Public Class Book
    Public ISBN As String
End Class
' When an instance of the Book class is serialized, it might
' produce this XML:
' <ISBN>1234567890</ISBN>.
```

```
public class Book
{
    public string ISBN;
}
// When an instance of the Book class is serialized, it might
// produce this XML:
// <ISBN>1234567890</ISBN>.
```

This default behavior can be changed if you want to give the element a new name. The following code shows how an attribute enables this by setting the `ElementName` property of a `XMLElementAttribute`.

```
Public Class TaxRates
    <XmlElement(ElementName = "TaxRate")> _
    Public ReturnTaxRate As Decimal
End Class
```

```
public class TaxRates {
    [XmlElement(ElementName = "TaxRate")]
    public decimal ReturnTaxRate;
}
```

For more information about attributes, see [Attributes](#). For a list of attributes that control XML serialization, see [Attributes That Control XML Serialization](#).

Controlling Array Serialization

The `XmlAttributeAttribute` and the `XmlAttributeItemAttribute` attributes are designed to control the serialization of arrays. Using these attributes, you can control the element name, namespace, and XML Schema (XSD) data type

(as defined in the World Wide Web Consortium [www.w3.org] document titled "XML Schema Part 2: Datatypes"). You can also specify the types that can be included in an array.

The [XmlAttributeAttribute](#) will determine the properties of the enclosing XML element that results when an array is serialized. For example, by default, serializing the array below will result in an XML element named `Employees`. The `Employees` element will contain a series of elements named after the array type `Employee`.

```
Public Class Group
    Public Employees() As Employee
End Class
Public Class Employee
    Public Name As String
End Class
```

```
public class Group {
    public Employee[] Employees;
}
public class Employee {
    public string Name;
}
```

A serialized instance might resemble the following.

```
<Group>
<Employees>
    <Employee>
        <Name>Haley</Name>
    </Employee>
</Employees>
</Group>
```

By applying a [XmlAttributeAttribute](#), you can change the name of the XML element, as follows.

```
Public Class Group
    <XmlAttribute("TeamMembers")> _
    Public Employees() As Employee
End Class
```

```
public class Group {
    [XmlAttribute("TeamMembers")]
    public Employee[] Employees;
}
```

The resulting XML might resemble the following.

```
<Group>
<TeamMembers>
    <Employee>
        <Name>Haley</Name>
    </Employee>
</TeamMembers>
</Group>
```

The [XmlAttributeItemAttribute](#), on the other hand, controls how the items contained in the array are serialized. Note that the attribute is applied to the field returning the array.

```

Public Class Group
    <XmlArrayItem("MemberName")> _
    Public Employee() As Employees
End Class

```

```

public class Group {
    [XmlArrayItem("MemberName")]
    public Employee[] Employees;
}

```

The resulting XML might resemble the following.

```

<Group>
    <Employees>
        <MemberName>Haley</MemberName>
    </Employees>
</Group>

```

Serializing Derived Classes

Another use of the [XmlAttributeAttribute](#) is to allow the serialization of derived classes. For example, another class named `Manager` that derives from `Employee` can be added to the previous example. If you do not apply the [XmlAttributeAttribute](#), the code will fail at run time because the derived class type will not be recognized. To remedy this, apply the attribute twice, each time setting the `Type` property for each acceptable type (base and derived).

```

Public Class Group
    <XmlAttribute(Type:=GetType(Employee)), _ 
     XmlAttribute(Type:=GetType(Manager))> _
    Public Employees() As Employee
End Class
Public Class Employee
    Public Name As String
End Class
Public Class Manager
    Inherits Employee
    Public Level As Integer
End Class

```

```

public class Group {
    [XmlAttribute(Type = typeof(Employee)),
     XmlAttribute(Type = typeof(Manager))]
    public Employee[] Employees;
}
public class Employee {
    public string Name;
}
public class Manager:Employee {
    public int Level;
}

```

A serialized instance might resemble the following.

```

<Group>
<Employees>
    <Employee>
        <Name>Haley</Name>
    </Employee>
    <Employee xsi:type = "Manager">
        <Name>Ann</Name>
        <Level>3</Level>
    </Employee>
</Employees>
</Group>

```

Serializing an Array as a Sequence of Elements

You can also serialize an array as a flat sequence of XML elements by applying a [XmlElementAttribute](#) to the field returning the array as follows.

```

Public Class Group
    <XmlElement> _
    Public Employees() As Employee
End Class

```

```

public class Group {
    [XmlElement]
    public Employee[] Employees;
}

```

A serialized instance might resemble the following.

```

<Group>
<Employees>
    <Name>Haley</Name>
</Employees>
<Employees>
    <Name>Noriko</Name>
</Employees>
<Employees>
    <Name>Marco</Name>
</Employees>
</Group>

```

Another way to differentiate the two XML streams is to use the XML Schema Definition tool to generate the XML Schema (XSD) document files from the compiled code. (For more details on using the tool, see [The XML Schema Definition Tool and XML Serialization](#).) When no attribute is applied to the field, the schema describes the element in the following manner.

```

<xss:element minOccurs="0" maxOccurs = "1" name="Employees" type="ArrayOfEmployee" />

```

When the [XmlElementAttribute](#) is applied to the field, the resulting schema describes the element as follows.

```

<xss:element minOccurs="0" maxOccurs="unbounded" name="Employees" type="Employee" />

```

Serializing an ArrayList

The [ArrayList](#) class can contain a collection of diverse objects. You can therefore use an [ArrayList](#) much as you use an array. Instead of creating a field that returns an array of typed objects, however, you can create a field that returns a single [ArrayList](#). However, as with arrays, you must inform the [XmlSerializer](#) of the types of objects the [ArrayList](#) contains. To accomplish this, assign multiple instances of the [XmlElementAttribute](#) to the field, as shown in the following example.

```
Public Class Group
    <XmlElement(Type:=GetType(Employee)), _
    XmlElement(Type:=GetType(Manager))> _
    Public Info As ArrayList
End Class
```

```
public class Group {
    [XmlElement(Type = typeof(Employee)),
    XmlElement(Type = typeof(Manager))]
    public ArrayList Info;
}
```

Controlling Serialization of Classes Using [XmlAttribute](#) and [XmlAttribute](#)

There are two attributes that can be applied to a class (and only a class): [XmlAttribute](#) and [XmlAttribute](#). These attributes are very similar. The [XmlAttribute](#) can be applied to only one class: the class that, when serialized, represents the XML document's opening and closing element—in other words, the root element. The [XmlAttribute](#), on the other hand, can be applied to any class, including the root class.

For example, in the previous examples, the `Group` class is the root class, and all its public fields and properties become the XML elements found in the XML document. Therefore, there can be only one root class. By applying the [XmlAttribute](#), you can control the XML stream generated by the [XmlSerializer](#). For example, you can change the element name and namespace.

The [XmlAttribute](#) allows you to control the schema of the generated XML. This capability is useful when you need to publish the schema through an XML Web service. The following example applies both the [XmlAttribute](#) and the [XmlAttribute](#) to the same class.

```
<XmlAttribute("NewGroupName"), _
XmlAttribute("NewTypeName")> _
Public Class Group
    Public Employees() As Employee
End Class
```

```
[XmlAttribute("NewGroupName")]
[XmlAttribute("NewTypeName")]
public class Group {
    public Employee[] Employees;
}
```

If this class is compiled, and the XML Schema Definition tool is used to generate its schema, you would find the following XML describing `Group`.

```
<xss:element name="NewGroupName" type="NewTypeName" />
```

In contrast, if you were to serialize an instance of the class, only `NewGroupName` would be found in the XML

document.

```
<NewGroupName>
    ...
</NewGroupName>
```

Preventing Serialization with the `XmlAttribute` Attribute

There might be situations when a public property or field does not need to be serialized. For example, a field or property could be used to contain metadata. In such cases, apply the [XmlAttribute](#) to the field or property and the [XmlSerializer](#) will skip over it.

See also

- [Attributes That Control XML Serialization](#)
- [Attributes That Control Encoded SOAP Serialization](#)
- [Introducing XML Serialization](#)
- [Examples of XML Serialization](#)
- [How to: Specify an Alternate Element Name for an XML Stream](#)
- [How to: Serialize an Object](#)
- [How to: Deserialize an Object](#)

Attributes That Control XML Serialization

9/20/2022 • 2 minutes to read • [Edit Online](#)

You can apply the attributes in the following table to classes and class members to control the way in which the `XmlSerializer` serializes or deserializes an instance of the class. To understand how these attributes control XML serialization, see [Controlling XML Serialization Using Attributes](#).

These attributes can also be used to control the literal style SOAP messages generated by an XML Web service. For more information about applying these attributes to an XML Web services method, see [XML Serialization with XML Web Services](#).

For more information about attributes, see [Attributes](#).

ATTRIBUTE	APPLIES TO	SPECIFIES
<code>XmlAnyAttributeAttribute</code>	Public field, property, parameter, or return value that returns an array of <code>XmlAttribute</code> objects.	When deserializing, the array will be filled with <code>XmlAttribute</code> objects that represent all XML attributes unknown to the schema.
<code>XmlAnyElementAttribute</code>	Public field, property, parameter, or return value that returns an array of <code>XmlElement</code> objects.	When deserializing, the array is filled with <code>XmlElement</code> objects that represent all XML elements unknown to the schema.
<code>XmlArrayAttribute</code>	Public field, property, parameter, or return value that returns an array of complex objects.	The members of the array will be generated as members of an XML array.
<code>XmlArrayItemAttribute</code>	Public field, property, parameter, or return value that returns an array of complex objects.	The derived types that can be inserted into an array. Usually applied in conjunction with an <code>XmlAttributeAttribute</code> .
<code>XmlAttributeAttribute</code>	Public field, property, parameter, or return value.	The member will be serialized as an XML attribute.
<code>XmlChoiceIdentifierAttribute</code>	Public field, property, parameter, or return value.	The member can be further disambiguated by using an enumeration.
<code>XmlElementAttribute</code>	Public field, property, parameter, or return value.	The field or property will be serialized as an XML element.
<code>XmlEnumAttribute</code>	Public field that is an enumeration identifier.	The element name of an enumeration member.
<code>XmlIgnoreAttribute</code>	Public properties and fields.	The property or field should be ignored when the containing class is serialized.

ATTRIBUTE	APPLIES TO	SPECIFIES
XmlAttributeAttribute	Public derived class declarations, and return values of public methods for Web Services Description Language (WSDL) documents.	The class should be included when generating schemas (to be recognized when serialized).
XmlRootAttribute	Public class declarations.	Controls XML serialization of the attribute target as an XML root element. Use the attribute to further specify the namespace and element name.
XmlTextAttribute	Public properties and fields.	The property or field should be serialized as XML text.
XmlAttributeAttribute	Public class declarations.	The name and namespace of the XML type.

In addition to these attributes, which are all found in the [System.Xml.Serialization](#) namespace, you can also apply the [DefaultValueAttribute](#) attribute to a field. The [DefaultValueAttribute](#) sets the value that will be automatically assigned to the member if no value is specified.

To control encoded SOAP XML serialization, see [Attributes That Control Encoded SOAP Serialization](#).

See also

- [XML and SOAP Serialization](#)
- [XmlSerializer](#)
- [Controlling XML Serialization Using Attributes](#)
- [How to: Specify an Alternate Element Name for an XML Stream](#)
- [How to: Serialize an Object](#)
- [How to: Deserialize an Object](#)

XML Serialization with XML Web Services

9/20/2022 • 5 minutes to read • [Edit Online](#)

XML serialization is the underlying transport mechanism used in the XML Web services architecture, performed by the [XmlSerializer](#) class. To control the XML generated by an XML Web service, you can apply the attributes listed in both [Attributes That Control XML Serialization](#) and [Attributes That Control Encoded SOAP Serialization](#) to the classes, return values, parameters, and fields of a file used to create an XML Web service (.asmx). For more information about creating an XML Web service, see [XML Web Services Using ASP.NET](#).

Literal and Encoded Styles

The XML generated by an XML Web service can be formatted in either one of two ways, either literal or encoded, as explained in [Customizing SOAP Message Formatting](#). Therefore there are two sets of attributes that control XML serialization. The attributes listed in [Attributes That Control XML Serialization](#) are designed to control literal style XML. The attributes listed in [Attributes That Control Encoded SOAP Serialization](#) control the encoded style. By selectively applying these attributes, you can tailor an application to return either, or both styles. Furthermore, these attributes can be applied (as appropriate) to return values and parameters.

Example of Using Both Styles

When you're creating an XML Web service, you can use both sets of attributes on the methods. In the following code example, the class named `MyService` contains two XML Web service methods, `MyLiteralMethod` and `MyEncodedMethod`. Both methods perform the same function: returning an instance of the `Order` class. In the `Order` class, the [XmlAttribute](#) and the [SoapTypeAttribute](#) attributes are both applied to the `OrderID` field, and both attributes have their `ElementName` property set to different values.

To run the example, paste the code into a file with an .asmx extension, and place the file into a virtual directory managed by Internet Information Services (IIS). From an HTML browser, such as Internet Explorer, type the name of the computer, virtual directory, and file.

```
<%@ WebService Language="VB" Class="MyService" %>
Imports System
Imports System.Web.Services
Imports System.Web.Services.Protocols
Imports System.Xml.Serialization
Public Class Order
    ' Both types of attributes can be applied. Depending on which type
    ' the method used, either one will affect the call.
    <SoapElement(ElementName:= "EncodedOrderID"), _
    XmlElement(ElementName:= "LiteralOrderID")> _
    public OrderID As String
End Class

Public Class MyService
    <WebMethod, SoapDocumentMethod> _
    public Function MyLiteralMethod() As Order
        Dim myOrder As Order = New Order()
        return myOrder
    End Function
    <WebMethod, SoapRpcMethod> _
    public Function MyEncodedMethod() As Order
        Dim myOrder As Order = New Order()
        return myOrder
    End Function
End Class
```

```

<%@ WebService Language="C#" Class="MyService" %>
using System;
using System.Web.Services;
using System.Web.Services.Protocols;
using System.Xml.Serialization;
public class Order {
    // Both types of attributes can be applied. Depending on which type
    // the method used, either one will affect the call.
    [SoapElement(ElementName = "EncodedOrderID")]
    [XmlElement(ElementName = "LiteralOrderID")]
    public String OrderID;
}
public class MyService {
    [WebMethod][SoapDocumentMethod]
    public Order MyLiteralMethod(){
        Order myOrder = new Order();
        return myOrder;
    }
    [WebMethod][SoapRpcMethod]
    public Order MyEncodedMethod(){
        Order myOrder = new Order();
        return myOrder;
    }
}

```

The following code example calls `MyLiteralMethod`. The element name is changed to "LiteralOrderID".

```

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Body>
        <MyLiteralMethodResponse xmlns="http://tempuri.org/">
            <MyLiteralMethodResult>
                <LiteralOrderID>string</LiteralOrderID>
            </MyLiteralMethodResult>
        </MyLiteralMethodResponse>
    </soap:Body>
</soap:Envelope>

```

The following code example calls `MyEncodedMethod`. The element name is "EncodedOrderID".

```

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:tns="http://tempuri.org/" xmlns:types="http://tempuri.org/encodedTypes"
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Body soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
        <tns:MyEncodedMethodResponse>
            <MyEncodedMethodResult href="#id1" />
        </tns:MyEncodedMethodResponse>
        <types:Order id="id1" xsi:type="types:Order">
            <EncodedOrderID xsi:type="xsd:string">string</EncodedOrderID>
        </types:Order>
    </soap:Body>
</soap:Envelope>

```

Applying Attributes to Return Values

You can also apply attributes to return values to control the namespace, element name, and so forth. The following code example applies the `XmlElementAttribute` attribute to the return value of the `MyLiteralMethod` method. Doing so allows you to control the namespace and element name.

```

<WebMethod, SoapDocumentMethod> _
public Function MyLiteralMethod() As _
<XmlElement(Namespace:="http://www.cohowinery.com", _
ElementName:= "BookOrder")> _
Order
    Dim myOrder As Order = New Order()
    return myOrder
End Function

```

```

[return: XmlElement(Namespace = "http://www.cohowinery.com",
ElementName = "BookOrder")]
[WebMethod][SoapDocumentMethod]
public Order MyLiteralMethod(){
    Order myOrder = new Order();
    return myOrder;
}

```

When invoked, the code returns XML that resembles the following.

```

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Body>
        <MyLiteralMethodResponse xmlns="http://tempuri.org/">
            <BookOrder xmlns="http://www.cohowinery.com">
                <LiteralOrderID>string</LiteralOrderID>
            </BookOrder>
        </MyLiteralMethodResponse>
    </soap:Body>
</soap:Envelope>

```

Attributes Applied to Parameters

You can also apply attributes to parameters to specify namespace, element name and so forth. The following code example adds a parameter to the `MyLiteralMethodResponse` method, and applies the `XmlAttributeAttribute` attribute to the parameter. The element name and namespace are both set for the parameter.

```

<WebMethod, SoapDocumentMethod> _
public Function MyLiteralMethod(<XmlElement _ 
("MyOrderID", Namespace:="http://www.microsoft.com")>ID As String) As _
<XmlElement(Namespace:="http://www.cohowinery.com", _
ElementName:= "BookOrder")> _
Order
    Dim myOrder As Order = New Order()
    myOrder.OrderID = ID
    return myOrder
End Function

```

```

[return: XmlElement(Namespace = "http://www.cohowinery.com",
ElementName = "BookOrder")]
[WebMethod][SoapDocumentMethod]
public Order MyLiteralMethod([XmlElement("MyOrderID",
Namespace="http://www.microsoft.com")] string ID){
    Order myOrder = new Order();
    myOrder.OrderID = ID;
    return myOrder;
}

```

The SOAP request would resemble the following.

```

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Body>
        <MyLiteralMethod xmlns="http://tempuri.org/">
            <MyOrderID xmlns="http://www.microsoft.com">string</MyOrderID>
        </MyLiteralMethod>
    </soap:Body>
</soap:Envelope>

```

Applying Attributes to Classes

If you need to control the namespace of elements that correlate to classes, you can apply `XmlAttribute`, `XmlRootAttribute`, and `SoapTypeAttribute`, as appropriate. The following code example applies all three to the `Order` class.

```

<XmlType("BigBookService"), _
SoapType("SoapBookService"), _
XmlRoot("BookOrderForm")> _
Public Class Order
    ' Both types of attributes can be applied. Depending on which
    ' the method used, either one will affect the call.
    <SoapElement(ElementName:= "EncodedOrderID"), _
    XmlElement(ElementName:= "LiteralOrderID")> _
    public OrderID As String
End Class

```

```

[XmlType("BigBooksService", Namespace = "http://www.cpandl.com")]
[SoapType("SoapBookService")]
[XmlRoot("BookOrderForm")]
public class Order {
    // Both types of attributes can be applied. Depending on which
    // the method used, either one will affect the call.
    [SoapElement(ElementName = "EncodedOrderID")]
    [XmlElement(ElementName = "LiteralOrderID")]
    public String OrderID;
}

```

The results of applying the `XmlAttribute` and `SoapTypeAttribute` can be seen when you examine the service description, as shown in the following code example.

```

<s:element name="BookOrderForm" type="s0:BigBookService" />
<s:complexType name="BigBookService">
    <s:sequence>
        <s:element minOccurs="0" maxOccurs="1" name="LiteralOrderID" type="s:string" />
    </s:sequence>

    <s:schema targetNamespace="http://tempuri.org/encodedTypes">
        <s:complexType name="SoapBookService">
            <s:sequence>
                <s:element minOccurs="1" maxOccurs="1" name="EncodedOrderID" type="s:string" />
            </s:sequence>
        </s:complexType>
    </s:schema>
</s:complexType>

```

The effect of the `XmlRootAttribute` can also be seen in the HTTP GET and HTTP POST results, as follows.

```
<?xml version="1.0" encoding="utf-8"?>
<BookOrderForm xmlns="http://tempuri.org/">
    <LiteralOrderID>string</LiteralOrderID>
</BookOrderForm>
```

See also

- [XML and SOAP Serialization](#)
- [Attributes That Control Encoded SOAP Serialization](#)
- [How to: Serialize an Object as a SOAP-Encoded XML Stream](#)
- [How to: Override Encoded SOAP XML Serialization](#)
- [Introducing XML Serialization](#)
- [How to: Serialize an Object](#)
- [How to: Deserialize an Object](#)

Attributes That Control Encoded SOAP Serialization

9/20/2022 • 2 minutes to read • [Edit Online](#)

The World Wide Web Consortium (W3C) document named [Simple Object Access Protocol \(SOAP\) 1.1](#) contains an optional section (section 5) that describes how SOAP parameters can be encoded. To conform to section 5 of the specification, you must use a special set of attributes found in the `System.Xml.Serialization` namespace. Apply those attributes as appropriate to classes and members of classes, and then use the `XmlSerializer` to serialize instances of the class or classes.

The following table shows the attributes, where they can be applied, and what they do. For more information about using these attributes to control XML serialization, see [How to: Serialize an Object as a SOAP-Encoded XML Stream](#) and [How to: Override Encoded SOAP XML Serialization](#).

For more information about attributes, see [Attributes](#).

ATTRIBUTE	APPLIES TO	SPECIFIES
<code>SoapAttributeAttribute</code>	Public field, property, parameter, or return value.	The class member will be serialized as an XML attribute.
<code>SoapElementAttribute</code>	Public field, property, parameter, or return value.	The class will be serialized as an XML element.
<code>SoapEnumAttribute</code>	Public field that is an enumeration identifier.	The element name of an enumeration member.
<code>SoapIgnoreAttribute</code>	Public properties and fields.	The property or field should be ignored when the containing class is serialized.
<code>SoapIncludeAttribute</code>	Public-derived class declarations and public methods for Web Services Description Language (WSDL) documents.	The type should be included when generating schemas (to be recognized when serialized).
<code>SoapTypeAttribute</code>	Public class declarations.	The class should be serialized as an XML type.

See also

- [XML and SOAP Serialization](#)
- [How to: Serialize an Object as a SOAP-Encoded XML Stream](#)
- [How to: Override Encoded SOAP XML Serialization](#)
- [Attributes](#)
- [XmlSerializer](#)
- [How to: Serialize an Object](#)
- [How to: Deserialize an Object](#)

How to: Serialize an Object

9/20/2022 • 2 minutes to read • [Edit Online](#)

To serialize an object, first create the object that is to be serialized and set its public properties and fields. To do this, you must determine the transport format in which the XML stream is to be stored, either as a stream or as a file. For example, if the XML stream must be saved in a permanent form, create a [FileStream](#) object.

NOTE

For more examples of XML serialization, see [Examples of XML Serialization](#).

To serialize an object

1. Create the object and set its public fields and properties.
2. Construct a [XmlSerializer](#) using the type of the object. For more information, see the [XmlSerializer](#) class constructors.
3. Call the [Serialize](#) method to generate either an XML stream or a file representation of the object's public properties and fields. The following example creates a file.

```
Dim myObject As MySerializableClass = New MySerializableClass()
' Insert code to set properties and fields of the object.
Dim mySerializer As XmlSerializer = New XmlSerializer(GetType(MySerializableClass))
' To write to a file, create a StreamWriter object.
Dim myWriter As StreamWriter = New StreamWriter("myFileName.xml")
mySerializer.Serialize(myWriter, myObject)
myWriter.Close()
```

```
MySerializableClass myObject = new MySerializableClass();
// Insert code to set properties and fields of the object.
XmlSerializer mySerializer = new
XmlSerializer(typeof(MySerializableClass));
// To write to a file, create a StreamWriter object.
StreamWriter myWriter = new StreamWriter("myFileName.xml");
mySerializer.Serialize(myWriter, myObject);
myWriter.Close();
```

See also

- [Introducing XML Serialization](#)
- [How to: Deserialize an Object](#)

How to deserialize an object using XmlSerializer

9/20/2022 • 2 minutes to read • [Edit Online](#)

When you deserialize an object, the transport format determines whether you will create a stream or file object. After the transport format is determined, you can call the [Serialize](#) or [Deserialize](#) methods, as required.

To deserialize an object

1. Construct a [XmlSerializer](#) using the type of the object to deserialize.
2. Call the [Deserialize](#) method to produce a replica of the object. When deserializing, you must cast the returned object to the type of the original, as shown in the following example, which deserializes the object from a file (although it could also be deserialized from a stream).

```
' Construct an instance of the XmlSerializer with the type
' of object that is being serialized.
Dim mySerializer As New XmlSerializer(GetType(MySerializableClass))
' To read the file, create a FileStream.
Using myFileStream As New FileStream("myFileName.xml", FileMode.Open)
    ' Call the Deserialize method and cast to the object type.
    Dim myObject = CType(
        mySerializer.Deserialize(myFileStream), MySerializableClass)
End Using
```

```
// Construct an instance of the XmlSerializer with the type
// of object that is being serialized.
var mySerializer = new XmlSerializer(typeof(MySerializableClass));
// To read the file, create a FileStream.
using var myFileStream = new FileStream("myFileName.xml", FileMode.Open);
// Call the Deserialize method and cast to the object type.
var myObject = (MySerializableClass)mySerializer.Deserialize(myFileStream);
```

See also

- [Introducing XML Serialization](#)
- [How to: Serialize an Object](#)

How to: Use the XML Schema Definition Tool to Generate Classes and XML Schema Documents

9/20/2022 • 2 minutes to read • [Edit Online](#)

The XML Schema Definition tool (Xsd.exe) allows you to generate an XML schema that describes a class or to generate the class defined by an XML schema. The following procedures show how to perform these operations.

The XML Schema Definition tool (Xsd.exe) usually can be found in the following path:

C:\Program Files (x86)\Microsoft SDKs\Windows\{version}\bin\NETFX {version} Tools

To generate classes that conform to a specific schema

1. Open a command prompt.
2. Pass the XML Schema as an argument to the XML Schema Definition tool, which creates a set of classes that are precisely matched to the XML Schema, for example:

```
xsd mySchema.xsd
```

The tool can only process schemas that reference the World Wide Web Consortium XML specification of March 16, 2001. In other words, the XML Schema namespace must be "http://www.w3.org/2001/XMLSchema" as shown in the following example.

```
<?xml version="1.0" encoding="utf-8"?>
<xss:schema attributeFormDefault="qualified" elementFormDefault="qualified" targetNamespace="" xmlns:xss="http://www.w3.org/2001/XMLSchema" />
```

3. Modify the classes with methods, properties, or fields, as necessary. For more information about modifying a class with attributes, see [Controlling XML Serialization Using Attributes](#) and [Attributes That Control Encoded SOAP Serialization](#).

It is often useful to examine the schema of the XML stream that is generated when instances of a class (or classes) are serialized. For example, you might publish your schema for others to use, or you might compare it to a schema with which you are trying to achieve conformity.

To generate an XML Schema document from a set of classes

1. Compile the class or classes into a DLL.
2. Open a command prompt.
3. Pass the DLL as an argument to Xsd.exe, for example:

```
xsd MyFile.dll
```

The schema (or schemas) will be written, beginning with the name "schema0.xsd".

See also

- [DataSet](#)
- [The XML Schema Definition Tool and XML Serialization](#)
- [Introducing XML Serialization](#)

- [XML Schema Definition Tool \(Xsd.exe\)](#)
- [XmlSerializer](#)
- [How to: Serialize an Object](#)
- [How to: Deserialize an Object](#)

How to: Control Serialization of Derived Classes

9/20/2022 • 4 minutes to read • [Edit Online](#)

Using the **XmlElementAttribute** attribute to change the name of an XML element is not the only way to customize object serialization. You can also customize the XML stream by deriving from an existing class and instructing the **XmlSerializer** instance how to serialize the new class.

For example, given a **Book** class, you can derive from it and create an **ExpandedBook** class that has a few more properties. However, you must instruct the **XmlSerializer** to accept the derived type when serializing or deserializing. This can be done by creating a **XmlElementAttribute** instance and setting its **Type** property to the derived class type. Add the **XmlElementAttribute** to a **XmlAttribute** instance. Then add the **XmlAttribute**s to a **XmlAttributeOverrides** instance, specifying the type being overridden and the name of the member that accepts the derived class. This is shown in the following example.

Example

```
Public Class Orders
    public Books() As Book
End Class

Public Class Book
    public ISBN As String
End Class

Public Class ExpandedBook
    Inherits Book
    public NewEdition As Boolean
End Class

Public Class Run
    Shared Sub Main()
        Dim t As Run = New Run()
        t.SerializeObject("Book.xml")
        t.DeserializeObject("Book.xml")
    End Sub

    Public Sub SerializeObject(filename As String)
        ' Each overridden field, property, or type requires
        ' an XmlAttributes instance.
        Dim attrs As XmlAttributes = New XmlAttributes()

        ' Creates an XmlElementAttribute instance to override the
        ' field that returns Book objects. The overridden field
        ' returns Expanded objects instead.
        Dim attr As XmlElementAttribute = _
            New XmlElementAttribute()
        attr.ElementName = "NewBook"
        attr.Type = GetType(ExpandedBook)

        ' Adds the element to the collection of elements.
        attrs.XmlElements.Add(attr)

        ' Creates the XmlAttributeOverrides.
        Dim attrOverrides As XmlAttributeOverrides = _
            New XmlAttributeOverrides()

        ' Adds the type of the class that contains the overridden
        ' member, as well as the XmlAttributes instance to override it
        ' with, to the XmlAttributeOverrides instance.
    End Sub
End Class
```

```

attrOverrides.Add(GetType(Orders), "Books", attrs)

' Creates the XmlSerializer using the XmlAttributeOverrides.
Dim s As XmlSerializer = _
New XmlSerializer(GetType(Orders), attrOverrides)

' Writing the file requires a TextWriter instance.
Dim writer As TextWriter = New StreamWriter(filename)

' Creates the object to be serialized.
Dim myOrders As Orders = New Orders()

' Creates an object of the derived type.
Dim b As ExpandedBook = New ExpandedBook()
b.ISBN= "123456789"
b.NewEdition = True
myOrders.Books = New ExpandedBook(){b}

' Serializes the object.
s.Serialize(writer,myOrders)
writer.Close()
End Sub

Public Sub DeserializeObject(filename As String)
Dim attrOverrides As XmlAttributeOverrides = _
New XmlAttributeOverrides()
Dim attrs As XmlAttributes = New XmlAttributes()

' Creates an XmlElementAttribute to override the
' field that returns Book objects. The overridden field
' returns Expanded objects instead.
Dim attr As XmlElementAttribute = _
New XmlElementAttribute()
attr.ElementName = "NewBook"
attr.Type = GetType(ExpandedBook)

' Adds the XmlElementAttribute to the collection of objects.
attrs.XmlElements.Add(attr)

attrOverrides.Add(GetType(Orders), "Books", attrs)

' Creates the XmlSerializer using the XmlAttributeOverrides.
Dim s As XmlSerializer = _
New XmlSerializer(GetType(Orders), attrOverrides)

Dim fs As FileStream = New FileStream(filename, FileMode.Open)
Dim myOrders As Orders = CType( s.Deserialize(fs), Orders)
Console.WriteLine("ExpandedBook:")

' The difference between deserializing the overridden
' XML document and serializing it is this: To read the derived
' object values, you must declare an object of the derived type
' and cast the returned object to it.
Dim expanded As ExpandedBook
Dim b As Book
for each b in myOrders.Books
    expanded = CType(b, ExpandedBook)
    Console.WriteLine(expanded.ISBN)
    Console.WriteLine(expanded.NewEdition)
Next
End Sub
End Class

```

```

public class Orders
{
    public Book[] Books;
}

```

```

public class Book
{
    public string ISBN;
}

public class ExpandedBook:Book
{
    public bool NewEdition;
}

public class Run
{
    public void SerializeObject(string filename)
    {
        // Each overridden field, property, or type requires
        // an XmlAttributes instance.
        XmlAttributes attrs = new XmlAttributes();

        // Creates an XmlElementAttribute instance to override the
        // field that returns Book objects. The overridden field
        // returns Expanded objects instead.
        XmlElementAttribute attr = new XmlElementAttribute();
        attr.ElementName = "NewBook";
        attr.Type = typeof(ExpandedBook);

        // Adds the element to the collection of elements.
        attrs.XmlElements.Add(attr);

        // Creates the XmlAttributeOverrides instance.
        XmlAttributeOverrides attrOverrides = new XmlAttributeOverrides();

        // Adds the type of the class that contains the overridden
        // member, as well as the XmlAttributes instance to override it
        // with, to the XmlAttributeOverrides.
        attrOverrides.Add(typeof(Orders), "Books", attrs);

        // Creates the XmlSerializer using the XmlAttributeOverrides.
        XmlSerializer s =
            new XmlSerializer(typeof(Orders), attrOverrides);

        // Writing the file requires a TextWriter instance.
        TextWriter writer = new StreamWriter(filename);

        // Creates the object to be serialized.
        Orders myOrders = new Orders();

        // Creates an object of the derived type.
        ExpandedBook b = new ExpandedBook();
        b.ISBN= "123456789";
        b.NewEdition = true;
        myOrders.Books = new ExpandedBook[]{b};

        // Serializes the object.
        s.Serialize(writer,myOrders);
        writer.Close();
    }

    public void DeserializeObject(string filename)
    {
        XmlAttributeOverrides attrOverrides =
            new XmlAttributeOverrides();
        XmlAttributes attrs = new XmlAttributes();

        // Creates an XmlElementAttribute to override the
        // field that returns Book objects. The overridden field
        // returns Expanded objects instead.
        XmlElementAttribute attr = new XmlElementAttribute();
        attr.ElementName = "NewBook";
    }
}

```

```
attr.Type = typeof(ExpandedBook);

// Adds the XmlElementAttribute to the collection of objects.
attrs.XmlElements.Add(attr);

attrOverrides.Add(typeof(Orders), "Books", attrs);

// Creates the XmlSerializer using the XmlAttributeOverrides.
XmlSerializer s =
new XmlSerializer(typeof(Orders), attrOverrides);

FileStream fs = new FileStream(filename, FileMode.Open);
Orders myOrders = (Orders) s.Deserialize(fs);
Console.WriteLine("ExpandedBook:");

// The difference between deserializing the overridden
// XML document and serializing it is this: To read the derived
// object values, you must declare an object of the derived type
// and cast the returned object to it.
ExpandedBook expanded;
foreach(Book b in myOrders.Books)
{
    expanded = (ExpandedBook)b;
    Console.WriteLine(
        expanded.ISBN + "\n" +
        expanded.NewEdition);
}
}
```

See also

- [XmlSerializer](#)
- [XmlElementAttribute](#)
- [XmlAttributeOverrides](#)
- [XmlAttributeOverrides](#)
- [XML and SOAP Serialization](#)
- [How to: Serialize an Object](#)
- [How to: Specify an Alternate Element Name for an XML Stream](#)

How to: Specify an Alternate Element Name for an XML Stream

9/20/2022 • 2 minutes to read • [Edit Online](#)

Using the [XmlSerializer](#), you can generate more than one XML stream with the same set of classes. You might want to do this because two different XML Web services require the same basic information, with only slight differences. For example, imagine two XML Web services that process orders for books, and thus both require ISBN numbers. One service uses the tag <ISBN> while the second uses the tag <BookID>. You have a class named `Book` that contains a field named `ISBN`. When an instance of the `Book` class is serialized, it will, by default, use the member name (`ISBN`) as the tag element name. For the first XML Web service, this is as expected. But to send the XML stream to the second XML Web service, you must override the serialization so that the tag's element name is `BookID`.

To create an XML stream with an alternate element name

1. Create an instance of the [XmlElementAttribute](#) class.
2. Set the `ElementName` of the [XmlElementAttribute](#) to "BookID".
3. Create an instance of the [XmlAttributeOverrides](#) class.
4. Add the `XmlElementAttribute` object to the collection accessed through the [XmlElementOverrides](#) property of [XmlAttributeOverrides](#).
5. Create an instance of the [XmlAttributeOverrides](#) class.
6. Add the `XmlAttributeOverrides` to the [XmlAttributeOverrides](#), passing the type of the object to override and the name of the member being overridden.
7. Create an instance of the [XmlSerializer](#) class with `XmlAttributeOverrides`.
8. Create an instance of the `Book` class, and serialize or deserialize it.

Example

```
Public Function SerializeOverride()
    ' Creates an XmlElementAttribute with the alternate name.
    Dim myElementAttribute As XmlElementAttribute = _
        New XmlElementAttribute()
    myElementAttribute.ElementName = "BookID"
    Dim myAttributes As XmlAttributes = New XmlAttributes()
    myAttributes.XmlElements.Add(myElementAttribute)
    Dim myOverrides As XmlAttributeOverrides = New XmlAttributeOverrides()
    myOverrides.Add(typeof(Book), "ISBN", myAttributes)
    Dim mySerializer As XmlSerializer = _
        New XmlSerializer(GetType(Book), myOverrides)
    Dim b As Book = New Book()
    b.ISBN = "123456789"
    ' Creates a StreamWriter to write the XML stream to.
    Dim writer As StreamWriter = New StreamWriter("Book.xml")
    mySerializer.Serialize(writer, b);
End Class
```

```
public void SerializeOverride()
{
    // Creates an XmlElementAttribute with the alternate name.
    XmlElementAttribute myElementAttribute = new XmlElementAttribute();
    myElementAttribute.ElementName = "BookID";
    XmlAttributes myAttributes = new XmlAttributes();
    myAttributes.XmlElements.Add(myElementAttribute);
    XmlAttributeOverrides myOverrides = new XmlAttributeOverrides();
    myOverrides.Add(typeof(Book), "ISBN", myAttributes);
    XmlSerializer mySerializer =
        new XmlSerializer(typeof(Book), myOverrides);
    Book b = new Book();
    b.ISBN = "123456789";
    // Creates a StreamWriter to write the XML stream to.
    StreamWriter writer = new StreamWriter("Book.xml");
    mySerializer.Serialize(writer, b);
}
```

The XML stream might resemble the following.

```
<Book>
    <BookID>123456789</BookID>
</Book>
```

See also

- [XmlElementAttribute](#)
- [XmlAttribute](#)
- [XmlAttributeOverrides](#)
- [XML and SOAP Serialization](#)
- [XmlSerializer](#)
- [How to: Serialize an Object](#)
- [How to: Deserialize an Object](#)

How to qualify XML element and XML attribute names

9/20/2022 • 3 minutes to read • [Edit Online](#)

XML namespaces contained by instances of the `XmlSerializerNamespaces` class must conform to the World Wide Web Consortium (W3C) specification called [Namespaces in XML](#).

XML namespaces provide a method for qualifying the names of XML elements and XML attributes in XML documents. A qualified name consists of a prefix and a local name, separated by a colon. The prefix functions only as a placeholder; it is mapped to a URI that specifies a namespace. The combination of the universally managed URI namespace and the local name produces a name that is guaranteed to be universally unique.

By creating an instance of `XmlSerializerNamespaces` and adding the namespace pairs to the object, you can specify the prefixes used in an XML document.

To create qualified names in an XML document

1. Create an instance of the `XmlSerializerNamespaces` class.
2. Add all prefixes and namespace pairs to the `XmlSerializerNamespaces`.
3. Apply the appropriate `System.Xml.Serialization` attribute to each member or class that the `XmlSerializer` is to serialize into an XML document.

The available attributes are: `XmlAnyElementAttribute`, `XmlAttributeAttribute`, `XmlAttributeItemAttribute`, `XmlAttributeAttribute`, `XmlElementAttribute`, `XmlRootAttribute`, and `XmlTypeAttribute`.

4. Set the `Namespace` property of each attribute to one of the namespace values from the `XmlSerializerNamespaces`.
5. Pass the `XmlSerializerNamespaces` to the `Serialize` method of the `XmlSerializer`.

Example

The following example creates an `XmlSerializerNamespaces`, and adds two prefix and namespace pairs to the object. The code creates an `XmlSerializer` that is used to serialize an instance of the `Books` class. The code calls the `Serialize` method with the `XmlSerializerNamespaces`, allowing the XML to contain prefixed namespaces.

```

Imports System.IO
Imports System.Xml
Imports System.Xml.Serialization

Public Module Program

    Public Sub Main()
        SerializeObject("XmlNamespaces.xml")
    End Sub

    Public Sub SerializeObject(filename As String)
        Dim mySerializer As New XmlSerializer(GetType(Books))
        ' Writing a file requires a TextWriter.
        Dim myWriter As New StreamWriter(filename)

        ' Creates an XmlSerializerNamespaces and adds two
        ' prefix-namespace pairs.
        Dim myNamespaces As New XmlSerializerNamespaces()
        myNamespaces.Add("books", "http://www.cpandl.com")
        myNamespaces.Add("money", "http://www.cohowinery.com")

        ' Creates a Book.
        Dim myBook As New Book()
        myBook.TITLE = "A Book Title"
        Dim myPrice As New Price()
        myPrice.price = CDec(9.95)
        myPrice.currency = "US Dollar"
        myBook.PRICE = myPrice
        Dim myBooks As New Books()
        myBooks.Book = myBook
        mySerializer.Serialize(myWriter, myBooks, myNamespaces)
        myWriter.Close()
    End Sub
End Module

Public Class Books
    <XmlElement([Namespace] := "http://www.cohowinery.com")> _
    Public Book As Book
End Class

<XmlType([Namespace] := "http://www.cpandl.com")> _
Public Class Book
    <XmlElement([Namespace] := "http://www.cpandl.com")> _
    Public TITLE As String
    <XmlElement([Namespace] := "http://www.cohowinery.com")> _
    Public PRICE As Price
End Class

Public Class Price
    <XmlAttribute([Namespace] := "http://www.cpandl.com")> _
    Public currency As String
    <XmlElement([Namespace] := "http://www.cohowinery.com")> _
    Public price As Decimal
End Class

```

```

using System;
using System.IO;
using System.Xml;
using System.Xml.Serialization;

public class Program
{
    public static void Main()
    {
        SerializeObject("XmlNamespaces.xml");
    }

    public static void SerializeObject(string filename)
    {
        var mySerializer = new XmlSerializer(typeof(Books));
        // Writing a file requires a TextWriter.
        TextWriter myWriter = new StreamWriter(filename);

        // Creates an XmlSerializerNamespaces and adds two
        // prefix-namespace pairs.
        var myNamespaces = new XmlSerializerNamespaces();
        myNamespaces.Add("books", "http://www.cpandl.com");
        myNamespaces.Add("money", "http://www.cohowinery.com");

        // Creates a Book.
        var myBook = new Book();
        myBook.TITLE = "A Book Title";
        var myPrice = new Price();
        myPrice.price = (decimal) 9.95;
        myPrice.currency = "US Dollar";
        myBook.PRICE = myPrice;
        var myBooks = new Books();
        myBooks.Book = myBook;
        mySerializer.Serialize(myWriter, myBooks, myNamespaces);
        myWriter.Close();
    }
}

public class Books
{
    [XmlElement(Namespace = "http://www.cohowinery.com")]
    public Book Book;
}

[XmlAttribute(Namespace = "http://www.cpandl.com")]
public class Book
{
    [XmlElement(Namespace = "http://www.cpandl.com")]
    public string TITLE;
    [XmlElement(Namespace = "http://www.cohowinery.com")]
    public Price PRICE;
}

public class Price
{
    [XmlAttribute(Namespace = "http://www.cpandl.com")]
    public string currency;
    [XmlElement(Namespace = "http://www.cohowinery.com")]
    public decimal price;
}

```

See also

- [XmlSerializer](#)
- [The XML Schema Definition Tool and XML Serialization](#)

- [Introducing XML Serialization](#)
- [XmlSerializer Class](#)
- [Attributes That Control XML Serialization](#)
- [How to: Specify an Alternate Element Name for an XML Stream](#)
- [How to: Serialize an Object](#)
- [How to: Deserialize an Object](#)

How to: Serialize an Object as a SOAP-Encoded XML Stream

9/20/2022 • 2 minutes to read • [Edit Online](#)

Because a SOAP message is built using XML, the `XmlSerializer` class can be used to serialize classes and generate encoded SOAP messages. The resulting XML conforms to [section 5 of the World Wide Web Consortium document "Simple Object Access Protocol \(SOAP\) 1.1"](#). When you are creating an XML Web service that communicates through SOAP messages, you can customize the XML stream by applying a set of special SOAP attributes to classes and members of classes. For a list of attributes, see [Attributes That Control Encoded SOAP Serialization](#).

To serialize an object as a SOAP-encoded XML stream

1. Create the class using the [XML Schema Definition Tool \(Xsd.exe\)](#).
2. Apply one or more of the special attributes found in `System.Xml.Serialization`. See the list in "Attributes That Control Encoded SOAP Serialization."
3. Create an `XmlTypeMapping` by creating a new `SapReflectionImporter`, and invoking the `ImportTypeMapping` method with the type of the serialized class.

The following code example calls the `ImportTypeMapping` method of the `SapReflectionImporter` class to create an `XmlTypeMapping`.

```
' Serializes a class named Group as a SOAP message.  
Dim myTypeMapping As XmlTypeMapping =  
    New SapReflectionImporter().ImportTypeMapping(GetType(Group))
```

```
// Serializes a class named Group as a SOAP message.  
XmlTypeMapping myTypeMapping =  
    new SapReflectionImporter().ImportTypeMapping(typeof(Group));
```

4. Create an instance of the `XmlSerializer` class by passing the `XmlTypeMapping` to the `XmlSerializer(XmlTypeMapping)` constructor.

```
Dim mySerializer As XmlSerializer = New XmlSerializer(myTypeMapping)
```

```
XmlSerializer mySerializer = new XmlSerializer(myTypeMapping);
```

5. Call the `Serialize` or `Deserialize` method.

Example

```
' Serializes a class named Group as a SOAP message.  
Dim myTypeMapping As XmlTypeMapping =  
    New SapReflectionImporter().ImportTypeMapping(GetType(Group))  
Dim mySerializer As XmlSerializer = New XmlSerializer(myTypeMapping)
```

```
// Serializes a class named Group as a SOAP message.  
XmlTypeMapping myTypeMapping =  
    new SoapReflectionImporter().ImportTypeMapping(typeof(Group));  
XmlSerializer mySerializer = new XmlSerializer(myTypeMapping);
```

See also

- [XML and SOAP Serialization](#)
- [Attributes That Control Encoded SOAP Serialization](#)
- [XML Serialization with XML Web Services](#)
- [How to: Serialize an Object](#)
- [How to: Deserialize an Object](#)
- [How to: Override Encoded SOAP XML Serialization](#)

How to: Override Encoded SOAP XML Serialization

9/20/2022 • 4 minutes to read • [Edit Online](#)

The process for overriding XML serialization of objects as SOAP messages is similar to the process for overriding standard XML serialization. For information about overriding standard XML serialization, see [How to: Specify an Alternate Element Name for an XML Stream](#).

To override serialization of objects as SOAP messages

1. Create an instance of the [SoapAttributeOverrides](#) class.
2. Create a [SoapAttributes](#) for each class member that is being serialized.
3. Create an instance of one or more of the attributes that affect XML serialization, as appropriate, to the member being serialized. For more information, see "Attributes That Control Encoded SOAP Serialization".
4. Set the appropriate property of [SoapAttributes](#) to the attribute created in step 3.
5. Add [SoapAttributes](#) to [SoapAttributeOverrides](#).
6. Create an [XmlTypeMapping](#) using the [SoapAttributeOverrides](#). Use the [SoapReflectionImporter.ImportTypeMapping](#) method.
7. Create an [XmlSerializer](#) using [XmlTypeMapping](#).
8. Serialize or deserialize the object.

Example

The following code example serializes a file in two ways: first, without overriding the [XmlSerializer](#) class's behavior, and second, by overriding the behavior. The example contains a class named [Group](#) with several members. Various attributes, such as the [SoapElementAttribute](#), have been applied to class members. When the class is serialized with the [SerializeOriginal](#) method, the attributes control the SOAP message content. When the [SerializeOverride](#) method is called, the behavior of the [XmlSerializer](#) is overridden by creating various attributes and setting the properties of a [SoapAttributes](#) to those attributes (as appropriate).

```
using System;
using System.IO;
using System.Xml;
using System.Xml.Serialization;
using System.Xml.Schema;

public class Group
{
    [SoapAttribute(Namespace = "http://www.cpandl.com")]
    public string GroupName;

    [SoapAttribute(DataType = "base64Binary")]
    public Byte[] GroupNumber;

    [SoapAttribute(DataType = "date", AttributeName = "CreationDate")]
    public DateTime Today;
    [SoapElement(DataType = "nonNegativeInteger", ElementName = "PosInt")]
    public string PositiveInt;
    // This is ignored when serialized unless it is overridden.
```

```

[SoapIgnore]
public bool IgnoreThis;

public GroupType GroupType;

[SoapInclude(typeof(Car))]
public Vehicle myCar(string licNumber)
{
    Vehicle v;
    if(licNumber == "")
    {
        v = new Car();
        v.licenseNumber = "!!!!!!";
    }
    else
    {
        v = new Car();
        v.licenseNumber = licNumber;
    }
    return v;
}

public abstract class Vehicle
{
    public string licenseNumber;
    public DateTime makeDate;
}

public class Car: Vehicle
{
}

public enum GroupType
{
    // These enums can be overridden.
    small,
    large
}

public class Run
{
    public static void Main()
    {
        Run test = new Run();
        test.SerializeOriginal("SoapOriginal.xml");
        test.SerializeOverride("SoapOverrides.xml");
        test.DeserializeOriginal("SoapOriginal.xml");
        test.DeserializeOverride("SoapOverrides.xml");

    }
    public void SerializeOriginal(string filename)
    {
        // Creates an instance of the XmlSerializer class.
        XmlTypeMapping myMapping =
        (new SoapReflectionImporter().ImportTypeMapping(
        typeof(Group)));
        XmlSerializer mySerializer =
        new XmlSerializer(myMapping);

        // Writing the file requires a TextWriter.
        TextWriter writer = new StreamWriter(filename);

        // Creates an instance of the class that will be serialized.
        Group myGroup = new Group();

        // Sets the object properties.
        myGroup.GroupName = ".NET";
    }
}

```

```

Byte [] hexByte = new Byte[2]{Convert.ToByte(100),
Convert.ToByte(50)};
myGroup.GroupNumber = hexByte;

DateTime myDate = new DateTime(2002,5,2);
myGroup.Today = myDate;

myGroup.PositiveInt= "10000";
myGroup.IgnoreThis=true;
myGroup.GroupType= GroupType.small;
Car thisCar =(Car) myGroup.myCar("1234566");

// Prints the license number just to prove the car was created.
Console.WriteLine("License#: " + thisCar.licenseNumber + "\n");

// Serializes the class and closes the TextWriter.
mySerializer.Serialize(writer, myGroup);
writer.Close();
}

public void SerializeOverride(string filename)
{
    // Creates an instance of the XmlSerializer class
    // that overrides the serialization.
    XmlSerializer overRideSerializer = CreateOverRideSerializer();

    // Writing the file requires a TextWriter.
    TextWriter writer = new StreamWriter(filename);

    // Creates an instance of the class that will be serialized.
    Group myGroup = new Group();

    // Sets the object properties.
    myGroup.GroupName = ".NET";

    Byte [] hexByte = new Byte[2]{Convert.ToByte(100),
Convert.ToByte(50)};
    myGroup.GroupNumber = hexByte;

    DateTime myDate = new DateTime(2002,5,2);
    myGroup.Today = myDate;

    myGroup.PositiveInt= "10000";
    myGroup.IgnoreThis=true;
    myGroup.GroupType= GroupType.small;
    Car thisCar =(Car) myGroup.myCar("1234566");

    // Serializes the class and closes the TextWriter.
    overRideSerializer.Serialize(writer, myGroup);
    writer.Close();
}

public void DeserializeOriginal(string filename)
{
    // Creates an instance of the XmlSerializer class.
    XmlTypeMapping myMapping =
    (new SoapReflectionImporter().ImportTypeMapping(
    typeof(Group)));
    XmlSerializer mySerializer =
    new XmlSerializer(myMapping);

    TextReader reader = new StreamReader(filename);

    // Deserializes and casts the object.
    Group myGroup;
    myGroup = (Group) mySerializer.Deserialize(reader);

    Console.WriteLine(myGroup.GroupName);
    Console.WriteLine(myGroup.GroupNumber[0]);
}

```

```

        Console.WriteLine(myGroup.GroupNumber[0]);
        Console.WriteLine(myGroup.GroupNumber[1]);
        Console.WriteLine(myGroup.Today);
        Console.WriteLine(myGroup.PositiveInt);
        Console.WriteLine(myGroup.IgnoreThis);
        Console.WriteLine();
    }

    public void DeserializeOverride(string filename)
    {
        // Creates an instance of the XmlSerializer class.
        XmlSerializer overRideSerializer = CreateOverrideSerializer();
        // Reading the file requires a TextReader.
        TextReader reader = new StreamReader(filename);

        // Deserializes and casts the object.
        Group myGroup;
        myGroup = (Group) overRideSerializer.Deserialize(reader);

        Console.WriteLine(myGroup.GroupName);
        Console.WriteLine(myGroup.GroupNumber[0]);
        Console.WriteLine(myGroup.GroupNumber[1]);
        Console.WriteLine(myGroup.Today);
        Console.WriteLine(myGroup.PositiveInt);
        Console.WriteLine(myGroup.IgnoreThis);
    }

    private XmlSerializer CreateOverrideSerializer()
    {
        SoapAttributeOverrides mySoapAttributeOverrides =
        new SoapAttributeOverrides();
        SoapAttributes soapAtts = new SoapAttributes();

        SoapElementAttribute mySoapElement = new SoapElementAttribute();
        mySoapElement.ElementName = "xxxx";
        soapAtts.SoapElement = mySoapElement;
        mySoapAttributeOverrides.Add(typeof(Group), "PositiveInt",
        soapAtts);

        // Overrides the IgnoreThis property.
        SoapIgnoreAttribute myIgnore = new SoapIgnoreAttribute();
        soapAtts = new SoapAttributes();
        soapAtts.SoapIgnore = false;
        mySoapAttributeOverrides.Add(typeof(Group), "IgnoreThis",
        soapAtts);

        // Overrides the GroupType enumeration.
        soapAtts = new SoapAttributes();
        SoapEnumAttribute xSoapEnum = new SoapEnumAttribute();
        xSoapEnum.Name = "Over1000";
        soapAtts.SoapEnum = xSoapEnum;

        // Adds the SoapAttributes to the
        // mySoapAttributeOverrides.
        mySoapAttributeOverrides.Add(typeof(GroupType), "large",
        soapAtts);

        // Creates a second enumeration and adds it.
        soapAtts = new SoapAttributes();
        xSoapEnum = new SoapEnumAttribute();
        xSoapEnum.Name = "ZeroTo1000";
        soapAtts.SoapEnum = xSoapEnum;
        mySoapAttributeOverrides.Add(typeof(GroupType), "small",
        soapAtts);

        // Overrides the Group type.
        soapAtts = new SoapAttributes();
        SoapTypeAttribute soapType = new SoapTypeAttribute();
        soapType.TypeName = "Team";
        soapAtts.SoapType = soapType;
    }
}

```

```
        SoapAttrs.Soaپtype = Soاپtpe,
        mySoapAttributeOverrides.Add(typeof(Group), soapAttrs);

        // Creates an XmlTypeMapping that is used to create an instance
        // of the XmlSerializer class. Then returns the XmlSerializer.
        XmlTypeMapping myMapping = (new SoapReflectionImporter(
        mySoapAttributeOverrides)).ImportTypeMapping(typeof(Group));

        XmlSerializer ser = new XmlSerializer(myMapping);
        return ser;
    }
}
```

See also

- [XML and SOAP Serialization](#)
- [Attributes That Control Encoded SOAP Serialization](#)
- [XML Serialization with XML Web Services](#)
- [How to: Serialize an Object](#)
- [How to: Deserialize an Object](#)
- [How to: Serialize an Object as a SOAP-Encoded XML Stream](#)

<system.xml.serialization> Element

9/20/2022 • 2 minutes to read • [Edit Online](#)

The top-level element for controlling XML serialization. For more information about configuration files, see [Configuration File Schema](#).

```
<configuration>
<system.xml.serialization>
```

Syntax

```
<system.xml.serialization>
</system.xml.serialization>
```

Attributes and Elements

The following sections describe attributes, child elements, and parent elements.

Attributes

None.

Child Elements

ELEMENT	DESCRIPTION
<dateTimeSerialization> Element	Determines the serialization mode of DateTime objects.
<schemalimporterExtensions> Element	Contains types that are used by the XmlSchemalimporter for mapping of XSD types to .NET types.

Parent Elements

ELEMENT	DESCRIPTION
<configuration> Element	The root element in every configuration file that is used by the common language runtime and .NET Framework applications.

Example

The following code example illustrates how to specify the serialization mode of a [DateTime](#) object, and the addition of types used by the [XmlSchemalimporter](#) when mapping XSD types to .NET types.

```
<system.xml.serialization>
  <xmlSerializer checkDeserializeAdvances="false" />
  <dateTimeSerialization mode = "Local" />
  <schemaImporterExtensions>
    <add
      name = "MobileCapabilities"
      type = "System.Web.Mobile.MobileCapabilities,
      System.Web.Mobile, Version=2.0.0.0, Culture=neutral,
      PublicKeyToken=b03f5f6f11d40a3a" />
  </schemaImporterExtensions>
</system.xml.serialization>
```

See also

- [XmlSchemaImporter](#)
- [DateTimeSerializationSection.DateTimeSerializationMode](#)
- [Configuration File Schema](#)
- [<dateTimeSerialization> Element](#)
- [<schemaImporterExtensions> Element](#)
- [<add> Element for <schemaImporterExtensions>](#)

<dateTimeSerialization> Element

9/20/2022 • 2 minutes to read • [Edit Online](#)

Determines the serialization mode of [DateTime](#) objects.

```
<configuration>
<dateTimeSerialization>
```

Syntax

```
<dateTimeSerialization
    mode = "Roundtrip|Local"
/>
```

Attributes and Elements

The following sections describe attributes, child elements, and parent elements.

Attributes

ATTRIBUTES	DESCRIPTION
mode	Optional. Specifies the serialization mode. Set to one of the DateTimeSerializationSection.DateTimeSerializationMode values. The default is RoundTrip .

Child Elements

None.

Parent Elements

ELEMENT	DESCRIPTION
system.xml.serialization	The top-level element for controlling XML serialization.

Remarks

When this property is set to **Local**, [DateTime](#) objects are always formatted as the local time. That is, local time zone information is always included with the serialized data.

When this property is set to **Roundtrip**, [DateTime](#) objects are examined to determine whether they are in the local, UTC, or an unspecified time zone. The [DateTime](#) objects are then serialized in such a way that this information is preserved. This is the default behavior and is the recommended behavior for all new applications that do not communicate with older versions of the framework.

See also

- [DateTime](#)
- [XmlSchemaImporter](#)
- [DateTimeSerializationSection.DateTimeSerializationMode](#)

- Configuration File Schema
- <schemalimporterExtensions> Element
- <add> Element for <schemalimporterExtensions>
- <system.xml.serialization> Element

<schemaImporterExtensions> element

9/20/2022 • 2 minutes to read • [Edit Online](#)

Contains types that are used by the [XmlSchemaImporter](#) for mapping of XSD types to .NET types. For more information about configuration files, see [Configuration File Schema](#).

Syntax

```
<schemaImporterExtensions>
    <!-- Add types -->
</schemaImporterExtensions>
```

Child Elements

ELEMENT	DESCRIPTION
<code><add></code> Element for <schemaImporterExtensions>	Adds types that are used by the XmlSchemaImporter to create mappings.

Parent Elements

ELEMENT	DESCRIPTION
<code><system.xml.serialization></code> Element	The top-level element for controlling XML serialization.

Example

The following code example illustrates how to add types that are used by the [XmlSchemaImporter](#) when mapping XSD types to .NET types.

```
<system.xml.serialization>
    <schemaImporterExtensions>
        <add name = "MobileCapabilities" type =
            "System.Web.Mobile.MobileCapabilities,
            System.Web.Mobile, Version - 2.0.0.0, Culture = neutral,
            PublicKeyToken = b03f5f6f11d40a3a" />
    </schemaImporterExtensions>
</system.xml.serialization>
```

See also

- [XmlSchemaImporter](#)
- [DateTimeSerializationSection.DateTimeSerializationMode](#)
- [Configuration File Schema](#)
- [`<dateTimeSerialization>` Element](#)
- [`<add>` Element for <schemaImporterExtensions>](#)
- [`<system.xml.serialization>` Element](#)

<add> Element for <schemalimporterExtensions>

9/20/2022 • 2 minutes to read • [Edit Online](#)

Adds types used by the [XmlSchemalImporter](#) for mapping XSD types to .NET types. For more information about configuration files, see [Configuration File Schema](#).

```
<configuration>
<system.xml.serialization>
<schemalimporterExtensions>
<add>
```

Syntax

```
<add name = "typeName" type="fully qualified type [,Version=version number] [,Culture=culture]
[,PublicKeyToken= token]"/>
```

Attributes and Elements

The following sections describe attributes, child elements, and parent elements.

Attributes

ATTRIBUTE	DESCRIPTION
name	A simple name that is used to find the instance.
type	Required. Specifies the schema extension class to add. The type attribute value must be on one line, and include the fully qualified type name. When the assembly is placed in the Global Assembly Cache (GAC), it must also include the version, culture, and public key token of the signed assembly.

Child Elements

None.

Parent Elements

ELEMENT	DESCRIPTION
<schemalimporterExtensions>	Contains the types that are used by the XmlSchemalImporter .

Example

The following code example adds an extension type that the [XmlSchemalImporter](#) can use when mapping types.

```
<configuration>
  <system.xml.serialization>
    <schemaImporterExtensions>
      <add name="contoso" type="System.Web.Mobile.MobileCapabilities,
        System.Web.Mobile, Version=2.0.0.0, Culture=neutral,
        PublicKeyToken=b03f5f7f11d50a3a" />
    </schemaImporterExtensions>
  </system.xml.serialization>
</configuration>
```

See also

- [XmlSchemaImporter](#)
- [<system.xml.serialization> Element](#)
- [<schemaImporterExtensions> Element](#)

<xmlSerializer> Element

9/20/2022 • 2 minutes to read • [Edit Online](#)

Specifies whether an additional check of progress of the [XmlSerializer](#) is done.

```
<configuration>
<system.xml.serialization>
```

Syntax

```
<xmlSerializer checkDeserializerAdvance = "true|false" />
```

Attributes and Elements

The following sections describe attributes, child elements, and parent elements.

Attributes

ATTRIBUTE	DESCRIPTION
checkDeserializeAdvances	Specifies whether the progress of the XmlSerializer is checked. Set the attribute to "true" or "false". The default is "true".
useLegacySerializationGeneration	Specifies whether the XmlSerializer uses legacy serialization generation which generates assemblies by writing C# code to a file and then compiling it to an assembly. The default is false .

Child Elements

None.

Parent Elements

ELEMENT	DESCRIPTION
<system.xml.serialization> Element	Contains configuration settings for the XmlSerializer and XmlSchemaImporter classes.

Remarks

By default, the [XmlSerializer](#) provides an additional layer of security against potential denial of service attacks when deserializing untrusted data. It does so by attempting to detect infinite loops during deserialization. If such a condition is detected, an exception is thrown with the following message: "Internal error: deserialization failed to advance over underlying stream."

Receiving this message does not necessarily indicate that a denial of service attack is in progress. In some rare circumstances, the infinite loop detection mechanism produces a false positive and the exception is thrown for a legitimate incoming message. If you find that in your particular application legitimate messages are being rejected by this extra layer of protection, set **checkDeserializeAdvances** attribute to "false".

Example

The following code example sets the `checkDeserializeAdvances` attribute to "false".

```
<configuration>
  <system.xml.serialization>
    <xmlSerializer checkDeserializeAdvances="false" />
  </system.xml.serialization>
</configuration>
```

See also

- [XmlSerializer](#)
- [<system.xml.serialization> Element](#)
- [XML and SOAP Serialization](#)

XML Serializer Generator Tool (Sgen.exe)

9/20/2022 • 2 minutes to read • [Edit Online](#)

The XML Serializer Generator creates an XML serialization assembly for types in a specified assembly. The serialization assembly improves the startup performance of a [XmlSerializer](#) when it serializes or deserializes objects of the specified types.

Syntax

Run the tool from the command line.

```
sgen [options]
```

TIP

For .NET Framework tools to function properly, you must either use [Visual Studio Developer Command Prompt](#) or [Visual Studio Developer PowerShell](#) or set the `Path`, `Include`, and `Lib` environment variables correctly. To set these environment variables, run `SDKVars.bat`, which is located in the `<SDK>|<version>|Bin` directory.

Parameters

OPTION	DESCRIPTION
<code>/a[sembly]:filename</code>	Generates serialization code for all the types contained in the assembly or executable specified by <code>filename</code> . Only one file name can be provided. If this argument is repeated, the last file name is used.
<code>/compiler:options</code>	Specifies the options to pass to the C# compiler. All csc.exe options are supported as they are passed to the compiler. This can be used to specify that the assembly should be signed and to specify the key file.
<code>/d[ebug]</code>	Generates an image that can be used with a debugger.
<code>/f[orce]</code>	Forces the overwriting of an existing assembly of the same name. The default is false .
<code>/help or /?</code>	Displays command syntax and options for the tool.
<code>/k[eep]</code>	Suppresses the deletion of the generated source files and other temporary files after they have been compiled into the serialization assembly. This can be used to determine whether the tool is generating serialization code for a particular type.
<code>/n[ologo]</code>	Suppresses the display of the Microsoft startup banner.

OPTION	DESCRIPTION
/o[ut]:path	Specifies the directory in which to save the generated assembly. Note: The name of the generated assembly is composed of the name of the input assembly plus "xmlSerializers.dll".
/p[roxytypes]	Generates serialization code only for the XML Web service proxy types.
/r[eference]:assemblyfiles	Specifies the assemblies that are referenced by the types requiring XML serialization. Accepts multiple assembly files separated by commas.
/s[ilent]	Suppresses the display of success messages.
/t[ype]:type	Generates serialization code only for the specified type.
/v[erbose]	Displays verbose output for debugging. Lists types from the target assembly that cannot be serialized with the XmlSerializer .
/?	Displays command syntax and options for the tool.

Remarks

When the XML Serializer Generator is not used, a [XmlSerializer](#) generates serialization code and a serialization assembly for each type every time an application is run. To improve the performance of XML serialization startup, use the Sgen.exe tool to generate those assemblies in advance. These assemblies can then be deployed with the application.

The XML Serializer Generator can also improve the performance of clients that use XML Web service proxies to communicate with servers because the serialization process will not incur a performance hit when the type is loaded the first time.

These generated assemblies cannot be used on the server side of a Web service. This tool is only for Web service clients and manual serialization scenarios.

If the assembly containing the type to serialize is named MyType.dll, then the associated serialization assembly will be named MyType.XmlSerializers.dll.

NOTE

The `srgen` tool is not compatible with `init`-only setters. The tool will fail if the target assembly contains any public properties that use this feature.

Examples

The following command creates an assembly named Data.XmlSerializers.dll for serializing all the types contained in the assembly named Data.dll.

```
srgen Data.dll
```

The Data.XmlSerializers.dll assembly can be referenced from code that needs to serialize and deserialize the

types in Data.dll.

See also

- [Tools](#)
- [Developer command-line shells](#)

XML Schema Definition Tool (Xsd.exe)

9/20/2022 • 9 minutes to read • [Edit Online](#)

The XML Schema Definition (Xsd.exe) tool generates XML schema or common language runtime classes from XDR, XML, and XSD files, or from classes in a runtime assembly.

The XML Schema Definition tool (Xsd.exe) usually can be found in the following path:

C:\Program Files (x86)\Microsoft SDKs\Windows\{version}\bin\NETFX {version} Tools

Syntax

Run the tool from the command line.

```
xsd file.xdr [-outputdir:directory][/parameters:file.xml]
xsd file.xml [-outputdir:directory] [/parameters:file.xml]
xsd file.xsd {/classes | /dataset} [/element:element]
    [/enableLinqDataSet] [/language:language]
    [/namespace:namespace] [-outputdir:directory] [URI:uri]
    [/parameters:file.xml]
xsd {file.dll | file.exe} [-outputdir:directory] [/type:typename [...]][/parameters:file.xml]
```

TIP

For .NET Framework tools to function properly, you must set your `Path`, `Include`, and `Lib` environment variables correctly. Set these environment variables by running `SDKVars.bat`, which is located in the `<SDK>\<version>\Bin` directory. `SDKVars.bat` must be executed in every command shell.

Argument

ARGUMENT	DESCRIPTION
----------	-------------

ARGUMENT	DESCRIPTION
<i>file.extension</i>	<p>Specifies the input file to convert. You must specify the extension as one of the following: .xdr, .xml, .xsd, .dll, or .exe.</p> <p>If you specify an XDR schema file (.xdr extension), Xsd.exe converts the XDR schema to an XSD schema. The output file has the same name as the XDR schema, but with the .xsd extension.</p> <p>If you specify an XML file (.xml extension), Xsd.exe infers a schema from the data in the file and produces an XSD schema. The output file has the same name as the XML file, but with the .xsd extension.</p> <p>If you specify an XML schema file (.xsd extension), Xsd.exe generates source code for runtime objects that correspond to the XML schema.</p> <p>If you specify a runtime assembly file (.exe or .dll extension), Xsd.exe generates schemas for one or more types in that assembly. You can use the <code>/type</code> option to specify the types for which to generate schemas. The output schemas are named schema0.xsd, schema1.xsd, and so on. Xsd.exe produces multiple schemas only if the given types specify a namespace using the <code>XMLRoot</code> custom attribute.</p>

General Options

OPTION	DESCRIPTION
<code>/h[elp]</code>	Displays command syntax and options for the tool.
<code>/o[utputdir]:directory</code>	Specifies the directory for output files. This argument can appear only once. The default is the current directory.
<code>/?</code>	Displays command syntax and options for the tool.
<code>/p[arameters]:file.xml</code>	Read options for various operation modes from the specified .xml file. The short form is <code>/p:</code> . For more information, see the Remarks section.

XSD File Options

You must specify only one of the following options for .xsd files.

OPTION	DESCRIPTION
<code>/c[lasses]</code>	Generates classes that correspond to the specified schema. To read XML data into the object, use the XmlSerializer.Deserialize method.
<code>/d[ataset]</code>	Generates a class derived from DataSet that corresponds to the specified schema. To read XML data into the derived class, use the DataSet.ReadXml method.

You can also specify any of the following options for .xsd files.

OPTION	DESCRIPTION
/e[lement]: <i>element</i>	Specifies the element in the schema to generate code for. By default all elements are typed. You can specify this argument more than once.
/enableDataBinding	Implements the INotifyPropertyChanged interface on all generated types to enable data binding. The short form is <code>/edb</code> .
/enableLinqDataSet	(Short form: <code>/eld</code>) Specifies that the generated DataSet can be queried against using LINQ to DataSet. This option is used when the /dataset option is also specified. For more information, see LINQ to DataSet Overview and Querying Typed DataSets . For general information about using LINQ, see Language-Integrated Query (LINQ) - C# or Language-Integrated Query (LINQ) - Visual Basic .
/f[ields]	Generates fields only. By default, properties with backing fields are generated.
/l[anguage]: <i>language</i>	Specifies the programming language to use. Choose from <code>cs</code> (C#, which is the default), <code>vb</code> (Visual Basic), <code>js</code> (JScript), or <code>vjs</code> (Visual J#). You can also specify a fully qualified name for a class implementing System.CodeDom.Compiler.CodeDomProvider
/n[amespace]: <i>namespace</i>	Specifies the runtime namespace for the generated types. The default namespace is <code>Schemas</code> .
/nologo	Suppresses the banner.
/order	Generates explicit order identifiers on all particle members.
/o[ut]: <i>directoryName</i>	Specifies the output directory to place the files in. The default is the current directory.
/u[ri]: <i>uri</i>	Specifies the URI for the elements in the schema to generate code for. This URI, if present, applies to all elements specified with the <code>/element</code> option.

DLL and EXE File Options

OPTION	DESCRIPTION
/t[type]: <i>typename</i>	Specifies the name of the type to create a schema for. You can specify multiple type arguments. If <i>typename</i> does not specify a namespace, Xsd.exe matches all types in the assembly with the specified type. If <i>typename</i> specifies a namespace, only that type is matched. If <i>typename</i> ends with an asterisk character (*), the tool matches all types that start with the string preceding the *. If you omit the <code>/type</code> option, Xsd.exe generates schemas for all types in the assembly.

Remarks

The following table shows the operations that Xsd.exe performs.

OPERATION	DESCRIPTION
XDR to XSD	Generates an XML schema from an XML-Data-Reduced schema file. XDR is an early XML-based schema format.
XML to XSD	Generates an XML schema from an XML file.
XSD to DataSet	Generates common language runtime DataSet classes from an XSD schema file. The generated classes provide a rich object model for regular XML data.
XSD to Classes	Generates runtime classes from an XSD schema file. The generated classes can be used in conjunction with System.Xml.Serialization.XmlSerializer to read and write XML code that follows the schema.
Classes to XSD	Generates an XML schema from a type or types in a runtime assembly file. The generated schema defines the XML format used by the XmlSerializer .

Xsd.exe only allows you to manipulate XML schemas that follow the XML Schema Definition (XSD) language proposed by the World Wide Web Consortium (W3C). For more information on the XML Schema Definition proposal or the XML standard, see <https://w3.org>.

Setting Options with an XML File

By using the `/parameters` switch, you can specify a single XML file that sets various options. The options you can set depend on how you are using the XSD.exe tool. Choices include generating schemas, generating code files, or generating code files that include `DataSet` features. For example, you can set the `<assembly>` element to the name of an executable (.exe) or type library (.dll) file when generating a schema, but not when generating a code file. The following XML shows how to use the `<generateSchemas>` element with a specified executable:

```
<!-- This is in a file named GenerateSchemas.xml. -->
<xsd xmlns='http://microsoft.com/dotnet/tools/xsd/'>
<generateSchemas>
    <assembly>ConsoleApplication1.exe</assembly>
</generateSchemas>
</xsd>
```

If the preceding XML is contained in a file named GenerateSchemas.xml, then use the `/parameters` switch by typing the following at a command prompt and pressing Enter:

```
xsd /p:GenerateSchemas.xml
```

On the other hand, if you are generating a schema for a single type found in the assembly, you can use the following XML:

```

<!-- This is in a file named GenerateSchemaFromType.xml. -->
<xsd xmlns='http://microsoft.com/dotnet/tools/xsd'>
<generateSchemas>
  <type>IDIItems</type>
</generateSchemas>
</xsd>

```

But to use preceding code, you must also supply the name of the assembly at the command prompt. Enter the following at a command prompt (presuming the XML file is named GenerateSchemaFromType.xml):

```
xsd /p:GenerateSchemaFromType.xml ConsoleApplication1.exe
```

You must specify only one of the following options for the `<generateSchemas>` element.

ELEMENT	DESCRIPTION
<code><assembly></code>	Specifies an assembly to generate the schema from.
<code><type></code>	Specifies a type found in an assembly to generate a schema for.
<code><xml></code>	Specifies an XML file to generate a schema for.
<code><xdr></code>	Specifies an XDR file to generate a schema for.

To generate a code file, use the `<generateClasses>` element. The following example generates a code file. Note that two attributes are also shown that allow you to set the programming language and namespace of the generated file.

```

<xsd xmlns='http://microsoft.com/dotnet/tools/xsd'>
<generateClasses language='VB' namespace='Microsoft.Serialization.Examples' />
</xsd>
<!-- You must supply an .xsd file when typing in the command line.-->
<!-- For example: xsd /p:genClasses mySchema.xsd -->

```

Options you can set for the `<generateClasses>` element include the following.

ELEMENT	DESCRIPTION
<code><element></code>	Specifies an element in the .xsd file to generate code for.
<code><schemalimporterExtensions></code>	Specifies a type derived from the SchemalimporterExtension class.
<code><schema></code>	Specifies a XML Schema file to generate code for. Multiple XML Schema files can be specified using multiple <code><schema></code> elements.

The following table shows the attributes that can also be used with the `<generateClasses>` element.

ATTRIBUTE	DESCRIPTION

ATTRIBUTE	DESCRIPTION
language	Specifies the programming language to use. Choose from <code>cs</code> (C#, the default), <code>vb</code> (Visual Basic), <code>js</code> (JScript), or <code>vjs</code> (Visual J#). You can also specify a fully qualified name for a class that implements CodeDomProvider .
namespace	Specifies the namespace for the generated code. The namespace must conform to CLR standards (for example, no spaces or backslash characters).
options	One of the following values: <code>none</code> , <code>properties</code> (generates properties instead of public fields), <code>order</code> , or <code>enableDataBinding</code> (see the <code>/order</code> and <code>/enableDataBinding</code> switches in the preceding XSD File Options section).

You can also control how `DataSet` code is generated by using the `<generateDataSet>` element. The following XML specifies that the generated code uses `DataSet` structures (such as the `DataTable` class) to create Visual Basic code for a specified element. The generated DataSet structures will support LINQ queries.

```
<xsd xmlns='http://microsoft.com/dotnet/tools/xsd/'>
  <generateDataSet language='VB' namespace='Microsoft.Serialization.Examples' enableLinqDataSet='true'>
  </generateDataSet>
</xsd>
```

Options you can set for the `<generateDataSet>` element include the following.

ELEMENT	DESCRIPTION
<code><schema></code>	Specifies an XML Schema file to generate code for. Multiple XML Schema files can be specified using multiple <code><schema></code> elements.

The following table shows the attributes that can be used with the `<generateDataSet>` element.

ATTRIBUTE	DESCRIPTION
<code>enableLinqDataSet</code>	Specifies that the generated DataSet can be queried against using LINQ to DataSet. The default value is false.
<code>language</code>	Specifies the programming language to use. Choose from <code>cs</code> (C#, the default), <code>vb</code> (Visual Basic), <code>js</code> (JScript), or <code>vjs</code> (Visual J#). You can also specify a fully qualified name for a class that implements CodeDomProvider .
<code>namespace</code>	Specifies the namespace for the generated code. The namespace must conform to CLR standards (for example, no spaces or backslash characters).

There are attributes that you can set on the top level `<xsd>` element. These options can be used with any of the child elements (`<generateSchemas>`, `<generateClasses>` or `<generateDataSet>`). The following XML code generates code for an element named "IDItems" in the output directory named "MyOutputDirectory".

```

<xsd xmlns='http://microsoft.com/dotnet/tools/xsd/' output='MyOutputDirectory'>
<generateClasses>
    <element>IDItems</element>
</generateClasses>
</xsd>

```

The following table shows the attributes that can also be used with the `<xsd>` element.

ATTRIBUTE	DESCRIPTION
output	The name of a directory where the generated schema or code file will be placed.
nologo	Suppresses the banner. Set to <code>true</code> or <code>false</code> .
help	Displays command syntax and options for the tool. Set to <code>true</code> or <code>false</code> .

Examples

The following command generates an XML schema from `myFile.xdr` and saves it to the current directory.

```
xsd myFile.xdr
```

The following command generates an XML schema from `myFile.xml` and saves it to the specified directory.

```
xsd myFile.xml /outputdir:myOutputDir
```

The following command generates a data set that corresponds to the specified schema in the C# language and saves it as `XSDSchemaFile.cs` in the current directory.

```
xsd /dataset /language:CS XSDSchemaFile.xsd
```

The following command generates XML schemas for all types in the assembly `myAssembly.dll` and saves them as `schema0.xsd` in the current directory.

```
xsd myAssembly.dll
```

See also

- [DataSet](#)
- [System.Xml.Serialization.XmlSerializer](#)
- [Tools](#)
- [Developer command-line shells](#)
- [LINQ to DataSet Overview](#)
- [Querying Typed DataSets](#)
- [LINQ \(Language-Integrated Query\) \(C#\)](#)
- [LINQ \(Language-Integrated Query\) \(Visual Basic\)](#)

System.CommandLine overview

9/20/2022 • 2 minutes to read • [Edit Online](#)

IMPORTANT

`System.CommandLine` is currently in PREVIEW, and this documentation is for version 2.0 beta 4. Some information relates to prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, express or implied, with respect to the information provided here.

The `System.CommandLine` library provides functionality that is commonly needed by command-line apps, such as parsing the command-line input and displaying help text.

Apps that use `System.CommandLine` include the .NET CLI, [additional tools](#), and many [global and local tools](#).

For app developers, the library:

- Lets you focus on writing your app code, since you don't have to write code to parse command-line input or produce a help page.
- Lets you test app code independently of input parsing code.
- Is [trim-friendly](#), making it a good choice for developing a fast, lightweight, AOT-capable CLI app.

Use of the library also benefits app users:

- It ensures that command-line input is parsed consistently according to [POSIX](#) or Windows conventions.
- It automatically supports [tab completion](#) and [response files](#).

NuGet package

The library is available in a NuGet package:

- [System.CommandLine](#)

Next steps

To get started with System.CommandLine, see the following resources:

- [Tutorial: Get started with System.CommandLine](#)
- [Command-line syntax overview](#)

To learn more, see the following resources:

- [How to define commands, options, and arguments](#)
- [How to bind arguments to handlers](#)
- [How to configure dependency injection](#)
- [How to enable and customize tab completion](#)
- [How to customize help](#)
- [How to handle termination](#)
- [How to write middleware and directives](#)
- [System.CommandLine API reference](#)

Tutorial: Get started with System.CommandLine

9/20/2022 • 14 minutes to read • [Edit Online](#)

IMPORTANT

`System.CommandLine` is currently in PREVIEW, and this documentation is for version 2.0 beta 4. Some information relates to prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, express or implied, with respect to the information provided here.

This tutorial shows how to create a .NET command-line app that uses the `System.CommandLine` library. You'll begin by creating a simple root command that has one option. Then you'll add to that base, creating a more complex app that contains multiple subcommands and different options for each command.

In this tutorial, you learn how to:

- Create commands, options, and arguments.
- Specify default values for options.
- Assign options and arguments to commands.
- Assign an option recursively to all subcommands under a command.
- Work with multiple levels of nested subcommands.
- Create aliases for commands and options.
- Work with `string`, `string[]`, `int`, `bool`, `FileInfo` and enum option types.
- Bind option values to command handler code.
- Use custom code for parsing and validating options.

Prerequisites

- A code editor, such as [Visual Studio Code](#) with the [C# extension](#).
- The [.NET 6 SDK](#).

Or

- [Visual Studio 2022](#) with the [.NET desktop development](#) workload installed.

Create the app

Create a .NET 6 console app project named "scl".

1. Create a folder named `scl` for the project, and then open a command prompt in the new folder.
2. Run the following command:

```
dotnet new console --framework net6.0
```

Install the System.CommandLine package

- Run the following command:

```
dotnet add package System.CommandLine --prerelease
```

The `--prerelease` option is necessary because the library is still in beta.

1. Replace the contents of *Program.cs* with the following code:

```
using System.CommandLine;

namespace scl;

class Program
{
    static async Task<int> Main(string[] args)
    {
        var fileOption = new Option<FileInfo?>(
            name: "--file",
            description: "The file to read and display on the console.");

        var rootCommand = new RootCommand("Sample app for System.CommandLine");
        rootCommand.AddOption(fileOption);

        rootCommand.SetHandler((file) =>
        {
            ReadFile(file!);
        },
        fileOption);

        return await rootCommand.InvokeAsync(args);
    }

    static void ReadFile(FileInfo file)
    {
        File.ReadLines(file.FullName).ToList()
            .ForEach(line => Console.WriteLine(line));
    }
}
```

The preceding code:

- Creates an `option` named `--file` of type `FileInfo` and assigns it to the `root command`:

```
var fileOption = new Option<FileInfo?>(
    name: "--file",
    description: "The file to read and display on the console.");

var rootCommand = new RootCommand("Sample app for System.CommandLine");
rootCommand.AddOption(fileOption);
```

- Specifies that `ReadFile` is the method that will be called when the root command is invoked:

```
rootCommand.SetHandler((file) =>
{
    ReadFile(file!);
},
fileOption);
```

- Displays the contents of the specified file when the root command is invoked:

```
static void ReadFile(FileInfo file)
{
    File.ReadLines(file.FullName).ToList()
        .ForEach(line => Console.WriteLine(line));
}
```

Test the app

You can use any of the following ways to test while developing a command-line app:

- Run the `dotnet build` command, and then open a command prompt in the `scl/bin/Debug/net6.0` folder to run the executable:

```
dotnet build
cd bin/Debug/net6.0
scl --file scl.runtimeconfig.json
```

- Use `dotnet run` and pass option values to the app instead of to the `run` command by including them after `--`, as in the following example:

```
dotnet run -- --file scl.runtimeconfig.json
```

In .NET 7.0.100 SDK Preview, you can use the `commandLineArgs` of a `launchSettings.json` file by running the command `dotnet run --launch-profile <profilename>`.

- [Publish the project to a folder](#), open a command prompt to that folder, and run the executable:

```
dotnet publish -o publish
cd ./publish
scl --file scl.runtimeconfig.json
```

- In Visual Studio 2022, select **Debug > Debug Properties** from the menu, and enter the options and arguments in the **Command line arguments** box. For example:



Then run the app, for example by pressing Ctrl+F5.

This tutorial assumes you're using the first of these options.

When you run the app, it displays the contents of the file specified by the `--file` option.

```
{  
    "runtimeOptions": {  
        "tfm": "net6.0",  
        "framework": {  
            "name": "Microsoft.NETCore.App",  
            "version": "6.0.0"  
        }  
    }  
}
```

Help output

`System.CommandLine` automatically provides help output:

```
scl --help
```

```
Description:  
  Sample app for System.CommandLine
```

```
Usage:  
  scl [options]
```

```
Options:  
  --file <file>  The file to read and display on the console.  
  --version        Show version information  
  -?, -h, --help  Show help and usage information
```

Version output

`System.CommandLine` automatically provides version output:

```
scl --version
```

```
1.0.0
```

Add a subcommand and options

In this section, you:

- Create more options.
- Create a subcommand.
- Assign the new options to the new subcommand.

The new options will let you configure the foreground and background text colors and the readout speed. These features will be used to read a collection of quotes that comes from the [Teleprompter console app tutorial](#).

1. Copy the `sampleQuotes.txt` file from the GitHub repository for this sample into your project directory. For information on how to download files, see the instructions in [Samples and Tutorials](#).
2. Open the project file and add an `<ItemGroup>` element just before the closing `</Project>` tag:

```
<ItemGroup>
  <Content Include="sampleQuotes.txt">
    <CopyToOutputDirectory>Always</CopyToOutputDirectory>
  </Content>
</ItemGroup>
```

Adding this markup causes the text file to be copied to the `bin/debug/net6.0` folder when you build the app. So when you run the executable in that folder, you can access the file by name without specifying a folder path.

3. In `Program.cs`, after the code that creates the `--file` option, create options to control the readout speed and text colors:

```
var delayOption = new Option<int>(
  name: "--delay",
  description: "Delay between lines, specified as milliseconds per character in a line.",
  getDefaultValue: () => 42);

var fgcolorOption = new Option<ConsoleColor>(
  name: "--fgcolor",
  description: "Foreground color of text displayed on the console.",
  getDefaultValue: () => ConsoleColor.White);

var lightModeOption = new Option<bool>(
  name: "--light-mode",
  description: "Background color of text displayed on the console: default is black, light mode is white.");
```

4. After the line that creates the root command, delete the line that adds the `--file` option to it. You're removing it here because you'll add it to a new subcommand.

```
var rootCommand = new RootCommand("Sample app for System.CommandLine");
//rootCommand.AddOption(fileOption);
```

5. After the line that creates the root command, create a `read` subcommand. Add the options to this subcommand, and add the subcommand to the root command.

```
var readCommand = new Command("read", "Read and display the file.")
{
  fileOption,
  delayOption,
  fgcolorOption,
  lightModeOption
};
rootCommand.AddCommand(readCommand);
```

6. Replace the `SetHandler` code with the following `SetHandler` code for the new subcommand:

```
readCommand.SetHandler(async (file, delay, fgcolor, lightMode) =>
{
  await ReadFile(file!, delay, fgcolor, lightMode);
},
fileOption, delayOption, fgcolorOption, lightModeOption);
```

You're no longer calling `SetHandler` on the root command because the root command no longer needs a handler. When a command has subcommands, you typically have to specify one of the subcommands when invoking a command-line app.

7. Replace the `ReadFile` handler method with the following code:

```
internal static async Task ReadFile(  
    FileInfo file, int delay, ConsoleColor fgColor, bool lightMode)  
{  
    Console.BackgroundColor = lightMode ? ConsoleColor.White : ConsoleColor.Black;  
    Console.ForegroundColor = fgColor;  
    List<string> lines = File.ReadLines(file.FullName).ToList();  
    foreach (string line in lines)  
    {  
        Console.WriteLine(line);  
        await Task.Delay(delay * line.Length);  
    };  
}
```

The app now looks like this:

```

using System.CommandLine;

namespace scl;

class Program
{
    static int Main(string[] args)
    {
        var fileOption = new Option<FileInfo?>(
            name: "--file",
            description: "The file to read and display on the console.");

        var delayOption = new Option<int>(
            name: "--delay",
            description: "Delay between lines, specified as milliseconds per character in a line.",
            getDefaultValue: () => 42);

        var fgcolorOption = new Option<ConsoleColor>(
            name: "--fgcolor",
            description: "Foreground color of text displayed on the console.",
            getDefaultValue: () => ConsoleColor.White);

        var lightModeOption = new Option<bool>(
            name: "--light-mode",
            description: "Background color of text displayed on the console: default is black, light mode is white.");

        var rootCommand = new RootCommand("Sample app for System.CommandLine");
        //rootCommand.AddOption(fileOption);

        var readCommand = new Command("read", "Read and display the file.")
        {
            fileOption,
            delayOption,
            fgcolorOption,
            lightModeOption
        };
        rootCommand.AddCommand(readCommand);

        readCommand.SetHandler(async (file, delay, fgcolor, lightMode) =>
        {
            await ReadFile(file!, delay, fgcolor, lightMode);
        },
        fileOption, delayOption, fgcolorOption, lightModeOption);

        return rootCommand.InvokeAsync(args).Result;
    }

    internal static async Task ReadFile(
        FileInfo file, int delay, ConsoleColor fgColor, bool lightMode)
    {
        Console.BackgroundColor = lightMode ? ConsoleColor.White : ConsoleColor.Black;
        Console.ForegroundColor = fgColor;
        List<string> lines = File.ReadLines(file.FullName).ToList();
        foreach (string line in lines)
        {
            Console.WriteLine(line);
            await Task.Delay(delay * line.Length);
        };
    }
}

```

Test the new subcommand

Now if you try to run the app without specifying the subcommand, you get an error message followed by a help

message that specifies the subcommand that is available.

```
scl --file sampleQuotes.txt
```

```
'--file' was not matched. Did you mean one of the following?
--help
Required command was not provided.
Unrecognized command or argument '--file'.
Unrecognized command or argument 'sampleQuotes.txt'.

Description:
  Sample app for System.CommandLine

Usage:
  scl [command] [options]

Options:
  --version      Show version information
  -?, -h, --help Show help and usage information

Commands:
  read  Read and display the file.
```

The help text for subcommand `read` shows that four options are available. It shows valid values for the enum.

```
scl read -h
```

```
Description:
  Read and display the file.

Usage:
  scl read [options]

Options:
  --file <file>                                The file to read and display on the console.
  --delay <delay>                                Delay between lines, specified as milliseconds
per                                         character in a line. [default: 42]
                                         Foreground color of text displayed on the
console.                                         <Black|Blue|Cyan|DarkBlue|DarkCyan|DarkGray|DarkGreen|Dark
Magenta|DarkRed|DarkYellow|Gray|Green|Magenta|Red|White|Ye
llow>
  --light-mode                                     Background color of text displayed on the
console:                                         default is black, light mode is white.
                                         Show help and usage information
  -?, -h, --help
```

Run subcommand `read` specifying only the `--file` option, and you get the default values for the other three options.

```
scl read --file sampleQuotes.txt
```

The 42 milliseconds per character default delay causes a slow readout speed. You can speed it up by setting `--delay` to a lower number.

```
scl read --file sampleQuotes.txt --delay 0
```

You can use `--fgcolor` and `--light-mode` to set text colors:

```
scl read --file sampleQuotes.txt --fgcolor red --light-mode
```

Provide an invalid value for `--delay` and you get an error message:

```
scl read --file sampleQuotes.txt --delay forty-two
```

```
Cannot parse argument 'forty-two' for option '--int' as expected type 'System.Int32'.
```

Provide an invalid value for `--file` and you get an exception:

```
scl read --file nofile
```

```
Unhandled exception: System.IO.FileNotFoundException:  
Could not find file 'C:\bin\Debug\net6.0\nofile'.
```

Add subcommands and custom validation

This section creates the final version of the app. When finished, the app will have the following commands and options:

- root command with a global* option named `--file`
 - `quotes` command
 - `read` command with options named `--delay`, `--fgcolor`, and `--light-mode`
 - `add` command with arguments named `quote` and `byline`
 - `delete` command with option named `--search-terms`

* A global option is available to the command it's assigned to and recursively to all its subcommands.

Here's sample command line input that invokes each of the available commands with its options and arguments:

```
scl quotes read --file sampleQuotes.txt --delay 40 --fgcolor red --light-mode  
scl quotes add "Hello world!" "Nancy Davolio"  
scl quotes delete --search-terms David "You can do" Antoine "Perfection is achieved"
```

1. In `Program.cs`, replace the code that creates the `--file` option with the following code:

```

var fileOption = new Option<FileInfo?>(
    name: "--file",
    description: "An option whose argument is parsed as a FileInfo",
    isDefault: true,
    parseArgument: result =>
{
    if (result.Tokens.Count == 0)
    {
        return new FileInfo("sampleQuotes.txt");
    }
    string? filePath = result.Tokens.Single().Value;
    if (!File.Exists(filePath))
    {
        result.ErrorMessage = "File does not exist";
        return null;
    }
    else
    {
        return new FileInfo(filePath);
    }
});

```

This code uses `ParseArgument<T>` to provide custom parsing, validation, and error handling.

Without this code, missing files are reported with an exception and stack trace. With this code just the specified error message is displayed.

This code also specifies a default value, which is why it sets `isDefault` to `true`. If you don't set `isDefault` to `true`, the `parseArgument` delegate doesn't get called when no input is provided for `--file`.

- After the code that creates `lightModeOption`, add options and arguments for the `add` and `delete` commands:

```

var searchTermsOption = new Option<string[]>(
    name: "--search-terms",
    description: "Strings to search for when deleting entries.")
{ IsRequired = true, AllowMultipleArgumentsPerToken = true };

var quoteArgument = new Argument<string>(
    name: "quote",
    description: "Text of quote.");

var bylineArgument = new Argument<string>(
    name: "byline",
    description: "Byline of quote.");

```

The `AllowMultipleArgumentsPerToken` setting lets you omit the `--search-terms` option name when specifying elements in the list after the first one. It makes the following examples of command-line input equivalent:

```

scl quotes delete --search-terms David "You can do"
scl quotes delete --search-terms David --search-terms "You can do"

```

- Replace the code that creates the root command and the `read` command with the following code:

```

var rootCommand = new RootCommand("Sample app for System.CommandLine");
rootCommand.AddGlobalOption(fileOption);

var quotesCommand = new Command("quotes", "Work with a file that contains quotes.");
rootCommand.AddCommand(quotesCommand);

var readCommand = new Command("read", "Read and display the file.")
{
    delayOption,
    fgcolorOption,
    lightModeOption
};
quotesCommand.AddCommand(readCommand);

var deleteCommand = new Command("delete", "Delete lines from the file.");
deleteCommand.AddOption(searchTermsOption);
quotesCommand.AddCommand(deleteCommand);

var addCommand = new Command("add", "Add an entry to the file.");
addCommand.AddArgument(quoteArgument);
addCommand.AddArgument(bylineArgument);
addCommand.AddAlias("insert");
quotesCommand.AddCommand(addCommand);

```

This code makes the following changes:

- Removes the `--file` option from the `read` command.
- Adds the `--file` option as a global option to the root command.
- Creates a `quotes` command and adds it to the root command.
- Adds the `read` command to the `quotes` command instead of to the root command.
- Creates `add` and `delete` commands and adds them to the `quotes` command.

The result is the following command hierarchy:

- Root command
 - `quotes`
 - `read`
 - `add`
 - `delete`

The app now implements the recommended pattern where the parent command (`quotes`) specifies an area or group, and its children commands (`read`, `add`, `delete`) are actions.

Global options are applied to the command and recursively to subcommands. Since `--file` is on the root command, it will be available automatically in all subcommands of the app.

4. After the `SetHandler` code, add new `SetHandler` code for the new subcommands:

```

deleteCommand.SetHandler((file, searchTerms) =>
{
    DeleteFromFile(file!, searchTerms);
},
fileOption, searchTermsOption);

addCommand.SetHandler((file, quote, byline) =>
{
    AddToFile(file!, quote, byline);
},
fileOption, quoteArgument, bylineArgument);

```

Subcommand `quotes` doesn't have a handler because it isn't a leaf command. Subcommands `read`, `add`, and `delete` are leaf commands under `quotes`, and `SetHandler` is called for each of them.

5. Add the handlers for `add` and `delete`.

```

internal static void DeleteFromFile(FileInfo file, string[] searchTerms)
{
    Console.WriteLine("Deleting from file");
    File.WriteAllLines(
        file.FullName, File.ReadLines(file.FullName)
            .Where(line => searchTerms.All(s => !line.Contains(s))).ToList());
}

internal static void AddToFile(FileInfo file, string quote, string byline)
{
    Console.WriteLine("Adding to file");
    using StreamWriter? writer = file.AppendText();
    writer.WriteLine($"{Environment.NewLine}{Environment.NewLine}{quote}");
    writer.WriteLine($"{Environment.NewLine}-{byline}");
    writer.Flush();
}

```

The finished app looks like this:

```

using System.CommandLine;

namespace scl;

class Program
{
    static async Task<int> Main(string[] args)
    {
        var fileOption = new Option<FileInfo?>(
            name: "--file",
            description: "An option whose argument is parsed as a FileInfo",
            isDefault: true,
            parseArgument: result =>
        {
            if (result.Tokens.Count == 0)
            {
                return new FileInfo("sampleQuotes.txt");
            }
            string? filePath = result.Tokens.Single().Value;
            if (!File.Exists(filePath))
            {
                result.ErrorMessage = "File does not exist";
                return null;
            }
            else
            {
                return new FileInfo(filePath);
            }
        });
    }
}

```

```

        });

var delayOption = new Option<int>(
    name: "--delay",
    description: "Delay between lines, specified as milliseconds per character in a line.",
    getDefaultValue: () => 42);

var fgcolorOption = new Option<ConsoleColor>(
    name: "--fgcolor",
    description: "Foreground color of text displayed on the console.",
    getDefaultValue: () => ConsoleColor.White);

var lightModeOption = new Option<bool>(
    name: "--light-mode",
    description: "Background color of text displayed on the console: default is black, light mode is white.");
}

var searchTermsOption = new Option<string[]>(
    name: "--search-terms",
    description: "Strings to search for when deleting entries."
    { IsRequired = true, AllowMultipleArgumentsPerToken = true });

var quoteArgument = new Argument<string>(
    name: "quote",
    description: "Text of quote.");

var bylineArgument = new Argument<string>(
    name: "byline",
    description: "Byline of quote.");

var rootCommand = new RootCommand("Sample app for System.CommandLine");
rootCommand.AddGlobalOption(fileOption);

var quotesCommand = new Command("quotes", "Work with a file that contains quotes.");
rootCommand.AddCommand(quotesCommand);

var readCommand = new Command("read", "Read and display the file.")
{
    delayOption,
    fgcolorOption,
    lightModeOption
};
quotesCommand.AddCommand(readCommand);

var deleteCommand = new Command("delete", "Delete lines from the file.");
deleteCommand.AddOption(searchTermsOption);
quotesCommand.AddCommand(deleteCommand);

var addCommand = new Command("add", "Add an entry to the file.");
addCommand.AddArgument(quoteArgument);
addCommand.AddArgument(bylineArgument);
addCommand.AddAlias("insert");
quotesCommand.AddCommand(addCommand);

readCommand.SetHandler(async (file, delay, fgcolor, lightMode) =>
{
    await ReadFile(file!, delay, fgcolor, lightMode);
},
fileOption, delayOption, fgcolorOption, lightModeOption);

deleteCommand.SetHandler((file, searchTerms) =>
{
    DeleteFromFile(file!, searchTerms);
},
fileOption, searchTermsOption);

addCommand.SetHandler((file, quote, byline) =>
{
    ...
}
);
}

```

```

        AddToFile(file!, quote, byline);
    },
    fileOption, quoteArgument, bylineArgument);

    return await rootCommand.InvokeAsync(args);
}

internal static async Task ReadFile(
    FileInfo file, int delay, ConsoleColor fgColor, bool lightMode)
{
    Console.BackgroundColor = lightMode ? ConsoleColor.White : ConsoleColor.Black;
    Console.ForegroundColor = fgColor;
    var lines = File.ReadLines(file.FullName).ToList();
    foreach (string line in lines)
    {
        Console.WriteLine(line);
        await Task.Delay(delay * line.Length);
    };
}

internal static void DeleteFromFile(FileInfo file, string[] searchTerms)
{
    Console.WriteLine("Deleting from file");
    File.WriteAllLines(
        file.FullName, File.ReadLines(file.FullName)
            .Where(line => searchTerms.All(s => !line.Contains(s))).ToList());
}
internal static void AddToFile(FileInfo file, string quote, string byline)
{
    Console.WriteLine("Adding to file");
    using StreamWriter? writer = file.AppendText();
    writer.WriteLine($"{Environment.NewLine}{Environment.NewLine}{quote}");
    writer.WriteLine($"{Environment.NewLine}-{byline}");
    writer.Flush();
}
}
}

```

Build the project, and then try the following commands.

Submit a nonexistent file to `--file` with the `read` command, and you get an error message instead of an exception and stack trace:

```
scl quotes read --file nofile
```

```
File does not exist
```

Try to run subcommand `quotes` and you get a message directing you to use `read`, `add`, or `delete`:

```
scl quotes
```

```
Required command was not provided.
```

Description:

```
Work with a file that contains quotes.
```

Usage:

```
scl quotes [command] [options]
```

Options:

```
--file <file> An option whose argument is parsed as a FileInfo [default: sampleQuotes.txt]  
-?, -h, --help Show help and usage information
```

Commands:

read	Read and display the file.
delete	Delete lines from the file.
add, insert <quote> <byline>	Add an entry to the file.

Run subcommand `add`, and then look at the end of the text file to see the added text:

```
scl quotes add "Hello world!" "Nancy Davolio"
```

Run subcommand `delete` with search strings from the beginning of the file, and then look at the beginning of the text file to see where text was removed:

```
scl quotes delete --search-terms David "You can do" Antoine "Perfection is achieved"
```

NOTE

If you're running in the `bin/debug/net6.0` folder, that folder is where you'll find the file with changes from the `add` and `delete` commands. The copy of the file in the project folder remains unchanged.

Next steps

In this tutorial, you created a simple command-line app that uses `System.CommandLine`. To learn more about the library, see [System.CommandLine overview](#).

Command-line syntax overview for System.CommandLine

9/20/2022 • 20 minutes to read • [Edit Online](#)

IMPORTANT

`System.CommandLine` is currently in PREVIEW, and this documentation is for version 2.0 beta 4. Some information relates to prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, express or implied, with respect to the information provided here.

This article explains the command-line syntax that `System.CommandLine` recognizes. The information will be useful to users as well as developers of .NET command-line apps, including the [.NET CLI](#).

Tokens

`System.CommandLine` parses command-line input into *tokens*, which are strings delimited by spaces. For example, consider the following command line:

```
dotnet tool install dotnet-suggest --global --verbosity quiet
```

This input is parsed by the `dotnet` application into tokens `tool`, `install`, `dotnet-suggest`, `--global`, `--verbosity`, and `quiet`.

Tokens are interpreted as commands, options, or arguments. The command-line app that is being invoked determines how the tokens after the first one are interpreted. The following table shows how

`System.CommandLine` interprets the preceding example:

TOKEN	PARSSED AS
<code>tool</code>	Subcommand
<code>install</code>	Subcommand
<code>dotnet-suggest</code>	Argument for install command
<code>--global</code>	Option for install command
<code>--verbosity</code>	Option for install command
<code>quiet</code>	Argument for <code>--verbosity</code> option

A token can contain spaces if it's enclosed in quotation marks (`"`). Here's an example:

```
dotnet tool search "ef migrations add"
```

Commands

A *command* in command-line input is a token that specifies an action or defines a group of related actions. For example:

- In `dotnet run`, `run` is a command that specifies an action.
- In `dotnet tool install`, `install` is a command that specifies an action, and `tool` is a command that specifies a group of related commands. There are other tool-related commands, such as `tool uninstall`, `tool list`, and `tool update`.

Root commands

The *root command* is the one that specifies the name of the app's executable. For example, the `dotnet` command specifies the `dotnet.exe` executable.

Subcommands

Most command-line apps support *subcommands*, also known as *verbs*. For example, the `dotnet` command has a `run` subcommand that you invoke by entering `dotnet run`.

Subcommands can have their own subcommands. In `dotnet tool install`, `install` is a subcommand of `tool`.

Options

An option is a named parameter that can be passed to a command. The [POSIX](#) convention is to prefix the option name with two hyphens (`--`). The following example shows two options:

```
dotnet tool update dotnet-suggest --verbosity quiet --global
      ^-----^          ^-----^
```

As this example illustrates, the value of the option may be explicit (`quiet` for `--verbosity`) or implicit (nothing follows `--global`). Options that have no value specified are typically Boolean parameters that default to `true` if the option is specified on the command line.

For some Windows command-line apps, you identify an option by using a leading slash (`/`) with the option name. For example:

```
msbuild /version
      ^-----^
```

`System.CommandLine` supports both POSIX and Windows prefix conventions. When you [configure an option](#), you specify the option name including the prefix.

Arguments

An argument is a value passed to an option or a command. The following examples show an argument for the `verbosity` option and an argument for the `build` command.

```
dotnet tool update dotnet-suggest --verbosity quiet --global
      ^---^
```

```
dotnet build myapp.csproj
      ^-----^
```

Arguments can have default values that apply if no argument is explicitly provided. For example, many options are implicitly Boolean parameters with a default of `true` when the option name is in the command line. The

following command-line examples are equivalent:

```
dotnet tool update dotnet-suggest --global  
^-----^  
  
dotnet tool update dotnet-suggest --global true  
^-----^
```

Some options have required arguments. For example in the .NET CLI, `--output` requires a folder name argument. If the argument is not provided, the command fails.

Arguments can have expected types, and `System.CommandLine` displays an error message if an argument can't be parsed into the expected type. For example, the following command errors because "silent" isn't one of the valid values for `--verbosity`:

```
dotnet build --verbosity silent
```

```
Cannot parse argument 'silent' for option '-v' as expected type 'Microsoft.DotNet.Cli.VerbosityOptions'. Did you mean one of the following?  
Detailed  
Diagnostic  
Minimal  
Normal  
Quiet
```

Arguments also have expectations about how many values can be provided. Examples are provided in the [section on argument arity](#).

Order of options and arguments

You can provide options before arguments or arguments before options on the command line. The following commands are equivalent:

```
dotnet add package System.CommandLine --prerelease  
dotnet add package --prerelease System.CommandLine
```

Options can be specified in any order. The following commands are equivalent:

```
dotnet add package System.CommandLine --prerelease --no-restore --source https://api.nuget.org/v3/index.json  
dotnet add package System.CommandLine --source https://api.nuget.org/v3/index.json --no-restore --prerelease
```

When there are multiple arguments, the order does matter. The following commands are not necessarily equivalent:

```
myapp argument1 argument2  
myapp argument2 argument1
```

These commands pass a list with the same values to the command handler code, but they differ in the order of the values, which could lead to different results.

Aliases

In both POSIX and Windows, it's common for some commands and options to have aliases. These are usually

short forms that are easier to type. Aliases can also be used for other purposes, such as to [simulate case-insensitivity](#) and to [support alternate spellings of a word](#).

POSIX short forms typically have a single leading hyphen followed by a single character. The following commands are equivalent:

```
dotnet build --verbosity quiet  
dotnet build -v quiet
```

The [GNU standard](#) recommends automatic aliases. That is, you can enter any part of a long-form command or option name and it will be accepted. This behavior would make the following command lines equivalent:

```
dotnet publish --output ./publish  
dotnet publish --outpu ./publish  
dotnet publish --outp ./publish  
dotnet publish --out ./publish  
dotnet publish --ou ./publish  
dotnet publish --o ./publish
```

`System.CommandLine` doesn't support automatic aliases.

Case sensitivity

Command and option names and aliases are case-sensitive by default according to POSIX convention, and `System.CommandLine` follows this convention. If you want your CLI to be case insensitive, define aliases for the various casing alternatives. For example, `--additional-probing-path` could have aliases `--Additional-Probing-Path` and `--ADDITIONAL-PROBING-PATH`.

In some command-line tools, a difference in casing specifies a difference in function. For example, `git clean -X` behaves differently than `git clean -x`. The .NET CLI is all lowercase.

Case sensitivity does not apply to argument values for options that are based on enums. Enum names are matched regardless of casing.

The `--` token

POSIX convention interprets the double-dash (`--`) token as an escape mechanism. Everything that follows the double-dash token is interpreted as arguments for the command. This functionality can be used to submit arguments that look like options, since it prevents them from being interpreted as options.

Suppose `myapp` takes a `message` argument, and you want the value of `message` to be `--interactive`. The following command line might give unexpected results.

```
myapp --interactive
```

If `myapp` doesn't have an `--interactive` option, the `--interactive` token is interpreted as an argument. But if the app does have an `--interactive` option, this input will be interpreted as referring to that option.

The following command line uses the double-dash token to set the value of the `message` argument to "`--interactive`":

```
myapp -- --interactive  
^^
```

`System.CommandLine` supports this double-dash functionality.

Option-argument delimiters

`System.CommandLine` lets you use a space, '=' or ':' as the delimiter between an option name and its argument. For example, the following commands are equivalent:

```
dotnet build -v quiet
dotnet build -v=quiet
dotnet build -v:quiet
```

A POSIX convention lets you omit the delimiter when you are specifying a single-character option alias. For example, the following commands are equivalent:

```
myapp -vquiet
myapp -v quiet
```

`System.CommandLine` supports this syntax by default.

Argument arity

The *arity* of an option or command's argument is the number of values that can be passed if that option or command is specified.

Arity is expressed with a minimum value and a maximum value, as the following table illustrates:

MIN	MAX	EXAMPLE VALIDITY	EXAMPLE
0	0	Valid:	--file
		Invalid:	--file a.json
		Invalid:	--file a.json --file b.json
0	1	Valid:	--flag
		Valid:	--flag true
		Valid:	--flag false
1	1	Invalid:	--flag false --flag false
		Valid:	--file a.json
		Invalid:	--file
0	n	Invalid:	--file a.json --file b.json
		Valid:	--file
		Valid:	--file a.json
		Valid:	--file a.json --file b.json

MIN	MAX	EXAMPLE VALIDITY	EXAMPLE
1	n	Valid:	--file a.json
		Valid:	--file a.json b.json
		Invalid:	--file

`System.CommandLine` has an [ArgumentArity](#) struct for defining arity, with the following values:

- [Zero](#) - No values allowed.
- [ZeroOrOne](#) - May have one value, may have no values.
- [ExactlyOne](#) - Must have one value.
- [ZeroOrMore](#) - May have one value, multiple values, or no values.
- [OneOrMore](#) - May have multiple values, must have at least one value.

Arity can often be inferred from the type. For example, an `int` option has arity of `ExactlyOne`, and a `List<int>` option has arity `OneOrMore`.

Option overrides

If the arity maximum is 1, `System.CommandLine` can still be configured to accept multiple instances of an option. In that case, the last instance of a repeated option overwrites any earlier instances. In the following example, the value 2 would be passed to the `myapp` command.

```
myapp --delay 3 --message example --delay 2
```

Multiple arguments

If the arity maximum is more than one, `System.CommandLine` can be configured to accept multiple arguments for one option without repeating the option name.

In the following example, the list passed to the `myapp` command would contain "a", "b", "c", and "d":

```
myapp --list a b c --list d
```

Option bundling

POSIX recommends that you support *bundling* of single-character options, also known as *stacking*. Bundled options are single-character option aliases specified together after a single hyphen prefix. Only the last option can specify an argument. For example, the following command lines are equivalent:

```
git clean -f -d -x
git clean -fdx
```

If an argument is provided after an option bundle, it applies to the last option in the bundle. The following command lines are equivalent:

```
myapp -a -b -c arg
myapp -abc arg
```

In both variants in this example, the argument `arg` would apply only to the option `-c`.

Boolean options (flags)

If `true` or `false` is passed for an option having a `bool` argument, it's parsed as expected. But an option whose argument type is `bool` typically doesn't require an argument to be specified. Boolean options, sometimes called "flags", typically have an [arity](#) of `ZeroOrOne`. The presence of the option name on the command line, with no argument following it, results in a default value of `true`. The absence of the option name in command-line input results in a value of `false`. If the `myapp` command prints out the value of a Boolean option named `--interactive`, the following input creates the following output:

```
myapp
myapp --interactive
myapp --interactive false
myapp --interactive true
```

```
False
True
False
True
```

The --help option

Command-line apps typically provide an option to display a brief description of the available commands, options, and arguments. `System.CommandLine` automatically generates help output. For example:

```
dotnet list --help
```

```
Description:
List references or packages of a .NET project.

Usage:
dotnet [options] list [<PROJECT | SOLUTION>] [command]

Arguments:
<PROJECT | SOLUTION> The project or solution file to operate on. If a file is not specified, the command will search the current directory for one.

Options:
-?, -h, --help Show command line help.

Commands:
package List all package references of the project or solution.
reference List all project-to-project references of the project.
```

App users might be accustomed to different ways to request help on different platforms, so apps built on `System.CommandLine` respond to many ways of requesting help. The following commands are all equivalent:

```
dotnet --help
dotnet -h
dotnet /h
dotnet -?
dotnet /?
```

Help output doesn't necessarily show all available commands, arguments, and options. Some of them may be *hidden*, which means they don't show up in help output but they can be specified on the command line.

The --version option

Apps built on `System.CommandLine` automatically provide the version number in response to the `--version` option used with the root command. For example:

```
dotnet --version
```

```
6.0.100
```

Response files

A *response file* is a file that contains a set of *tokens* for a command-line app. Response files are a feature of `System.CommandLine` that is useful in two scenarios:

- To invoke a command-line app by specifying input that is longer than the character limit of the terminal.
- To invoke the same command repeatedly without retyping the whole line.

To use a response file, enter the file name prefixed by an `@` sign wherever in the line you want to insert commands, options, and arguments. The `.rsp` file extension is a common convention, but you can use any file extension.

The following lines are equivalent:

```
dotnet build --no-restore --output ./build-output/
dotnet @sample1.rsp
dotnet build @sample2.rsp --output ./build-output/
```

Contents of *sample1.rsp*:

```
build
--no-restore
--output
./build-output/
```

Contents of *sample2.rsp*:

```
--no-restore
```

Here are syntax rules that determine how the text in a response file is interpreted:

- Tokens are delimited by spaces. A line that contains *Good morning!* is treated as two tokens, *Good* and *morning!*.
- Multiple tokens enclosed in quotes are interpreted as a single token. A line that contains "*Good morning!*" is treated as one token, *Good morning!*.
- Any text between a `#` symbol and the end of the line is treated as a comment and ignored.
- Tokens prefixed with `@` can reference additional response files.
- The response file can have multiple lines of text. The lines are concatenated and interpreted as a sequence of tokens.

Directives

`System.CommandLine` introduces a syntactic element called a *directive*. The `[parse]` directive is an example.

When you include `[parse]` after the app's name, `System.CommandLine` displays a diagram of the parse result instead of invoking the command-line app:

```
dotnet [parse] build --no-restore --output ./build-output/  
      ^-----^
```

```
[ dotnet [ build [ --no-restore <True> ] [ --output <./build-output/> ] ] ]
```

The purpose of directives is to provide cross-cutting functionality that can apply across command-line apps. Because directives are syntactically distinct from the app's own syntax, they can provide functionality that applies across apps.

A directive must conform to the following syntax rules:

- It's a token on the command line that comes after the app's name but before any subcommands or options.
- It's enclosed in square brackets.
- It doesn't contain spaces.

An unrecognized directive is ignored without causing a parsing error.

A directive can include an argument, separated from the directive name by a colon.

The following directives are built in:

- `[parse]`
- `[suggest]`

The `[parse]` directive

Both users and developers may find it useful to see how an app will interpret a given input. One of the default features of a `System.CommandLine` app is the `[parse]` directive, which lets you preview the result of parsing command input. For example:

```
myapp [parse] --delay not-an-int --interactive --file filename.txt extra
```

```
![ myapp [ --delay !<not-an-int> ] [ --interactive <True> ] [ --file <filename.txt> ] *[ --fgcolor <White> ]  
] ??--> extra
```

In the preceding example:

- The command (`myapp`), its child options, and the arguments to those options are grouped using square brackets.
- For the option result `[--delay !<not-an-int>]`, the `!` indicates a parsing error. The value `not-an-int` for an `int` option can't be parsed to the expected type. The error is also flagged by `!` in front of the command that contains the errored option: `![myapp...]`.
- For the option result `*[--fgcolor <White>]`, the option wasn't specified on the command line, so the configured default was used. `White` is the effective value for this option. The asterisk indicates that the value is the default.
- `??-->` points to input that wasn't matched to any of the app's commands or options.

The `[suggest]` directive

The `[suggest]` directive lets you search for commands when you don't know the exact command.

```
dotnet [suggest] buil
```

```
build  
build-server  
msbuild
```

Design guidance

The following sections present guidance that we recommend you follow when designing a CLI. Think of what your app expects on the command line as similar to what a REST API server expects in the URL. Consistent rules for REST APIs are what make them readily usable to client app developers. In the same way, users of your command-line apps will have a better experience if the CLI design follows common patterns.

Once you create a CLI it is hard to change, especially if your users have used your CLI in scripts they expect to keep running. The guidelines here were developed after the .NET CLI, and it doesn't always follow these guidelines. We are updating the .NET CLI where we can do it without introducing breaking changes. An example of this work is the new design for `dotnet new` in .NET 7.

Commands and subcommands

If a command has subcommands, the command should function as an area, or a grouping identifier for the subcommands, rather than specify an action. When you invoke the app, you specify the grouping command and one of its subcommands. For example, try to run `dotnet tool`, and you get an error message because the `tool` command only identifies a group of tool-related subcommands, such as `install` and `list`. You can run `dotnet tool install`, but `dotnet tool` by itself would be incomplete.

One of the ways that defining areas helps your users is that it organizes the help output.

Within a CLI there is often an implicit area. For example, in the .NET CLI, the implicit area is the project and in the Docker CLI it is the image. As a result, you can use `dotnet build` without including an area. Consider whether your CLI has an implicit area. If it does, consider whether to allow the user to optionally include or omit it as in `docker build` and `docker image build`. If you optionally allow the implicit area to be typed by your user, you also automatically have help and tab completion for this grouping of commands. Supply the optional use of the implicit group by defining two commands that perform the same operation.

Options as parameters

Options should provide parameters to commands, rather than specifying actions themselves. This is a recommended design principle although it isn't always followed by `System.CommandLine` (`--help` displays help information).

Short-form aliases

In general, we recommend that you minimize the number of short-form option aliases that you define.

In particular, avoid using any of the following aliases differently than their common usage in the .NET CLI and other .NET command-line apps:

- `-i` for `--interactive`.

This option signals to the user that they may be prompted for inputs to questions that the command needs answered. For example, prompting for a username. Your CLI may be used in scripts, so use caution in prompting users that have not specified this switch.

- `-o` for `--output`.

Some commands produce files as the result of their execution. This option should be used to help

determine where those files should be located. In cases where a single file is created, this option should be a file path. In cases where many files are created, this option should be a directory path.

- `-v` for `--verbosity`.

Commands often provide output to the user on the console; this option is used to specify the amount of output the user requests. For more information, see [The `--verbosity` option](#) later in this article.

There are also some aliases with common usage limited to the .NET CLI. You can use these aliases for other options in your apps, but be aware of the possibility of confusion.

- `-c` for `--configuration`

This option often refers to a named Build Configuration, like `Debug` or `Release`. You can use any name you want for a configuration, but most tools are expecting one of those. This setting is often used to configure other properties in a way that makes sense for that configuration—for example, doing less code optimization when building the `Debug` configuration. Consider this option if your command has different modes of operation.

- `-f` for `--framework`

This option is used to select a single [Target Framework Moniker \(TFM\)](#) to execute for, so if your CLI application has differing behavior based on which TFM is chosen, you should support this flag.

- `-p` for `--property`

If your application eventually invokes MSBuild, the user will often need to customize that call in some way. This option allows for MSBuild properties to be provided on the command line and passed on to the underlying MSBuild call. If your app doesn't use MSBuild but needs a set of key-value pairs, consider using this same option name to take advantage of users' expectations.

- `-r` for `--runtime`

If your application can run on different runtimes, or has runtime-specific logic, consider supporting this option as a way of specifying a [Runtime Identifier](#). If your app supports `--runtime`, consider supporting `--os` and `--arch` also. These options let you specify just the OS or the architecture parts of the RID, leaving the part not specified to be determined from the current platform. For more information, see [ddotnet publish](#).

Short names

Make names for commands, options, and arguments as short and easy to spell as possible. For example, if `class` is clear enough don't make the command `classification`.

Lowercase names

Define names in lowercase only, except you can make uppercase aliases to make commands or options case insensitive.

Kebab case names

Use [kebab case](#) to distinguish words. For example, `--additional-probing-path`.

Pluralization

Within an app, be consistent in pluralization. For example, don't mix plural and singular names for options that can have multiple values (maximum arity greater than one):

OPTION NAMES	CONSISTENCY
<code>--additional-probing-paths</code> and <code>--sources</code>	✓

OPTION NAMES	CONSISTENCY
--additional-probing-path and --source	✓
--additional-probing-paths and --source	✗
--additional-probing-path and --sources	✗

Verbs vs. nouns

Use verbs rather than nouns for commands that refer to actions (those without subcommands under them), for example: `dotnet workload remove`, not `dotnet workload removal`. And use nouns rather than verbs for options, for example: `--configuration`, not `--configure`.

The `--verbosity` option

`System.CommandLine` applications typically offer a `--verbosity` option that specifies how much output is sent to the console. Here are the standard five settings:

- `Q[uiet]`
- `M[inimal]`
- `N[ormal]`
- `D[etailed]`
- `Diag[nostic]`

These are the standard names, but existing apps sometimes use `Silent` in place of `Quiet`, and `Trace`, `Debug`, or `Verbose` in place of `Diagnostic`.

Each app defines its own criteria that determine what gets displayed at each level. Typically an app only needs three levels:

- Quiet
- Normal
- Diagnostic

If an app doesn't need five different levels, the option should still define the same five settings. In that case, `Minimal` and `Normal` will produce the same output, and `Detailed` and `Diagnostic` will likewise be the same. This allows your users to just type what they are familiar with, and the best fit will be used.

The expectation for `Quiet` is that no output is displayed on the console. However, if an app offers an interactive mode, the app should do one of the following alternatives:

- Display prompts for input when `--interactive` is specified, even if `--verbosity` is `Quiet`.
- Disallow the use of `--verbosity Quiet` and `--interactive` together.

Otherwise the app will wait for input without telling the user what it's waiting for. It will appear that your application froze and the user will have no idea the application is waiting for input.

If you define aliases, use `-v` for `--verbosity` and make `-v` without an argument an alias for `--verbosity Diagnostic`. Use `-q` for `--verbosity Quiet`.

The .NET CLI and POSIX conventions

The .NET CLI does not consistently follow all POSIX conventions.

Double-dash

Several commands in the .NET CLI have a special implementation of the double-dash token. In the case of `dotnet run`, `dotnet watch`, and `dotnet tool run`, tokens that follow `--` are passed to the app that is being run by the command. For example:

```
dotnet run --project ./myapp.csproj -- --message "Hello world!"  
      ^^
```

In this example, the `--project` option is passed to the `dotnet run` command, and the `--message` option with its argument is passed as a command-line option to *myapp* when it runs.

The `--` token is not always required for passing options to an app that you run by using `dotnet run`. Without the double-dash, the `dotnet run` command automatically passes on to the app being run any options that aren't recognized as applying to `dotnet run` itself or to MSBuild. So the following command lines are equivalent because `dotnet run` doesn't recognize the arguments and options:

```
dotnet run -- quotes read --delay 0 --fg-color red  
dotnet run quotes read --delay 0 --fg-color red
```

Omission of the option-to-argument delimiter

The .NET CLI doesn't support the POSIX convention that lets you omit the delimiter when you are specifying a single-character option alias.

Multiple arguments without repeating the option name

The .NET CLI doesn't accept multiple arguments for one option without repeating the option name.

Boolean options

In the .NET CLI, some Boolean options result in the same behavior when you pass `false` as when you pass `true`. This behavior results when .NET CLI code that implements the option only checks for the presence or absence of the option, ignoring the value. An example is `--no-restore` for the `dotnet build` command. Pass `no-restore false` and the restore operation will be skipped the same as when you specify `no-restore true` or `no-restore`.

Kebab case

In some cases, the .NET CLI doesn't use kebab case for command, option, or argument names. For example, there is a .NET CLI option that is named `--additionalprobingpath` instead of `--additional-probing-path`.

See also

- [Open-source CLI design guidance](#)
- [GNU standards](#)
- [System.CommandLine overview](#)
- [Tutorial: Get started with System.CommandLine](#)

How to define commands, options, and arguments in System.CommandLine

9/20/2022 • 6 minutes to read • [Edit Online](#)

IMPORTANT

`System.CommandLine` is currently in PREVIEW, and this documentation is for version 2.0 beta 4. Some information relates to prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, express or implied, with respect to the information provided here.

This article explains how to define [commands](#), [options](#), and [arguments](#) in command-line apps that are built with the `System.CommandLine` library. To build a complete application that illustrates these techniques, see the tutorial [Get started with System.CommandLine](#).

For guidance on how to design a command-line app's commands, options, and arguments, see [Design guidance](#).

Define a root command

Every command-line app has a [root command](#), which refers to the executable file itself. The simplest case for invoking your code, if you have an app with no subcommands, options, or arguments, would look like this:

```
using System.CommandLine;

class Program
{
    static async Task Main(string[] args)
    {
        var rootCommand = new RootCommand("Sample command-line app");

        rootCommand.SetHandler(() =>
        {
            Console.WriteLine("Hello world!");
        });

        await rootCommand.InvokeAsync(args);
    }
}
```

Define subcommands

Commands can have child commands, known as [subcommands or verbs](#), and they can nest as many levels as you need. You can add subcommands as shown in the following example:

```
var rootCommand = new RootCommand();
var sub1Command = new Command("sub1", "First-level subcommand");
rootCommand.Add(sub1Command);
var sub1aCommand = new Command("sub1a", "Second level subcommand");
sub1Command.Add(sub1aCommand);
```

The innermost subcommand in this example can be invoked like this:

```
myapp sub1 sub1a
```

Define options

A command handler method typically has parameters, and the values can come from command-line [options](#). The following example creates two options and adds them to the root command. The option names include double-hyphen prefixes, in accordance with [POSIX conventions](#). The command handler code displays the values of those options:

```
var delayOption = new Option<int>
    (name: "--delay",
     description: "An option whose argument is parsed as an int.",
     getDefaultValue: () => 42);
var messageOption = new Option<string>
    ("--message", "An option whose argument is parsed as a string.");

var rootCommand = new RootCommand();
rootCommand.Add(delayOption);
rootCommand.Add(messageOption);

rootCommand.SetHandler((delayOptionValue, messageOptionValue) =>
{
    Console.WriteLine($"--delay = {delayOptionValue}");
    Console.WriteLine($"--message = {messageOptionValue}");
},
delayOption, messageOption);
```

Here's an example of command-line input and the resulting output for the preceding example code:

```
myapp --delay 21 --message "Hello world!"
```

```
--delay = 21
--message = Hello world!
```

Global options

To add an option to one command at a time, use the [Add](#) or [AddOption](#) method as shown in the preceding example. To add an option to a command and recursively to all of its subcommands, use the [AddGlobalOption](#) method, as shown in the following example:

```

var delayOption = new Option<int>
    ("--delay", "An option whose argument is parsed as an int.");
var messageOption = new Option<string>
    ("--message", "An option whose argument is parsed as a string.");

var rootCommand = new RootCommand();
rootCommand.AddGlobalOption(delayOption);
rootCommand.Add(messageOption);

var subCommand1 = new Command("sub1", "First level subcommand");
rootCommand.Add(subCommand1);

var subCommand1a = new Command("sub1a", "Second level subcommand");
subCommand1.Add(subCommand1a);

subCommand1a.SetHandler((delayOptionValue) =>
{
    Console.WriteLine($"--delay = {delayOptionValue}");
},
delayOption);

await rootCommand.InvokeAsync(args);

```

The preceding code adds `--delay` as a global option to the root command, and it's available in the handler for `subCommand1a`.

Define arguments

[Arguments](#) are defined and added to commands like options. The following example is like the options example, but it defines arguments instead of options:

```

var delayArgument = new Argument<int>
    (name: "delay",
     description: "An argument that is parsed as an int.",
     getDefaultValue: () => 42);
var messageArgument = new Argument<string>
    ("message", "An argument that is parsed as a string.");

var rootCommand = new RootCommand();
rootCommand.Add(delayArgument);
rootCommand.Add(messageArgument);

rootCommand.SetHandler((delayArgumentValue, messageArgumentValue) =>
{
    Console.WriteLine($"<delay> argument = {delayArgumentValue}");
    Console.WriteLine($"<message> argument = {messageArgumentValue}");
},
delayArgument, messageArgument);

await rootCommand.InvokeAsync(args);

```

Here's an example of command-line input and the resulting output for the preceding example code:

```
myapp 42 "Hello world!"
```

```
<delay> argument = 42
<message> argument = Hello world!
```

An argument that is defined without a default value, such as `messageArgument` in the preceding example, is

treated as a required argument. An error message is displayed, and the command handler isn't called, if a required argument isn't provided.

Define aliases

Both commands and options support [aliases](#). You can add an alias to an option by calling `AddAlias`:

```
var option = new Option("--framework");
option.AddAlias("-f");
```

Given this alias, the following command lines are equivalent:

```
myapp -f net6.0
myapp --framework net6.0
```

Command aliases work the same way.

```
var command = new Command("serialize");
command.AddAlias("serialise");
```

This code makes the following command lines equivalent:

```
myapp serialize
myapp serialise
```

We recommend that you minimize the number of option aliases that you define, and avoid defining certain aliases in particular. For more information, see [Short-form aliases](#).

Required options

To make an option required, set its `IsRequired` property to `true`, as shown in the following example:

```
var endpointOption = new Option<Uri>("--endpoint") { IsRequired = true };
var command = new RootCommand();
command.Add(endpointOption);

command.SetHandler((uri) =>
{
    Console.WriteLine(uri?.GetType());
    Console.WriteLine(uri?.ToString());
},
endpointOption);

await command.InvokeAsync(args);
```

The options section of the command help indicates the option is required:

```
Options:
--endpoint <uri> (REQUIRED)
--version           Show version information
-?, -h, --help      Show help and usage information
```

If the command line for this example app doesn't include `--endpoint`, an error message is displayed and the command handler isn't called:

```
Option '--endpoint' is required.
```

If a required option has a default value, the option doesn't have to be specified on the command line. In that case, the default value provides the required option value.

Hidden commands, options, and arguments

You might want to support a command, option, or argument, but avoid making it easy to discover. For example, it might be a deprecated or administrative or preview feature. Use the `IsHidden` property to prevent users from discovering such features by using tab completion or help, as shown in the following example:

```
var endpointOption = new Option<Uri>("--endpoint") { IsHidden = true };
var command = new RootCommand();
command.Add(endpointOption);

command.SetHandler((uri) =>
{
    Console.WriteLine(uri?.GetType());
    Console.WriteLine(uri?.ToString());
},
endpointOption);

await command.InvokeAsync(args);
```

The options section of this example's command help omits the `--endpoint` option.

```
Options:
--version           Show version information
-?, -h, --help     Show help and usage information
```

Set argument arity

You can explicitly set argument `arity` by using the `Arity` property, but in most cases that is not necessary.

`System.CommandLine` automatically determines the argument arity based on the argument type:

ARGUMENT TYPE	DEFAULT ARITY
<code>Boolean</code>	<code>ArgumentArity.ZeroOrOne</code>
Collection types	<code>ArgumentArity.ZeroOrMore</code>
Everything else	<code>ArgumentArity.ExactlyOne</code>

Multiple arguments

By default, when you call a command, you can repeat an option name to specify multiple arguments for an option that has maximum `arity` greater than one.

```
myapp --items one --items two --items three
```

To allow multiple arguments without repeating the option name, set `Option.AllowMultipleArgumentsPerToken` to `true`. This setting lets you enter the following command line.

```
myapp --items one two three
```

The same setting has a different effect if maximum argument arity is 1. It allows you to repeat an option but takes only the last value on the line. In the following example, the value `three` would be passed to the app.

```
myapp --item one --item two --item three
```

List valid argument values

To specify a list of valid values for an option or argument, specify an enum as the option type or use [FromAmong](#), as shown in the following example:

```
var languageOption = new Option<string>(
    "--language",
    "An option that must be one of the values of a static list.")
    .FromAmong(
        "csharp",
        "fsharp",
        "vb",
        "pwsh",
        "sql");
```

Here's an example of command-line input and the resulting output for the preceding example code:

```
myapp --language not-a-language
```

```
Argument 'not-a-language' not recognized. Must be one of:
  'csharp'
  'fsharp'
  'vb'
  'pwsh'
  'sql'
```

The options section of command help shows the valid values:

```
Options:
--language <csharp|fsharp|vb|pwsh|sql> An option that must be one of the values of a static list.
--version                                         Show version information
-?, -h, --help                                     Show help and usage information
```

Option and argument validation

For information about argument validation and how to customize it, see the following sections in the [Parameter binding](#) article:

- [Built-in type and arity argument validation](#)
- [Custom validation and binding](#)

See also

- [System.CommandLine overview](#)
- [Parameter binding](#)

How to bind arguments to handlers in System.CommandLine

9/20/2022 • 10 minutes to read • [Edit Online](#)

IMPORTANT

`System.CommandLine` is currently in PREVIEW, and this documentation is for version 2.0 beta 4. Some information relates to prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, express or implied, with respect to the information provided here.

The process of parsing arguments and providing them to command handler code is called *parameter binding*. `System.CommandLine` has the ability to bind many argument types built in. For example, integers, enums, and file system objects such as `FileInfo` and `DirectoryInfo` can be bound. Several `System.CommandLine` types can also be bound.

Built-in argument validation

Arguments have expected types and [arity](#). `System.CommandLine` rejects arguments that don't match these expectations.

For example, a parse error is displayed if the argument for an integer option isn't an integer.

```
myapp --delay not-an-int
```

```
Cannot parse argument 'not-an-int' as System.Int32.
```

An arity error is displayed if multiple arguments are passed to an option that has maximum arity of one:

```
myapp --delay-option 1 --delay-option 2
```

```
Option '--delay' expects a single argument but 2 were provided.
```

This behavior can be overridden by setting `Option.AllowMultipleArgumentsPerToken` to `true`. In that case you can repeat an option that has maximum arity of one, but only the last value on the line is accepted. In the following example, the value `three` would be passed to the app.

```
myapp --item one --item two --item three
```

Parameter binding up to 16 options and arguments

The following example shows how to bind options to command handler parameters, by calling `SetHandler`:

```

var delayOption = new Option<int>
    ("--delay", "An option whose argument is parsed as an int.");
var messageOption = new Option<string>
    ("--message", "An option whose argument is parsed as a string.");

var rootCommand = new RootCommand("Parameter binding example");
rootCommand.Add(delayOption);
rootCommand.Add(messageOption);

rootCommand.SetHandler(
    (delayOptionValue, messageOptionValue) =>
{
    DisplayIntAndString(delayOptionValue, messageOptionValue);
},
delayOption, messageOption);

await rootCommand.InvokeAsync(args);

```

```

public static void DisplayIntAndString(int delayOptionValue, string messageOptionValue)
{
    Console.WriteLine($"--delay = {delayOptionValue}");
    Console.WriteLine($"--message = {messageOptionValue}");
}

```

The lambda parameters are variables that represent the values of options and arguments:

```

(delayOptionValue, messageOptionValue) =>
{
    DisplayIntAndString(delayOptionValue, messageOptionValue);
},

```

The variables that follow the lambda represent the option and argument objects that are the sources of the option and argument values:

```

delayOption, messageOption);

```

The options and arguments must be declared in the same order in the lambda and in the parameters that follow the lambda. If the order is not consistent, one of the following scenarios will result:

- If the out-of-order options or arguments are of different types, a run-time exception is thrown. For example, an `int` might appear where a `string` should be in the list of sources.
- If the out-of-order options or arguments are of the same type, the handler silently gets the wrong values in the parameters provided to it. For example, `string` option `x` might appear where `string` option `y` should be in the list of sources. In that case, the variable for the option `y` value gets the option `x` value.

There are overloads of `SetHandler` that support up to 8 parameters, with both synchronous and asynchronous signatures.

Parameter binding more than 8 options and arguments

To handle more than 8 options, or to construct a custom type from multiple options, you can use `InvocationContext` or a custom binder.

Use `InvocationContext`

A `SetHandler` overload provides access to the `InvocationContext` object, and you can use `InvocationContext` to get any number of option and argument values. For examples, see [Set exit codes](#) and [Handle termination](#).

Use a custom binder

A custom binder lets you combine multiple option or argument values into a complex type and pass that into a single handler parameter. Suppose you have a `Person` type:

```
public class Person
{
    public string? FirstName { get; set; }
    public string? LastName { get; set; }
}
```

Create a class derived from `BinderBase<T>`, where `T` is the type to construct based on command line input:

```
public class PersonBinder : BinderBase<Person>
{
    private readonly Option<string> _firstNameOption;
    private readonly Option<string> _lastNameOption;

    public PersonBinder(Option<string> firstNameOption, Option<string> lastNameOption)
    {
        _firstNameOption = firstNameOption;
        _lastNameOption = lastNameOption;
    }

    protected override Person GetBoundValue(BindingContext bindingContext) =>
        new Person
    {
        FirstName = bindingContext.ParseResult.GetValueForOption(_firstNameOption),
        LastName = bindingContext.ParseResult.GetValueForOption(_lastNameOption)
    };
}
```

With the custom binder, you can get your custom type passed to your handler the same way you get values for options and arguments:

```
rootCommand.SetHandler((fileOptionValue, person) =>
{
    DoRootCommand(fileOptionValue, person);
},
fileOption, new PersonBinder(firstNameOption, lastNameOption));
```

Here's the complete program that the preceding examples are taken from:

```

using System.CommandLine;
using System.CommandLine.Binding;

public class Program
{
    internal static async Task Main(string[] args)
    {
        var fileOption = new Option<FileInfo?>(
            name: "--file",
            description: "An option whose argument is parsed as a FileInfo",
            getDefaultValue: () => new FileInfo("scl.runtimeconfig.json"));

        var firstNameOption = new Option<string>(
            name: "--first-name",
            description: "Person.FirstName");

        var lastNameOption = new Option<string>(
            name: "--last-name",
            description: "Person.LastName");

        var rootCommand = new RootCommand();
        rootCommand.Add(fileOption);
        rootCommand.Add(firstNameOption);
        rootCommand.Add(lastNameOption);

        rootCommand.SetHandler((fileOptionValue, person) =>
        {
            DoRootCommand(fileOptionValue, person);
        },
        fileOption, new PersonBinder(firstNameOption, lastNameOption));

        await rootCommand.InvokeAsync(args);
    }
}

public static void DoRootCommand(FileInfo? aFile, Person aPerson)
{
    Console.WriteLine($"File = {aFile?.FullName}");
    Console.WriteLine($"Person = {aPerson?.FirstName} {aPerson?.LastName}");
}

public class Person
{
    public string? FirstName { get; set; }
    public string? LastName { get; set; }
}

public class PersonBinder : BinderBase<Person>
{
    private readonly Option<string> _firstNameOption;
    private readonly Option<string> _lastNameOption;

    public PersonBinder(Option<string> firstNameOption, Option<string> lastNameOption)
    {
        _firstNameOption = firstNameOption;
        _lastNameOption = lastNameOption;
    }

    protected override Person GetBoundValue(BindingContext bindingContext) =>
        new Person
    {
        FirstName = bindingContext.ParseResult.GetValueForOption(_firstNameOption),
        LastName = bindingContext.ParseResult.GetValueForOption(_lastNameOption)
    };
}
}

```

Set exit codes

There are [Task](#)-returning [Func](#) overloads of [SetHandler](#). If your handler is called from async code, you can return a [Task<int>](#) from a handler that uses one of these, and use the [int](#) value to set the process exit code, as in the following example:

```
static async Task<int> Main(string[] args)
{
    var delayOption = new Option<int>("--delay");
    var messageOption = new Option<string>("--message");

    var rootCommand = new RootCommand("Parameter binding example");
    rootCommand.Add(delayOption);
    rootCommand.Add(messageOption);

    rootCommand.SetHandler((delayOptionValue, messageOptionValue) =>
    {
        Console.WriteLine($"--delay = {delayOptionValue}");
        Console.WriteLine($"--message = {messageOptionValue}");
        return Task.FromResult(100);
    },
    delayOption, messageOption);

    return await rootCommand.InvokeAsync(args);
}
```

However, if the lambda itself needs to be async, you can't return a [Task<int>](#). In that case, use [InvocationContext.ExitCode](#). You can get the [InvocationContext](#) instance injected into your lambda by using a [SetHandler](#) overload that specifies the [InvocationContext](#) as the sole parameter. This [SetHandler](#) overload doesn't let you specify [IValueDescriptor<T>](#) objects, but you can get option and argument values from the [ParseResult](#) property of [InvocationContext](#), as shown in the following example:

```
static async Task<int> Main(string[] args)
{
    var delayOption = new Option<int>("--delay");
    var messageOption = new Option<string>("--message");

    var rootCommand = new RootCommand("Parameter binding example");
    rootCommand.Add(delayOption);
    rootCommand.Add(messageOption);

    rootCommand.SetHandler(async (context) =>
    {
        int delayOptionValue = context.ParseResult.GetValueForOption(delayOption);
        string? messageOptionValue = context.ParseResult.GetValueForOption(messageOption);

        Console.WriteLine($"--delay = {delayOptionValue}");
        await Task.Delay(delayOptionValue);
        Console.WriteLine($"--message = {messageOptionValue}");
        context.ExitCode = 100;
    });

    return await rootCommand.InvokeAsync(args);
}
```

If you don't have asynchronous work to do, you can use the [Action](#) overloads. In that case, set the exit code by using [InvocationContext.ExitCode](#) the same way you would with an async lambda.

The exit code defaults to 1. If you don't set it explicitly, its value is set to 0 when your handler exits normally. If an exception is thrown, it keeps the default value.

Supported types

The following examples show code that binds some commonly used types.

Enums

The values of `enum` types are bound by name, and the binding is case insensitive:

```
var colorOption = new Option<ConsoleColor>("--color");

var rootCommand = new RootCommand("Enum binding example");
rootCommand.Add(colorOption);

rootCommand.SetHandler((colorOptionValue) =>
    { Console.WriteLine(colorOptionValue); },
    colorOption);

await rootCommand.InvokeAsync(args);
```

Here's sample command-line input and resulting output from the preceding example:

```
myapp --color red
myapp --color RED
```

```
Red
Red
```

Arrays and lists

Many common types that implement `IEnumerable` are supported. For example:

```
var itemsOption = new Option<IEnumerable<string>>("--items")
    { AllowMultipleArgumentsPerToken = true };

var command = new RootCommand("IEnumerable binding example");
command.Add(itemsOption);

command.SetHandler((items) =>
{
    Console.WriteLine(items.GetType());

    foreach (string item in items)
    {
        Console.WriteLine(item);
    }
},
itemsOption);

await command.InvokeAsync(args);
```

Here's sample command-line input and resulting output from the preceding example:

```
--items one --items two --items three
```

```
System.Collections.Generic.List`1[System.String]
one
two
three
```

Because `AllowMultipleArgumentsPerToken` is set to `true`, the following input results in the same output:

```
--items one two three
```

File system types

Command-line applications that work with the file system can use the `FileSystemInfo`, `FileInfo`, and `DirectoryInfo` types. The following example shows the use of `FileSystemInfo`:

```
var fileOrDirectoryOption = new Option<FileSystemInfo>("--file-or-directory");

var command = new RootCommand();
command.Add(fileOrDirectoryOption);

command.SetHandler((fileSystemInfo) =>
{
    switch (fileSystemInfo)
    {
        case FileInfo file:
            Console.WriteLine($"File name: {file.FullName}");
            break;
        case DirectoryInfo directory:
            Console.WriteLine($"Directory name: {directory.FullName}");
            break;
        default:
            Console.WriteLine("Not a valid file or directory name.");
            break;
    }
},
fileOrDirectoryOption);

await command.InvokeAsync(args);
```

With `FileInfo` and `DirectoryInfo` the pattern matching code is not required:

```
var fileOption = new Option<FileInfo>("--file");

var command = new RootCommand();
command.Add(fileOption);

command.SetHandler((file) =>
{
    if (file is not null)
    {
        Console.WriteLine($"File name: {file?.FullName}");
    }
    else
    {
        Console.WriteLine("Not a valid file name.");
    }
},
fileOption);

await command.InvokeAsync(args);
```

Other supported types

Many types that have a constructor that takes a single string parameter can be bound in this way. For example, code that would work with `FileInfo` works with a `Uri` instead.

```
var endpointOption = new Option<Uri>("--endpoint");

var command = new RootCommand();
command.Add(endpointOption);

command.SetHandler((uri) =>
{
    Console.WriteLine($"URL: {uri?.ToString()}");
},
endpointOption);

await command.InvokeAsync(args);
```

Besides the file system types and `Uri`, the following types are supported:

- `bool`
- `byte`
- `DateTime`
- `DateTimeOffset`
- `decimal`
- `double`
- `float`
- `Guid`
- `int`
- `long`
- `sbyte`
- `short`
- `uint`
- `ulong`
- `ushort`

Use `System.CommandLine` objects

There's a `SetHandler` overload that gives you access to the `InvocationContext` object. That object can then be used to access other `System.CommandLine` objects. For example, you have access to the following objects:

- `InvocationContext`
- `CancellationToken`
- `IConsole`
- `ParseResult`

`InvocationContext`

For examples, see [Set exit codes](#) and [Handle termination](#).

`CancellationToken`

For information about how to use `CancellationToken`, see [How to handle termination](#).

`IConsole`

`IConsole` makes testing as well as many extensibility scenarios easier than using `System.Console`. It's available in the `InvocationContext.Console` property.

`ParseResult`

The `ParseResult` object is available in the `InvocationContext.ParseResult` property. It's a singleton structure that represents the results of parsing the command line input. You can use it to check for the presence of options or arguments on the command line or to get the `ParseResult.UnmatchedTokens` property. This property contains a list of the `tokens` that were parsed but didn't match any configured command, option, or argument.

The list of unmatched tokens is useful in commands that behave like wrappers. A wrapper command takes a set of `tokens` and forwards them to another command or app. The `sudo` command in Linux is an example. It takes the name of a user to impersonate followed by a command to run. For example:

```
sudo -u admin apt update
```

This command line would run the `apt update` command as the user `admin`.

To implement a wrapper command like this one, set the command property `TreatUnmatchedTokensAsErrors` to `false`. Then the `ParseResult.UnmatchedTokens` property will contain all of the arguments that don't explicitly belong to the command. In the preceding example, `ParseResult.UnmatchedTokens` would contain the `apt` and `update` tokens. Your command handler could then forward the `UnmatchedTokens` to a new shell invocation, for example.

Custom validation and binding

To provide custom validation code, call `AddValidator` on your command, option, or argument, as shown in the following example:

```
var delayOption = new Option<int>("--delay");
delayOption.AddValidator(result =>
{
    if (result.GetValueForOption(delayOption) < 1)
    {
        result.ErrorMessage = "Must be greater than 0";
    }
});
```

If you want to parse as well as validate the input, use a `ParseArgument<T>` delegate, as shown in the following example:

```

var delayOption = new Option<int>(
    name: "--delay",
    description: "An option whose argument is parsed as an int.",
    isDefault: true,
    parseArgument: result =>
{
    if (!result.Tokens.Any())
    {
        return 42;
    }

    if (int.TryParse(result.Tokens.Single().Value, out var delay))
    {
        if (delay < 1)
        {
            result.ErrorMessage = "Must be greater than 0";
        }
        return delay;
    }
    else
    {
        result.ErrorMessage = "Not an int.";
        return 0; // Ignored.
    }
});

```

The preceding code sets `isDefault` to `true` so that the `parseArgument` delegate will be called even if the user didn't enter a value for this option.

Here are some examples of what you can do with `ParseArgument<T>` that you can't do with `AddValidator`:

- Parsing of custom types, such as the `Person` class in the following example:

```

public class Person
{
    public string? FirstName { get; set; }
    public string? LastName { get; set; }
}

```

```

var personOption = new Option<Person?>(
    name: "--person",
    description: "An option whose argument is parsed as a Person",
    parseArgument: result =>
{
    if (result.Tokens.Count != 2)
    {
        result.ErrorMessage = "--person requires two arguments";
        return null;
    }
    return new Person
    {
        FirstName = result.Tokens.First().Value,
        LastName = result.Tokens.Last().Value
    };
})
{
    Arity = ArgumentArity.OneOrMore,
    AllowMultipleArgumentsPerToken = true
};

```

- Parsing of other kinds of input strings (for example, parse "1,2,3" into `int[]`).

- Dynamic arity. For example, you have two arguments that are defined as string arrays, and you have to handle a sequence of strings in the command line input. The [ArgumentResult.OnlyTake](#) method enables you to dynamically divide up the input strings between the arguments.

See also

[System.CommandLine overview](#)

Tab completion for System.CommandLine

9/20/2022 • 3 minutes to read • [Edit Online](#)

IMPORTANT

`System.CommandLine` is currently in PREVIEW, and this documentation is for version 2.0 beta 4. Some information relates to prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, express or implied, with respect to the information provided here.

Apps that use `System.CommandLine` have built-in support for tab completion in certain shells. To enable it, the end user has to take a few steps once per shell. Once the user does this, tab completion is automatic for static values in your app, such as enum values or values you define by calling [FromAmong](#). You can also customize the tab completion by getting values dynamically at runtime.

Enable tab completion

On the machine where you'd like to enable tab completion, do the following steps.

For the .NET CLI:

- See [How to enable tab completion](#).

For other command-line apps built on `System.CommandLine`:

- Install the `dotnet-suggest` global tool.
- Add the appropriate shim script to your shell profile. You may have to create a shell profile file. The shim script forwards completion requests from your shell to the `dotnet-suggest` tool, which delegates to the appropriate `System.CommandLine`-based app.
 - For `bash`, add the contents of `dotnet-suggest-shim.bash` to `~/.bash_profile`.
 - For `zsh`, add the contents of `dotnet-suggest-shim.zsh` to `~/.zshrc`.
 - For PowerShell, add the contents of `dotnet-suggest-shim.ps1` to your PowerShell profile. You can find the expected path to your PowerShell profile by running the following command in your console:

```
echo $profile
```

Once the user's shell is set up, completions will work for all apps that are built by using `System.CommandLine`.

For `cmd.exe` on Windows (the Windows Command Prompt) there is no pluggable tab completion mechanism, so no shim script is available. For other shells, [look for a GitHub issue that is labeled "Area-Completions"](#). If you don't find an issue, you can [open a new one](#).

Get tab completion values at run-time

The following code shows an app that gets values for tab completion dynamically at runtime. The code gets a list of the next two weeks of dates following the current date. The list is provided to the `--date` option by calling `AddCompletions`:

```

using System.CommandLine;
using System.CommandLine.Completions;
using System.CommandLine.Parsing;

await new DateCommand().InvokeAsync(args);

class DateCommand : Command
{
    private Argument<string> subjectArgument =
        new ("subject", "The subject of the appointment.");
    private Option<DateTime> dateOption =
        new ("--date", "The day of week to schedule. Should be within one week.");

    public DateCommand() : base("schedule", "Makes an appointment for sometime in the next week.")
    {
        this.AddArgument(subjectArgument);
        this.AddOption(dateOption);

        dateOption.AddCompletions((ctx) => {
            var today = System.DateTime.Today;
            var dates = new List<CompletionItem>();
            foreach (var i in Enumerable.Range(1, 7))
            {
                var date = today.AddDays(i);
                dates.Add(new CompletionItem(
                    label: date.ToShortDateString(),
                    sortText: $"{i:2}"));
            }
            return dates;
        });

        this.SetHandler((subject, date) =>
        {
            Console.WriteLine($"Scheduled \\"{subject}\\" for {date}");
        },
        subjectArgument, dateOption);
    }
}

```

The values shown when the tab key is pressed are provided as `CompletionItem` instances:

```

dates.Add(new CompletionItem(
    label: date.ToShortDateString(),
    sortText: $"{i:2}"));

```

The following `CompletionItem` properties are set:

- `Label` is the completion value to be shown.
- `SortText` ensures that the values in the list are presented in the right order. It's set by converting `i` to a two-digit string, so that sorting is based on 01, 02, 03, and so on, through 14. If you don't set this parameter, sorting is based on the `Label`, which in this example is in short date format and won't sort correctly.

There are other `CompletionItem` properties, such as `Documentation` and `Detail`, but they aren't used yet in `System.CommandLine`.

The dynamic tab completion list created by this code also appears in help output:

Description:

Makes an appointment for sometime in the next week.

Usage:

schedule <subject> [options]

Arguments:

<subject> The subject of the appointment.

Options:

--date

The day of week to

schedule. Should be within one week.

<2/4/2022|2/5/2022|2/6/2022|2/7/2022|2/8/2022|2/9/2022|2/10/2022>

--version

Show version information

-?, -h, --help

See also

[System.CommandLine overview](#)

How to configure dependency injection in System.CommandLine

9/20/2022 • 2 minutes to read • [Edit Online](#)

IMPORTANT

`System.CommandLine` is currently in PREVIEW, and this documentation is for version 2.0 beta 4. Some information relates to prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, express or implied, with respect to the information provided here.

Use a [custom binder](#) to inject custom types into a command handler.

We recommend handler-specific dependency injection (DI) for the following reasons:

- Command-line apps are often short-lived processes, in which startup cost can have a noticeable impact on performance. Optimizing performance is particularly important when tab completions have to be calculated. Command-line apps are unlike Web and GUI apps, which tend to be relatively long-lived processes. Unnecessary startup time is not appropriate for short-lived processes.
- When a command-line app that has multiple subcommands is run, only one of those subcommands will be executed. If an app configures dependencies for the subcommands that don't run, it needlessly degrades performance.

To configure DI, create a class that derives from `BinderBase<T>` where `T` is the interface that you want to inject an instance for. In the `GetBoundValue` method override, get and return the instance you want to inject. The following example injects the default logger implementation for `ILogger`:

```
public class MyCustomBinder : BinderBase<ILogger>
{
    protected override ILogger GetBoundValue(
        BindingContext bindingContext) => GetLogger(bindingContext);

    ILogger GetLogger(BindingContext bindingContext)
    {
        ILoggerFactory loggerFactory = LoggerFactory.Create(
            builder => builder.AddConsole());
        ILogger logger = loggerFactory.CreateLogger("LoggerCategory");
        return logger;
    }
}
```

When calling the `SetHandler` method, pass to the lambda an instance of the injected class and pass an instance of your binder class in the list of services:

```
rootCommand.SetHandler(async (fileOptionValue, logger) =>
{
    await DoRootCommand(fileOptionValue!, logger);
},
fileOption, new MyCustomBinder());
```

The following code is a complete program that contains the preceding examples:

```
using System.CommandLine;
using System.CommandLine.Binding;
using Microsoft.Extensions.Logging;

class Program
{
    static async Task Main(string[] args)
    {
        var fileOption = new Option<FileInfo?>(
            name: "--file",
            description: "An option whose argument is parsed as a FileInfo");

        var rootCommand = new RootCommand("Dependency Injection sample");
        rootCommand.Add(fileOption);

        rootCommand.SetHandler(async (fileOptionValue, logger) =>
        {
            await DoRootCommand(fileOptionValue!, logger);
        },
        fileOption, new MyCustomBinder());

        await rootCommand.InvokeAsync("--file scl.runtimeconfig.json");
    }

    public static async Task DoRootCommand(FileInfo aFile, ILogger logger)
    {
        Console.WriteLine($"File = {aFile?.FullName}");
        logger.LogCritical("Test message");
        await Task.Delay(1000);
    }

    public class MyCustomBinder : BinderBase<ILogger>
    {
        protected override ILogger GetBoundValue(
            BindingContext bindingContext) => GetLogger(bindingContext);

        ILogger GetLogger(BindingContext bindingContext)
        {
            ILoggerFactory loggerFactory = LoggerFactory.Create(
                builder => builder.AddConsole());
            ILogger logger = loggerFactory.CreateLogger("LoggerCategory");
            return logger;
        }
    }
}
```

See also

[System.CommandLine overview](#)

How to customize help in apps that are built with the System.Commandline library

9/20/2022 • 4 minutes to read • [Edit Online](#)

You can customize help for a specific command, option, or argument, and you can add or replace whole help sections.

The examples in this article work with the following command-line application:

This code requires a `using` directive:

```
using System.CommandLine;
```

```
var fileOption = new Option<FileInfo>(
    "--file",
    description: "The file to print out.",
    getDefaultValue: () => new FileInfo("scl.runtimeconfig.json"));
var lightModeOption = new Option<bool> (
    "--light-mode",
    description: "Determines whether the background color will be black or white");
var foregroundColorOption = new Option<ConsoleColor>(
    "--color",
    description: "Specifies the foreground color of console output",
    getDefaultValue: () => ConsoleColor.White);

var rootCommand = new RootCommand("Read a file")
{
    fileOption,
    lightModeOption,
    foregroundColorOption
};

rootCommand.SetHandler((file, lightMode, color) =>
{
    Console.BackgroundColor = lightMode ? ConsoleColor.White : ConsoleColor.Black;
    Console.ForegroundColor = color;
    Console.WriteLine($"--file = {file?.FullName}");
    Console.WriteLine($"File contents:\n{file?.OpenText().ReadToEnd()}");
},
    fileOption,
    lightModeOption,
    foregroundColorOption);

await rootCommand.InvokeAsync(args);
```

Without customization, the following help output is produced:

```

Description:
  Read a file

Usage:
  scl [options]

Options:
  --file <file>                               The file to print out. [default:
  scl.runtimeconfig.json]                         Determines whether the background color will
  --light-mode                                     be black or
                                                 white
  be black or                                     Specifies the foreground color of console
  --color                                         output
  <Black|Blue|Cyan|DarkBlue|DarkCyan|DarkGray|DarkGreen|Dark [default: White]
  Magenta|DarkRed|DarkYellow|Gray|Green|Magenta|Red|White|Ye
  llow>
  --version                                       Show version information
  -?, -h, --help                                    Show help and usage information

```

Customize help for a single option or argument

IMPORTANT

`System.CommandLine` is currently in PREVIEW, and this documentation is for version 2.0 beta 4. Some information relates to prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, express or implied, with respect to the information provided here.

To customize the name of an option's argument, use the option's [ArgumentHelpName](#) property. And [HelpBuilder.CustomizeSymbol](#) lets you customize several parts of the help output for a command, option, or argument ([Symbol](#) is the base class for all three types). With `CustomizeSymbol`, you can specify:

- The first column text.
- The second column text.
- The way a default value is described.

In the sample app, `--light-mode` is explained adequately, but changes to the `--file` and `--color` option descriptions will be helpful. For `--file`, the argument can be identified as a `<FILEPATH>` instead of `<file>`. For the `--color` option, you can shorten the list of available colors in column one, and in column two you can add a warning that some colors won't work with some backgrounds.

To make these changes, delete the `await rootCommand.InvokeAsync(args);` line shown in the preceding code and add in its place the following code:

```

fileOption.ArgumentHelpName = "FILEPATH";

var parser = new CommandLineBuilder(rootCommand)
    .UseDefaults()
    .UseHelp(ctx =>
{
    ctx.HelpBuilder.CustomizeSymbol(foregroundColorOption,
        firstColumnText: "--color <Black, White, Red, or Yellow>",
        secondColumnText: "Specifies the foreground color. " +
            "Choose a color that provides enough contrast " +
            "with the background color. " +
            "For example, a yellow foreground can't be read " +
            "against a light mode background.");
})
.Build();

parser.Invoke(args);

```

The updated code requires additional `using` directives:

```

using System.CommandLine.Builder;
using System.CommandLine.Help;
using System.CommandLine.Parsing;

```

The app now produces the following help output:

```

Description:
  Read a file

Usage:
  scl [options]

Options:
  --file <FILEPATH>          The file to print out. [default: CustomHelp.runtimeconfig.json]
  --light-mode                Determines whether the background color will be black or white
  --color <Black, White, Red, or Yellow> Specifies the foreground color. Choose a color that provides
enough contrast
                                         with the background color. For example, a yellow foreground can't
be read
                                         against a light mode background.
  --version                   Show version information
  -?, -h, --help              Show help and usage information

```

This output shows that the `firstColumnText` and `secondColumnText` parameters support word wrapping within their columns.

Add or replace help sections

You can add or replace a whole section of the help output. For example, suppose you want to add some ASCII art to the description section by using the [Spectre.Console](#) NuGet package.

Change the layout by adding a call to `HelpBuilder.CustomizeLayout` in the lambda passed to the `UseHelp` method:

```

fileOption.ArgumentHelpName = "FILEPATH";

var parser = new CommandLineBuilder(rootCommand)
    .UseDefaults()
    .UseHelp(ctx =>
{
    ctx.HelpBuilder.CustomizeSymbol(foregroundColorOption,
        firstColumnText: "--color <Black, White, Red, or Yellow>",
        secondColumnText: "Specifies the foreground color. " +
            "Choose a color that provides enough contrast " +
            "with the background color. " +
            "For example, a yellow foreground can't be read " +
            "against a light mode background.");
    ctx.HelpBuilder.CustomizeLayout(
        _ =>
            HelpBuilder.Default
                .GetLayout()
                .Skip(1) // Skip the default command description section.
                .Prepend(
                    _ => Spectre.Console.AnsiConsole.Write(
                        new FigletText(rootCommand.Description!)))
    );
})
.Build();

await parser.InvokeAsync(args);

```

The preceding code requires an additional `using` directive:

```
using Spectre.Console;
```

The [System.CommandLine.Help.HelpBuilder.Default](#) class lets you reuse pieces of existing help formatting functionality and compose them into your custom help.

The help output now looks like this:



```

Usage:
scl [options]

Options:
--file <FILEPATH>                               The file to print out. [default: CustomHelp.runtimeconfig.json]
--light-mode                                      Determines whether the background color will be black or white
--color <Black, White, Red, or Yellow>           Specifies the foreground color. Choose a color that provides
enough contrast                                     with the background color. For example, a yellow foreground can't
be read                                            against a light mode background.
--version                                           Show version information
-?, -h, --help                                       Show help and usage information

```

If you want to just use a string as the replacement section text instead of formatting it with `Spectre.Console`, replace the `Prepend` code in the preceding example with the following code:

```
.Prepend(  
    _ => _.Output.WriteLine("**New command description section**")
```

See also

[System.CommandLine overview](#)

How to handle termination in System.CommandLine

9/20/2022 • 2 minutes to read • [Edit Online](#)

IMPORTANT

`System.CommandLine` is currently in PREVIEW, and this documentation is for version 2.0 beta 4. Some information relates to prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, express or implied, with respect to the information provided here.

To handle termination, inject a `CancellationToken` instance into your handler code. This token can then be passed along to async APIs that you call from within your handler, as shown in the following example:

```
static async Task<int> Main(string[] args)
{
    int returnCode = 0;

    var urlOption = new Option<string>("--url", "A URL.");

    var rootCommand = new RootCommand("Handle termination example");
    rootCommand.Add(urlOption);

    rootCommand.SetHandler(async (context) =>
    {
        string? urlOptionValue = context.ParseResult.GetValueForOption(urlOption);
        var token = context.GetCancellationToken();
        returnCode = await DoRootCommand(urlOptionValue, token);
    });

    await rootCommand.InvokeAsync(args);

    return returnCode;
}

public static async Task<int> DoRootCommand(
    string? urlOptionValue, CancellationToken cancellationToken)
{
    try
    {
        using (var httpClient = new HttpClient())
        {
            await httpClient.GetAsync(urlOptionValue, cancellationToken);
        }
        return 0;
    }
    catch (OperationCanceledException)
    {
        Console.Error.WriteLine("The operation was aborted");
        return 1;
    }
}
```

The preceding code uses a `SetHandler` overload that gets an `InvocationContext` instance rather than one or more `IValueDescriptor<T>` objects. The `InvocationContext` is used to get the `CancellationToken` and `ParseResult` objects. `ParseResult` can provide argument or option values.

Cancellation actions can also be added directly using the [CancellationToken.Register](#) method.

For information about an alternative way to set the process exit code, see [Set exit codes](#).

See also

[System.CommandLine overview](#)

How to use middleware in System.CommandLine

9/20/2022 • 2 minutes to read • [Edit Online](#)

IMPORTANT

`System.CommandLine` is currently in PREVIEW, and this documentation is for version 2.0 beta 4. Some information relates to prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, express or implied, with respect to the information provided here.

This article explains how to work with middleware in command-line apps that are built with the `System.CommandLine` library. Use of middleware is an advanced topic that most `System.CommandLine` users won't need to consider.

Introduction to middleware

While each command has a handler that `System.CommandLine` will route to based on input, there's also a mechanism for short-circuiting or altering the input before your application logic is invoked. In between parsing and invocation, there's a chain of responsibility, which you can customize. A number of built-in features of `System.CommandLine` make use of this capability. This is how the `--help` and `--version` options short-circuit calls to your handler.

Each call in the pipeline can take action based on the `ParseResult` and return early, or choose to call the next item in the pipeline. The `ParseResult` can even be replaced during this phase. The last call in the chain is the handler for the specified command.

Add to the middleware pipeline

You can add a call to this pipeline by calling `CommandLineBuilderExtensions.AddMiddleware`. Here's an example of code that enables a custom `directive`. After creating a root command named `rootCommand`, the code as usual adds options, arguments, and handlers. Then the middleware is added:

```
var commandLineBuilder = new CommandLineBuilder(rootCommand);

commandLineBuilder.AddMiddleware(async (context, next) =>
{
    if (context.ParseResult.Directives.Contains("just-say-hi"))
    {
        context.Console.WriteLine("Hi!");
    }
    else
    {
        await next(context);
    }
});

commandLineBuilder.UseDefaults();
var parser = commandLineBuilder.Build();
await parser.InvokeAsync(args);
```

In the preceding code, the middleware writes out "Hi!" if the directive `[just-say-hi]` is found in the parse result. When this happens, the command's normal handler isn't invoked. It isn't invoked because the middleware doesn't call the `next` delegate.

In the example, `context` is [InvocationContext](#), a singleton structure that acts as the "root" of the entire command-handling process. This is the most powerful structure in [System.CommandLine](#), in terms of capabilities. There are two main uses for it in middleware:

- It provides access to the [BindingContext](#), [Parser](#), [Console](#), and [HelpBuilder](#) to retrieve dependencies that a middleware requires for its custom logic.
- You can set the [InvocationResult](#) or [ExitCode](#) properties in order to terminate command processing in a short-circuiting manner. An example is the `--help` option, which is implemented in this manner.

Here's the complete program, including required `using` directives.

```
using System.CommandLine;
using System.CommandLine.Builder;
using System.CommandLine.Parsing;

class Program
{
    static async Task Main(string[] args)
    {
        var delayOption = new Option<int>("--delay");
        var messageOption = new Option<string>("--message");

        var rootCommand = new RootCommand("Middleware example");
        rootCommand.Add(delayOption);
        rootCommand.Add(messageOption);

        rootCommand.SetHandler((delayOptionValue, messageOptionValue) =>
        {
            DoRootCommand(delayOptionValue, messageOptionValue);
        },
        delayOption, messageOption);

        var commandLineBuilder = new CommandLineBuilder(rootCommand);

        commandLineBuilder.AddMiddleware(async (context, next) =>
        {
            if (context.ParseResult.Directives.Contains("just-say-hi"))
            {
                context.Console.WriteLine("Hi!");
            }
            else
            {
                await next(context);
            }
        });
    }

    public static void DoRootCommand(int delay, string message)
    {
        Console.WriteLine($"--delay = {delay}");
        Console.WriteLine($"--message = {message}");
    }
}
```

Here's an example command line and resulting output from the preceding code:

```
myapp [just-say-hi] --delay 42 --message "Hello world!"
```

Hi!

See also

[System.CommandLine overview](#)

File and Stream I/O

9/20/2022 • 7 minutes to read • [Edit Online](#)

File and stream I/O (input/output) refers to the transfer of data either to or from a storage medium. In .NET, the `System.IO` namespaces contain types that enable reading and writing, both synchronously and asynchronously, on data streams and files. These namespaces also contain types that perform compression and decompression on files, and types that enable communication through pipes and serial ports.

A file is an ordered and named collection of bytes that has persistent storage. When you work with files, you work with directory paths, disk storage, and file and directory names. In contrast, a stream is a sequence of bytes that you can use to read from and write to a backing store, which can be one of several storage mediums (for example, disks or memory). Just as there are several backing stores other than disks, there are several kinds of streams other than file streams, such as network, memory, and pipe streams.

Files and directories

You can use the types in the `System.IO` namespace to interact with files and directories. For example, you can get and set properties for files and directories, and retrieve collections of files and directories based on search criteria.

For path naming conventions and the ways to express a file path for Windows systems, including with the DOS device syntax supported in .NET Core 1.1 and later and .NET Framework 4.6.2 and later, see [File path formats on Windows systems](#).

Here are some commonly used file and directory classes:

- `File` - provides static methods for creating, copying, deleting, moving, and opening files, and helps create a `FileStream` object.
- `FileInfo` - provides instance methods for creating, copying, deleting, moving, and opening files, and helps create a `FileStream` object.
- `Directory` - provides static methods for creating, moving, and enumerating through directories and subdirectories.
- `DirectoryInfo` - provides instance methods for creating, moving, and enumerating through directories and subdirectories.
- `Path` - provides methods and properties for processing directory strings in a cross-platform manner.

You should always provide robust exception handling when calling filesystem methods. For more information, see [Handling I/O errors](#).

In addition to using these classes, Visual Basic users can use the methods and properties provided by the `Microsoft.VisualBasic.FileIO.FileSystem` class for file I/O.

See [How to: Copy Directories](#), [How to: Create a Directory Listing](#), and [How to: Enumerate Directories and Files](#).

Streams

The abstract base class `Stream` supports reading and writing bytes. All classes that represent streams inherit from the `Stream` class. The `Stream` class and its derived classes provide a common view of data sources and repositories, and isolate the programmer from the specific details of the operating system and underlying devices.

Streams involve three fundamental operations:

- Reading - transferring data from a stream into a data structure, such as an array of bytes.
- Writing - transferring data to a stream from a data source.
- Seeking - querying and modifying the current position within a stream.

Depending on the underlying data source or repository, a stream might support only some of these capabilities. For example, the [PipeStream](#) class does not support seeking. The [CanRead](#), [CanWrite](#), and [CanSeek](#) properties of a stream specify the operations that the stream supports.

Here are some commonly used stream classes:

- [FileStream](#) – for reading and writing to a file.
- [IsolatedStorageFileStream](#) – for reading and writing to a file in isolated storage.
- [MemoryStream](#) – for reading and writing to memory as the backing store.
- [BufferedStream](#) – for improving performance of read and write operations.
- [NetworkStream](#) – for reading and writing over network sockets.
- [PipeStream](#) – for reading and writing over anonymous and named pipes.
- [CryptoStream](#) – for linking data streams to cryptographic transformations.

For an example of working with streams asynchronously, see [Asynchronous File I/O](#).

Readers and writers

The [System.IO](#) namespace also provides types for reading encoded characters from streams and writing them to streams. Typically, streams are designed for byte input and output. The reader and writer types handle the conversion of the encoded characters to and from bytes so the stream can complete the operation. Each reader and writer class is associated with a stream, which can be retrieved through the class's [BaseStream](#) property.

Here are some commonly used reader and writer classes:

- [BinaryReader](#) and [BinaryWriter](#) – for reading and writing primitive data types as binary values.
- [StreamReader](#) and [StreamWriter](#) – for reading and writing characters by using an encoding value to convert the characters to and from bytes.
- [StringReader](#) and [StringWriter](#) – for reading and writing characters to and from strings.
- [TextReader](#) and [TextWriter](#) – serve as the abstract base classes for other readers and writers that read and write characters and strings, but not binary data.

See [How to: Read Text from a File](#), [How to: Write Text to a File](#), [How to: Read Characters from a String](#), and [How to: Write Characters to a String](#).

Asynchronous I/O operations

Reading or writing a large amount of data can be resource-intensive. You should perform these tasks asynchronously if your application needs to remain responsive to the user. With synchronous I/O operations, the UI thread is blocked until the resource-intensive operation has completed. Use asynchronous I/O operations when developing Windows 8.x Store apps to prevent creating the impression that your app has stopped working.

The asynchronous members contain `Async` in their names, such as the `CopyToAsync`, `FlushAsync`, `ReadAsync`, and `WriteAsync` methods. You use these methods with the `async` and `await` keywords.

For more information, see [Asynchronous File I/O](#).

Compression

Compression refers to the process of reducing the size of a file for storage. Decompression is the process of extracting the contents of a compressed file so they are in a usable format. The [System.IO.Compression](#) namespace contains types for compressing and decompressing files and streams.

The following classes are frequently used when compressing and decompressing files and streams:

- [ZipArchive](#) – for creating and retrieving entries in the zip archive.
- [ZipArchiveEntry](#) – for representing a compressed file.
- [ZipFile](#) – for creating, extracting, and opening a compressed package.
- [ZipFileExtensions](#) – for creating and extracting entries in a compressed package.
- [DeflateStream](#) – for compressing and decompressing streams using the Deflate algorithm.
- [GZipStream](#) – for compressing and decompressing streams in gzip data format.

See [How to: Compress and Extract Files](#).

Isolated storage

Isolated storage is a data storage mechanism that provides isolation and safety by defining standardized ways of associating code with saved data. The storage provides a virtual file system that is isolated by user, assembly, and (optionally) domain. Isolated storage is particularly useful when your application does not have permission to access user files. You can save settings or files for your application in a manner that is controlled by the computer's security policy.

Isolated storage is not available for Windows 8.x Store apps; instead, use application data classes in the [Windows.Storage](#) namespace. For more information, see [Application data](#).

The following classes are frequently used when implementing isolated storage:

- [IsolatedStorage](#) – provides the base class for isolated storage implementations.
- [IsolatedStorageFile](#) – provides an isolated storage area that contains files and directories.
- [IsolatedStorageFileStream](#) - exposes a file within isolated storage.

See [Isolated Storage](#).

I/O operations in Windows Store apps

.NET for Windows 8.x Store apps contains many of the types for reading from and writing to streams; however, this set does not include all the .NET I/O types.

Some important differences to note when using I/O operations in Windows 8.x Store apps:

- Types specifically related to file operations, such as [File](#), [FileInfo](#), [Directory](#) and [DirectoryInfo](#), are not included in the .NET for Windows 8.x Store apps. Instead, use the types in the [Windows.Storage](#) namespace of the Windows Runtime, such as [StorageFile](#) and [StorageFolder](#).
- Isolated storage is not available; instead, use [application data](#).

- Use asynchronous methods, such as [ReadAsync](#) and [WriteAsync](#), to prevent blocking the UI thread.
- The path-based compression types [ZipFile](#) and [ZipFileExtensions](#) are not available. Instead, use the types in the [Windows.Storage.Compression](#) namespace.

You can convert between .NET Framework streams and Windows Runtime streams, if necessary. For more information, see [How to: Convert Between .NET Framework Streams and Windows Runtime Streams](#) or [WindowsRuntimeStreamExtensions](#).

For more information about I/O operations in a Windows 8.x Store app, see [Quickstart: Reading and writing files](#).

I/O and security

When you use the classes in the [System.IO](#) namespace, you must follow operating system security requirements such as access control lists (ACLs) to control access to files and directories. This requirement is in addition to any [FileIOPermission](#) requirements. You can manage ACLs programmatically. For more information, see [How to: Add or Remove Access Control List Entries](#).

Default security policies prevent Internet or intranet applications from accessing files on the user's computer. Therefore, do not use the I/O classes that require a path to a physical file when writing code that will be downloaded over the internet or intranet. Instead, use [isolated storage](#) for .NET applications.

A security check is performed only when the stream is constructed. Therefore, do not open a stream and then pass it to less-trusted code or application domains.

Related topics

- [Common I/O Tasks](#)
Provides a list of I/O tasks associated with files, directories, and streams, and links to relevant content and examples for each task.
- [Asynchronous File I/O](#)
Describes the performance advantages and basic operation of asynchronous I/O.
- [Isolated Storage](#)
Describes a data storage mechanism that provides isolation and safety by defining standardized ways of associating code with saved data.
- [Pipes](#)
Describes anonymous and named pipe operations in .NET.
- [Memory-Mapped Files](#)
Describes memory-mapped files, which contain the contents of files on disk in virtual memory. You can use memory-mapped files to edit very large files and to create shared memory for interprocess communication.

File path formats on Windows systems

9/20/2022 • 15 minutes to read • [Edit Online](#)

Members of many of the types in the [System.IO](#) namespace include a `path` parameter that lets you specify an absolute or relative path to a file system resource. This path is then passed to [Windows file system APIs](#). This topic discusses the formats for file paths that you can use on Windows systems.

Traditional DOS paths

A standard DOS path can consist of three components:

- A volume or drive letter followed by the volume separator (`:`).
- A directory name. The [directory separator character](#) separates subdirectories within the nested directory hierarchy.
- An optional filename. The [directory separator character](#) separates the file path and the filename.

If all three components are present, the path is absolute. If no volume or drive letter is specified and the directory name begins with the [directory separator character](#), the path is relative from the root of the current drive. Otherwise, the path is relative to the current directory. The following table shows some possible directory and file paths.

PATH	DESCRIPTION
<code>C:\Documents\Newsletters\Summer2018.pdf</code>	An absolute file path from the root of drive <code>C:</code> .
<code>\Program Files\Custom Utilities\StringFinder.exe</code>	An absolute path from the root of the current drive.
<code>2018\January.xlsx</code>	A relative path to a file in a subdirectory of the current directory.
<code>..\Publications\TravelBrochure.pdf</code>	A relative path to a file in a directory starting from the current directory.
<code>C:\Projects\apilibrary\apilibrary.sln</code>	An absolute path to a file from the root of drive <code>C:</code> .
<code>C:Projects\apilibrary\apilibrary.sln</code>	A relative path from the current directory of the <code>C:</code> drive.

IMPORTANT

Note the difference between the last two paths. Both specify the optional volume specifier (`C:` in both cases), but the first begins with the root of the specified volume, whereas the second does not. As result, the first is an absolute path from the root directory of drive `C:`, whereas the second is a relative path from the current directory of drive `C:`. Use of the second form when the first is intended is a common source of bugs that involve Windows file paths.

You can determine whether a file path is fully qualified (that is, if the path is independent of the current directory and does not change when the current directory changes) by calling the [Path.IsPathFullyQualified](#) method. Note that such a path can include relative directory segments (`.` and `..`) and still be fully qualified if the resolved path always points to the same location.

The following example illustrates the difference between absolute and relative paths. It assumes that the

directory `D:\FY2018\` exists, and that you haven't set any current directory for `D:\` from the command prompt before running the example.

```
using System;
using System.Diagnostics;
using System.IO;
using System.Reflection;

public class Example
{
    public static void Main(string[] args)
    {
        Console.WriteLine($"Current directory is '{Environment.CurrentDirectory}'");
        Console.WriteLine("Setting current directory to 'C:\\\\'");

        Directory.SetCurrentDirectory(@"C:\\");
        string path = Path.GetFullPath(@"D:\FY2018");
        Console.WriteLine($"'D:\\FY2018' resolves to {path}");
        path = Path.GetFullPath(@"D:FY2018");
        Console.WriteLine($"'D:FY2018' resolves to {path}");

        Console.WriteLine("Setting current directory to 'D:\\Docs'");
        Directory.SetCurrentDirectory(@"D:\\Docs");

        path = Path.GetFullPath(@"D:\FY2018");
        Console.WriteLine($"'D:\\FY2018' resolves to {path}");
        path = Path.GetFullPath(@"D:FY2018");

        // This will be "D:\\Docs\\FY2018" as it happens to match the drive of the current directory
        Console.WriteLine($"'D:FY2018' resolves to {path}");

        Console.WriteLine("Setting current directory to 'C:\\\\'");
        Directory.SetCurrentDirectory(@"C:\\");

        path = Path.GetFullPath(@"D:\FY2018");
        Console.WriteLine($"'D:\\FY2018' resolves to {path}");

        // This will be either "D:\\FY2018" or "D:\\FY2018\\FY2018" in the subprocess. In the sub process,
        // the command prompt set the current directory before launch of our application, which
        // sets a hidden environment variable that is considered.
        path = Path.GetFullPath(@"D:FY2018");
        Console.WriteLine($"'D:FY2018' resolves to {path}");

        if (args.Length < 1)
        {
            Console.WriteLine(@"Launching again, after setting current directory to D:\\FY2018");
            Uri currentExe = new Uri(Assembly.GetExecutingAssembly().GetName().CodeBase, UriKind.Absolute);
            string commandLine = $"{"/C cd D:\\FY2018 & \'{currentExe.LocalPath}\' stop"}";
            ProcessStartInfo psi = new ProcessStartInfo("cmd", commandLine); ;
            Process.Start(psi).WaitForExit();

            Console.WriteLine("Sub process returned:");
            path = Path.GetFullPath(@"D:\FY2018");
            Console.WriteLine($"'D:\\FY2018' resolves to {path}");
            path = Path.GetFullPath(@"D:FY2018");
            Console.WriteLine($"'D:FY2018' resolves to {path}");
        }
        Console.WriteLine("Press any key to continue... ");
        Console.ReadKey();
    }
}

// The example displays the following output:
//      Current directory is 'C:\\Programs\\file-paths'
//      Setting current directory to 'C:\\'
//      'D:\\FY2018' resolves to D:\\FY2018
//      'D:FY2018' resolves to d:\\FY2018
//      Setting current directory to 'D:\\Docs'
```

```

//      'D:\FY2018' resolves to D:\FY2018
//      'D:FY2018' resolves to D:\Docs\FY2018
//      Setting current directory to 'C:\'
//      'D:\FY2018' resolves to D:\FY2018
//      'D:FY2018' resolves to d:\FY2018
//      Launching again, after setting current directory to D:\FY2018
//      Sub process returned:
//      'D:\FY2018' resolves to D:\FY2018
//      'D:FY2018' resolves to d:\FY2018
// The subprocess displays the following output:
//      Current directory is 'C:\'
//      Setting current directory to 'C:\'
//      'D:\FY2018' resolves to D:\FY2018
//      'D:FY2018' resolves to D:\FY2018\FY2018
//      Setting current directory to 'D:\Docs'
//      'D:\FY2018' resolves to D:\FY2018
//      'D:FY2018' resolves to D:\Docs\FY2018
//      Setting current directory to 'C:\'
//      'D:\FY2018' resolves to D:\FY2018
//      'D:FY2018' resolves to D:\FY2018\FY2018

```

```

Imports System.Diagnostics
Imports System.IO
Imports System.Reflection

Public Module Example

    Public Sub Main(args() As String)
        Console.WriteLine($"Current directory is '{Environment.CurrentDirectory}'")
        Console.WriteLine("Setting current directory to 'C:\\'")
        Directory.SetCurrentDirectory("C:\\")

        Dim filePath As String = Path.GetFullPath("D:\FY2018")
        Console.WriteLine($"'D:\\FY2018' resolves to {filePath}")
        filePath = Path.GetFullPath("D:FY2018")
        Console.WriteLine($"'D:FY2018' resolves to {filePath}")

        Console.WriteLine("Setting current directory to 'D:\\Docs\\'")
        Directory.SetCurrentDirectory("D:\\Docs\\")

        filePath = Path.GetFullPath("D:\FY2018")
        Console.WriteLine($"'D:\\FY2018' resolves to {filePath}")
        filePath = Path.GetFullPath("D:FY2018")

        ' This will be "D:\\Docs\\FY2018" as it happens to match the drive of the current directory
        Console.WriteLine($"'D:FY2018' resolves to {filePath}")

        Console.WriteLine("Setting current directory to 'C:\\\\'")
        Directory.SetCurrentDirectory("C:\\")

        filePath = Path.GetFullPath("D:\FY2018")
        Console.WriteLine($"'D:\\FY2018' resolves to {filePath}")

        ' This will be either "D:\FY2018" or "D:\FY2018\FY2018" in the subprocess. In the sub process,
        ' the command prompt set the current directory before launch of our application, which
        ' sets a hidden environment variable that is considered.
        filePath = Path.GetFullPath("D:FY2018")
        Console.WriteLine($"'D:FY2018' resolves to {filePath}")

        If args.Length < 1 Then
            Console.WriteLine("Launching again, after setting current directory to D:\FY2018")
            Dim currentExe As New Uri(Assembly.GetExecutingAssembly().GetName().CodeBase, UriKind.Absolute)
            Dim commandLine As String = $"/C cd D:\FY2018 & \"{currentExe.LocalPath}"" stop"
            Dim psi As New ProcessStartInfo("cmd", commandLine)
            Process.Start(psi).WaitForExit()

            Console.WriteLine("Sub process returned:")
            Console.WriteLine("D:\FY2018 - D:\FY2018\FY2018\\FY2018")
        End If
    End Sub
End Module

```

```

    filePath = Path.GetFullPath("D:\FY2018")
    Console.WriteLine($"'D:\FY2018' resolves to {filePath}")
    filePath = Path.GetFullPath("D:FY2018")
    Console.WriteLine($"'D:FY2018' resolves to {filePath}")
End If
Console.WriteLine("Press any key to continue... ")
Console.ReadKey()
End Sub
End Module
' The example displays the following output:
'   Current directory is 'C:\Programs\file-paths'
'   Setting current directory to 'C:\'
'   'D:\FY2018' resolves to D:\FY2018
'   'D:FY2018' resolves to d:\FY2018
'   Setting current directory to 'D:\Docs'
'   'D:\FY2018' resolves to D:\FY2018
'   'D:FY2018' resolves to D:\Docs\FY2018
'   Setting current directory to 'C:\'
'   'D:\FY2018' resolves to D:\FY2018
'   'D:FY2018' resolves to d:\FY2018
'   Launching again, after setting current directory to D:\FY2018
Sub process returned:
'   'D:\FY2018' resolves to D:\FY2018
'   'D:FY2018' resolves to d:\FY2018
' The subprocess displays the following output:
'   Current directory is 'C:\'
'   Setting current directory to 'C:\'
'   'D:\FY2018' resolves to D:\FY2018
'   'D:FY2018' resolves to D:\FY2018\FY2018
'   Setting current directory to 'D:\Docs'
'   'D:\FY2018' resolves to D:\FY2018
'   'D:FY2018' resolves to D:\Docs\FY2018
'   Setting current directory to 'C:\'
'   'D:\FY2018' resolves to D:\FY2018
'   'D:FY2018' resolves to D:\FY2018\FY2018

```

If you would like to see code comments translated to languages other than English, let us know in [this GitHub discussion issue](#).

UNC paths

Universal naming convention (UNC) paths, which are used to access network resources, have the following format:

- A server or host name, which is prefaced by `\\"`. The server name can be a NetBIOS machine name or an IP/FQDN address (IPv4 as well as v6 are supported).
- A share name, which is separated from the host name by `\`. Together, the server and share name make up the volume.
- A directory name. The [directory separator character](#) separates subdirectories within the nested directory hierarchy.
- An optional filename. The [directory separator character](#) separates the file path and the filename.

The following are some examples of UNC paths:

PATH	DESCRIPTION
<code>\\"system07\C\$\\"</code>	The root directory of the <code>C:</code> drive on <code>system07</code> .
<code>\\"Server2\Share\Test\Foo.txt\"</code>	The <code>Foo.txt</code> file in the <code>Test</code> directory of the <code>\\"Server2\Share</code> volume.

UNC paths must always be fully qualified. They can include relative directory segments (`.` and `..`), but these must be part of a fully qualified path. You can use relative paths only by mapping a UNC path to a drive letter.

DOS device paths

The Windows operating system has a unified object model that points to all resources, including files. These object paths are accessible from the console window and are exposed to the Win32 layer through a special folder of symbolic links that legacy DOS and UNC paths are mapped to. This special folder is accessed via the DOS device path syntax, which is one of:

```
\.\.\C:\Test\Foo.txt \?\C:\Test\Foo.txt
```

In addition to identifying a drive by its drive letter, you can identify a volume by using its volume GUID. This takes the form:

```
\.\.\Volume{b75e2c83-0000-0000-0000-602f00000000}\Test\Foo.txt  
\?\Volume{b75e2c83-0000-0000-0000-602f00000000}\Test\Foo.txt
```

NOTE

DOS device path syntax is supported on .NET implementations running on Windows starting with .NET Core 1.1 and .NET Framework 4.6.2.

The DOS device path consists of the following components:

- The device path specifier (`\.\.` or `\?\`), which identifies the path as a DOS device path.

NOTE

The `\?\` is supported in all versions of .NET Core and .NET 5+ and in .NET Framework starting with version 4.6.2.

- A symbolic link to the "real" device object (C: in the case of a drive name, or Volume{b75e2c83-0000-0000-0000-602f00000000} in the case of a volume GUID).

The first segment of the DOS device path after the device path specifier identifies the volume or drive. (For example, `\?\C:\` and `\.\BootPartition\`.)

There is a specific link for UNC's that is called, not surprisingly, `UNC`. For example:

```
\.\UNC\Server\Share\Test\Foo.txt \?\UNC\Server\Share\Test\Foo.txt
```

For device UNC's, the server/share portion forms the volume. For example, in `\?\server1\c:\utilities\filecomparer\`, the server/share portion is `server1\utilities`. This is significant when calling a method such as `Path.GetFullPath(String, String)` with relative directory segments; it is never possible to navigate past the volume.

DOS device paths are fully qualified by definition and cannot begin with a relative directory segment (`.` or `..`). Current directories never enter into their usage.

Example: Ways to refer to the same file

The following example illustrates some of the ways in which you can refer to a file when using the APIs in the `System.IO` namespace. The example instantiates a `FileInfo` object and uses its `Name` and `Length` properties to display the filename and the length of the file.

```

using System;
using System.IO;

class Program
{
    static void Main()
    {
        string[] filenames = {
            @"c:\temp\test-file.txt",
            @"\127.0.0.1\c$\temp\test-file.txt",
            @"\LOCALHOST\c$\temp\test-file.txt",
            @"\.\c:\temp\test-file.txt",
            @"\?\c:\temp\test-file.txt",
            @"\.\UNC\LOCALHOST\c$\temp\test-file.txt",
            @"\127.0.0.1\c$\temp\test-file.txt" };

        foreach (var filename in filenames)
        {
            FileInfo fi = new FileInfo(filename);
            Console.WriteLine($"file {fi.Name}: {fi.Length:N0} bytes");
        }
    }
}

// The example displays output like the following:
//     file test-file.txt: 22 bytes

```

```

Imports System.IO

Module Program
    Sub Main()
        Dim filenames() As String = {
            "c:\temp\test-file.txt",
            "\127.0.0.1\c$\temp\test-file.txt",
            @"\LOCALHOST\c$\temp\test-file.txt",
            @"\.\c:\temp\test-file.txt",
            @"\?\c:\temp\test-file.txt",
            @"\.\UNC\LOCALHOST\c$\temp\test-file.txt",
            @"\127.0.0.1\c$\temp\test-file.txt" }

        For Each filename In filenames
            Dim fi As New FileInfo(filename)
            Console.WriteLine($"file {fi.Name}: {fi.Length:N0} bytes")
        Next
    End Sub
End Module

```

Path normalization

Almost all paths passed to Windows APIs are normalized. During normalization, Windows performs the following steps:

- Identifies the path.
- Applies the current directory to partially qualified (relative) paths.
- Canonicalizes component and directory separators.
- Evaluates relative directory components (. for the current directory and .. for the parent directory).

- Trims certain characters.

This normalization happens implicitly, but you can do it explicitly by calling the [Path.GetFullPath](#) method, which wraps a call to the [GetFullPathName\(\)](#) function. You can also call the Windows [GetFullPathName\(\)](#) function directly using P/Invoke.

Identify the path

The first step in path normalization is identifying the type of path. Paths fall into one of a few categories:

- They are device paths; that is, they begin with two separators and a question mark or period (`\?\?` or `\?\.`).
- They are UNC paths; that is, they begin with two separators without a question mark or period.
- They are fully qualified DOS paths; that is, they begin with a drive letter, a volume separator, and a component separator (`C:\`).
- They designate a legacy device (`CON`, `LPT1`).
- They are relative to the root of the current drive; that is, they begin with a single component separator (`\`).
- They are relative to the current directory of a specified drive; that is, they begin with a drive letter, a volume separator, and no component separator (`C:\`).
- They are relative to the current directory; that is, they begin with anything else (`temp\testfile.txt`).

The type of the path determines whether or not a current directory is applied in some way. It also determines what the "root" of the path is.

Handle legacy devices

If the path is a legacy DOS device such as `CON`, `COM1`, or `LPT1`, it is converted into a device path by prepending `\?\.` and returned.

A path that begins with a legacy device name is always interpreted as a legacy device by the [Path.GetFullPath\(String\)](#) method. For example, the DOS device path for `CON.TXT` is `\?\.\CON`, and the DOS device path for `COM1.TXT\file1.txt` is `\?\.\COM1`.

Apply the current directory

If a path isn't fully qualified, Windows applies the current directory to it. UNC paths and device paths do not have the current directory applied. Neither does a full drive with separator `C:\`.

If the path starts with a single component separator, the drive from the current directory is applied. For example, if the file path is `\utilities` and the current directory is `C:\temp\`, normalization produces `C:\utilities`.

If the path starts with a drive letter, volume separator, and no component separator, the last current directory set from the command shell for the specified drive is applied. If the last current directory was not set, the drive alone is applied. For example, if the file path is `D:sources`, the current directory is `C:\Documents\`, and the last current directory on drive D: was `D:\sources\`, the result is `D:\sources\sources`. These "drive relative" paths are a common source of program and script logic errors. Assuming that a path beginning with a letter and a colon isn't relative is obviously not correct.

If the path starts with something other than a separator, the current drive and current directory are applied. For example, if the path is `filecompare` and the current directory is `C:\utilities\`, the result is `C:\utilities\filecompare\`.

IMPORTANT

Relative paths are dangerous in multithreaded applications (that is, most applications) because the current directory is a per-process setting. Any thread can change the current directory at any time. Starting with .NET Core 2.1, you can call the [Path.GetFullPath\(String, String\)](#) method to get an absolute path from a relative path and the base path (the current directory) that you want to resolve it against.

Canonicalize separators

All forward slashes (/) are converted into the standard Windows separator, the back slash (\). If they are present, a series of slashes that follow the first two slashes are collapsed into a single slash.

Evaluate relative components

As the path is processed, any components or segments that are composed of a single or a double period (.) or (...) are evaluated:

- For a single period, the current segment is removed, since it refers to the current directory.
- For a double period, the current segment and the parent segment are removed, since the double period refers to the parent directory.

Parent directories are only removed if they aren't past the root of the path. The root of the path depends on the type of path. It is the drive (c:\) for DOS paths, the server/share for UNC's (\Server\Share), and the device path prefix for device paths (\?\ or \.\).

Trim characters

Along with the runs of separators and relative segments removed earlier, some additional characters are removed during normalization:

- If a segment ends in a single period, that period is removed. (A segment of a single or double period is normalized in the previous step. A segment of three or more periods is not normalized and is actually a valid file/directory name.)
- If the path doesn't end in a separator, all trailing periods and spaces (U+0020) are removed. If the last segment is simply a single or double period, it falls under the relative components rule above.

This rule means that you can create a directory name with a trailing space by adding a trailing separator after the space.

IMPORTANT

You should **never** create a directory or filename with a trailing space. Trailing spaces can make it difficult or impossible to access a directory, and applications commonly fail when attempting to handle directories or files whose names include trailing spaces.

Skip normalization

Normally, any path passed to a Windows API is (effectively) passed to the [GetFullPathName function](#) and normalized. There is one important exception: a device path that begins with a question mark instead of a period. Unless the path starts exactly with \?\ (note the use of the canonical backslash), it is normalized.

Why would you want to skip normalization? There are three major reasons:

1. To get access to paths that are normally unavailable but are legal. A file or directory called hidden., for example, is impossible to access in any other way.
2. To improve performance by skipping normalization if you've already normalized.
3. On .NET Framework only, to skip the MAX_PATH check for path length to allow for paths that are greater than 259 characters. Most APIs allow this, with some exceptions.

NOTE

.NET Core and .NET 5+ handles long paths implicitly and does not perform a `MAX_PATH` check. The `MAX_PATH` check applies only to .NET Framework.

Skipping normalization and max path checks is the only difference between the two device path syntaxes; they are otherwise identical. Be careful with skipping normalization, since you can easily create paths that are difficult for "normal" applications to deal with.

Paths that start with `\?\` are still normalized if you explicitly pass them to the [GetFullPathName function](#).

You can pass paths of more than `MAX_PATH` characters to [GetFullPathName](#) without `\?\`. It supports arbitrary length paths up to the maximum string size that Windows can handle.

Case and the Windows file system

A peculiarity of the Windows file system that non-Windows users and developers find confusing is that path and directory names are case-insensitive. That is, directory and file names reflect the casing of the strings used when they are created. For example, the method call

```
Directory.CreateDirectory("TeStDiReCtOrY");
```

```
Directory.CreateDirectory("TeStDiReCtOrY")
```

creates a directory named `TeStDiReCtOrY`. If you rename a directory or file to change its case, the directory or file name reflects the case of the string used when you rename it. For example, the following code renames a file named `test.txt` to `Test.txt`:

```
using System.IO;

class Example
{
    static void Main()
    {
        var fi = new FileInfo(@".\test.txt");
        fi.MoveTo(@".\Test.txt");
    }
}
```

```
Imports System.IO

Module Example
    Public Sub Main()
        Dim fi As New FileInfo(".\test.txt")
        fi.MoveTo(".\Test.txt")
    End Sub
End Module
```

However, directory and file name comparisons are case-insensitive. If you search for a file named `"test.txt"`, .NET file system APIs ignore case in the comparison. `"Test.txt"`, `"TEST.TXT"`, `"test.TXT"`, and any other combination of uppercase and lowercase letters will match `"test.txt"`.

Common I/O Tasks

9/20/2022 • 2 minutes to read • [Edit Online](#)

The [System.IO](#) namespace provides several classes that allow for various actions, such as reading and writing, to be performed on files, directories, and streams. For more information, see [File and Stream I/O](#).

Common File Tasks

TO DO THIS...	SEE THE EXAMPLE IN THIS TOPIC...
Create a text file	File.CreateText method FileInfo.CreateText method File.Create method FileInfo.Create method
Write to a text file	How to: Write Text to a File How to: Write a Text File (C++/CLI)
Read from a text file	How to: Read Text from a File
Append text to a file	How to: Open and Append to a Log File File.AppendText method FileInfo.AppendText method
Rename or move a file	File.Move method FileInfo.MoveTo method
Delete a file	File.Delete method FileInfo.Delete method
Copy a file	File.Copy method FileInfo.CopyTo method
Get the size of a file	FileInfo.Length property
Get the attributes of a file	File.GetAttributes method
Set the attributes of a file	File.SetAttributes method
Determine whether a file exists	File.Exists method
Read from a binary file	How to: Read and Write to a Newly Created Data File

TO DO THIS...	SEE THE EXAMPLE IN THIS TOPIC...
Write to a binary file	How to: Read and Write to a Newly Created Data File
Retrieve a file name extension	Path.GetExtension method
Retrieve the fully qualified path of a file	Path.GetFullPath method
Retrieve the file name and extension from a path	Path.GetFileName method
Change the extension of a file	Path.ChangeExtension method

Common Directory Tasks

TO DO THIS...	SEE THE EXAMPLE IN THIS TOPIC...
Access a file in a special folder such as My Documents	How to: Write Text to a File
Create a directory	Directory.CreateDirectory method FileInfo.Directory property
Create a subdirectory	DirectoryInfo.CreateSubdirectory method
Rename or move a directory	Directory.Move method DirectoryInfo.MoveTo method
Copy a directory	How to: Copy Directories
Delete a directory	Directory.Delete method DirectoryInfo.Delete method
See the files and subdirectories in a directory	How to: Enumerate Directories and Files
Find the size of a directory	System.IO.Directory class
Determine whether a directory exists	Directory.Exists method

See also

- [File and Stream I/O](#)
- [Composing Streams](#)
- [Asynchronous File I/O](#)

How to: Copy directories

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article demonstrates how to use I/O classes to synchronously copy the contents of a directory to another location.

For an example of asynchronous file copy, see [Asynchronous file I/O](#).

This example copies subdirectories by setting the `recursive` parameter of the `CopyDirectory` method to `true`. The `copyDirectory` method recursively copies subdirectories by calling itself on each subdirectory until there are no more to copy.

Example

```
using System.IO;

CopyDirectory(@".\", @".\copytest", true);

static void CopyDirectory(string sourceDir, string destinationDir, bool recursive)
{
    // Get information about the source directory
    var dir = new DirectoryInfo(sourceDir);

    // Check if the source directory exists
    if (!dir.Exists)
        throw new DirectoryNotFoundException($"Source directory not found: {dir.FullName}");

    // Cache directories before we start copying
    DirectoryInfo[] dirs = dir.GetDirectories();

    // Create the destination directory
    Directory.CreateDirectory(destinationDir);

    // Get the files in the source directory and copy to the destination directory
    foreach (FileInfo file in dir.GetFiles())
    {
        string targetFilePath = Path.Combine(destinationDir, file.Name);
        file.CopyTo(targetFilePath);
    }

    // If recursive and copying subdirectories, recursively call this method
    if (recursive)
    {
        foreach (DirectoryInfo subDir in dirs)
        {
            string newDestinationDir = Path.Combine(destinationDir, subDir.Name);
            CopyDirectory(subDir.FullName, newDestinationDir, true);
        }
    }
}
```

```

Imports System.IO

Module Program
    Sub Main(args As String())
        CopyDirectory(".", ".\copytest", True)
    End Sub

    Public Sub CopyDirectory(sourceDir As String, destinationDir As String, recursive As Boolean)

        ' Get information about the source directory
        Dim dir As New DirectoryInfo(sourceDir)

        ' Check if the source directory exists
        If Not dir.Exists Then
            Throw New DirectoryNotFoundException($"Source directory not found: {dir.FullName}")
        End If

        ' Cache directories before we start copying
        Dim dirs As DirectoryInfo() = dir.GetDirectories()

        ' Create the destination directory
        Directory.CreateDirectory(destinationDir)

        ' Get the files in the source directory and copy to the destination directory
        For Each file As FileInfo In dir.GetFiles()
            Dim targetFilePath As String = Path.Combine(destinationDir, file.Name)
            file.CopyTo(targetFilePath)
        Next

        ' If recursive and copying subdirectories, recursively call this method
        If recursive Then
            For Each subDir As DirectoryInfo In dirs
                Dim newDestinationDir As String = Path.Combine(destinationDir, subDir.Name)
                CopyDirectory(subDir.FullName, newDestinationDir, True)
            Next
        End If
    End Sub
End Module

```

If you would like to see code comments translated to languages other than English, let us know in [this GitHub discussion issue](#).

See also

- [FileInfo](#)
- [DirectoryInfo](#)
- [FileStream](#)
- [File and stream I/O](#)
- [Common I/O tasks](#)
- [Asynchronous file I/O](#)

How to: Enumerate directories and files

9/20/2022 • 4 minutes to read • [Edit Online](#)

Enumerable collections provide better performance than arrays when you work with large collections of directories and files. To enumerate directories and files, use methods that return an enumerable collection of directory or file names, or their [DirectoryInfo](#), [FileInfo](#), or [FileSystemInfo](#) objects.

If you want to search and return only the names of directories or files, use the enumeration methods of the [Directory](#) class. If you want to search and return other properties of directories or files, use the [DirectoryInfo](#) and [FileSystemInfo](#) classes.

You can use enumerable collections from these methods as the [IEnumerable<T>](#) parameter for constructors of collection classes like [List<T>](#).

The following table summarizes the methods that return enumerable collections of files and directories:

TO SEARCH AND RETURN	USE METHOD
Directory names	Directory.EnumerateDirectories
Directory information (DirectoryInfo)	DirectoryInfo.EnumerateDirectories
File names	Directory.EnumerateFiles
File information (FileInfo)	DirectoryInfo.EnumerateFiles
File system entry names	Directory.EnumerateFileSystemEntries
File system entry information (FileSystemInfo)	DirectoryInfo.EnumerateFileSystemInfos
Directory and file names	Directory.EnumerateFileSystemEntries

NOTE

Although you can immediately enumerate all the files in the subdirectories of a parent directory by using the [AllDirectories](#) option of the optional [SearchOption](#) enumeration, [UnauthorizedAccessException](#) errors may make the enumeration incomplete. You can catch these exceptions by first enumerating directories and then enumerating files.

Examples: Use the Directory class

The following example uses the [Directory.EnumerateDirectories\(String\)](#) method to get a list of the top-level directory names in a specified path.

```

using System;
using System.Collections.Generic;
using System.IO;

class Program
{
    private static void Main(string[] args)
    {
        try
        {
            // Set a variable to the My Documents path.
            string docPath = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments);

            List<string> dirs = new List<string>(Directory.EnumerateDirectories(docPath));

            foreach (var dir in dirs)
            {
                Console.WriteLine($"{dir.Substring(dir.LastIndexOf(Path.DirectorySeparatorChar) + 1)}");
            }
            Console.WriteLine($"{dirs.Count} directories found.");
        }
        catch (UnauthorizedAccessException ex)
        {
            Console.WriteLine(ex.Message);
        }
        catch (PathTooLongException ex)
        {
            Console.WriteLine(ex.Message);
        }
    }
}

```

```

Imports System.Collections.Generic
Imports System.IO

Module Module1

    Sub Main()
        Try
            Dim dirPath As String = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments)

            Dim dirs As List(Of String) = New List(Of String)(Directory.EnumerateDirectories(dirPath))

            For Each folder In dirs
                Console.WriteLine($"{dir.Substring(dir.LastIndexOf(Path.DirectorySeparatorChar) + 1)}")
            Next
            Console.WriteLine($"{dirs.Count} directories found.")
        Catch ex As UnauthorizedAccessException
            Console.WriteLine(ex.Message)
        Catch ex As PathTooLongException
            Console.WriteLine(ex.Message)
        End Try
    End Sub
End Module

```

The following example uses the [Directory.EnumerateFiles\(String, String, SearchOption\)](#) method to recursively enumerate all file names in a directory and subdirectories that match a certain pattern. It then reads each line of each file and displays the lines that contain a specified string, with their filenames and paths.

```

using System;
using System.IO;
using System.Linq;

class Program
{
    static void Main(string[] args)
    {
        try
        {
            // Set a variable to the My Documents path.
            string docPath =
                Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments);

            var files = from file in Directory.EnumerateFiles(docPath, "*.txt", SearchOption.AllDirectories)
                       from line in File.ReadLines(file)
                       where line.Contains("Microsoft")
                       select new
                       {
                           File = file,
                           Line = line
                       };

            foreach (var f in files)
            {
                Console.WriteLine($"{f.File}\t{f.Line}");
            }
            Console.WriteLine($"{files.Count()} files found.");
        }
        catch (UnauthorizedAccessException uAEx)
        {
            Console.WriteLine(uAEx.Message);
        }
        catch (PathTooLongException pathEx)
        {
            Console.WriteLine(pathEx.Message);
        }
    }
}

```

```

Imports System.IO
Imports System.Xml.Linq

Module Module1

    Sub Main()
        Try
            Dim docPath As String = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments)
            Dim files = From chkFile In Directory.EnumerateFiles(docPath, "*.txt",
SearchOption.AllDirectories)
                        From line In File.ReadLines(chkFile)
                        Where line.Contains("Microsoft")
                        Select New With {.curFile = chkFile, .curLine = line}

            For Each f In files
                Console.WriteLine($"{f.File}\t{f.Line}")
            Next
            Console.WriteLine($"{files.Count} files found.")
        Catch uAEx As UnauthorizedAccessException
            Console.WriteLine(uAEx.Message)
        Catch pathEx As PathTooLongException
            Console.WriteLine(pathEx.Message)
        End Try
    End Sub
End Module

```

Examples: Use the DirectoryInfo class

The following example uses the [DirectoryInfo.EnumerateDirectories](#) method to list a collection of top-level directories whose [CreationTimeUtc](#) is earlier than a certain [DateTime](#) value.

```
using System;
using System.IO;

namespace EnumDir
{
    class Program
    {
        static void Main(string[] args)
        {
            // Set a variable to the Documents path.
            string docPath = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments);

            DirectoryInfo dirPrograms = new DirectoryInfo(docPath);
            DateTime StartOf2009 = new DateTime(2009, 01, 01);

            var dirs = from dir in dirPrograms.EnumerateDirectories()
                       where dir.CreationTimeUtc > StartOf2009
                       select new
                       {
                           ProgDir = dir,
                       };

            foreach (var di in dirs)
            {
                Console.WriteLine($"{di.ProgDir.Name}");
            }
        }
    }
} // </Snippet1>
```

```
Imports System.IO

Module Module1

    Sub Main()

        Dim dirPath As String = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments)
        Dim dirPrograms As New DirectoryInfo(dirPath)
        Dim StartOf2009 As New DateTime(2009, 1, 1)

        Dim dirs = From dir In dirPrograms.EnumerateDirectories()
                   Where dir.CreationTimeUtc > StartOf2009

        For Each di As DirectoryInfo In dirs
            Console.WriteLine("{0}", di.Name)
        Next

    End Sub

End Module
```

The following example uses the [DirectoryInfo.EnumerateFiles](#) method to list all files whose [Length](#) exceeds 10MB. This example first enumerates the top-level directories, to catch possible unauthorized access exceptions, and then enumerates the files.

```
using System;
using System.IO;
```

```
class Program
{
    static void Main(string[] args)
    {
        // Set a variable to the My Documents path.
        string docPath = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments);

        DirectoryInfo diTop = new DirectoryInfo(docPath);

        try
        {
            foreach (var fi in diTop.EnumerateFiles())
            {
                try
                {
                    // Display each file over 10 MB;
                    if (fi.Length > 10000000)
                    {
                        Console.WriteLine($"{fi.FullName}\t\t{fi.Length.ToString("N0")}");
                    }
                }
                catch (UnauthorizedAccessException unAuthTop)
                {
                    Console.WriteLine($"{unAuthTop.Message}");
                }
            }

            foreach (var di in diTop.EnumerateDirectories("*"))
            {
                try
                {
                    foreach (var fi in di.EnumerateFiles("*", SearchOption.AllDirectories))
                    {
                        try
                        {
                            // Display each file over 10 MB;
                            if (fi.Length > 10000000)
                            {
                                Console.WriteLine($"{fi.FullName}\t\t{fi.Length.ToString("N0")}");
                            }
                        }
                        catch (UnauthorizedAccessException unAuthFile)
                        {
                            Console.WriteLine($"unAuthFile: {unAuthFile.Message}");
                        }
                    }
                }
                catch (UnauthorizedAccessException unAuthSubDir)
                {
                    Console.WriteLine($"unAuthSubDir: {unAuthSubDir.Message}");
                }
            }

            catch (DirectoryNotFoundException dirNotFound)
            {
                Console.WriteLine($"{dirNotFound.Message}");
            }
            catch (UnauthorizedAccessException unAuthDir)
            {
                Console.WriteLine($"unAuthDir: {unAuthDir.Message}");
            }
            catch (PathTooLongException longPath)
            {
                Console.WriteLine($"{longPath.Message}");
            }
        }
    }
}
```

```

Imports System.IO

Class Program
    Public Shared Sub Main(ByVal args As String())
        Dim dirPath As String = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments)
        Dim diTop As New DirectoryInfo(dirPath)
        Try
            For Each fi In diTop.EnumerateFiles()
                Try
                    ' Display each file over 10 MB;
                    If fi.Length > 10000000 Then
                        Console.WriteLine("{0}" & vbTab & "{1}", fi.FullName,
fi.Length.ToString("N0"))
                    End If
                Catch unAuthTop As UnauthorizedAccessException
                    Console.WriteLine($"{unAuthTop.Message}")
                End Try
            Next

            For Each di In diTop.EnumerateDirectories("*")
                Try
                    For Each fi In di.EnumerateFiles("*", SearchOption.AllDirectories)
                        Try
                            ' // Display each file over 10 MB;
                            If fi.Length > 10000000 Then
                                Console.WriteLine("{0}" & vbTab &
vbTab & "{1}", fi.FullName, fi.Length.ToString("N0"))
                            End If
                        Catch unAuthFile As UnauthorizedAccessException
                            Console.WriteLine($"{unAuthFile.Message}")
                        End Try
                    Next
                Catch unAuthSubDir As UnauthorizedAccessException
                    Console.WriteLine($"{unAuthSubDir.Message}")
                End Try
            Next

            Catch dirNotFound As DirectoryNotFoundException
                Console.WriteLine($"{dirNotFound.Message}")
            Catch unAuthDir As UnauthorizedAccessException
                Console.WriteLine($"{unAuthDir.Message}")
            Catch longPath As PathTooLongException
                Console.WriteLine($"{longPath.Message}")
            End Try
        End Sub
    End Class

```

See also

- [File and stream I/O](#)

How to: Read and write to a newly created data file

9/20/2022 • 2 minutes to read • [Edit Online](#)

The [System.IO.BinaryWriter](#) and [System.IO.BinaryReader](#) classes are used for writing and reading data other than character strings. The following example shows how to create an empty file stream, write data to it, and read data from it.

The example creates a data file called *Test.data* in the current directory, creates the associated [BinaryWriter](#) and [BinaryReader](#) objects, and uses the [BinaryWriter](#) object to write the integers 0 through 10 to *Test.data*, which leaves the file pointer at the end of the file. The [BinaryReader](#) object then sets the file pointer back to the origin and reads out the specified content.

NOTE

If *Test.data* already exists in the current directory, an [IOException](#) exception is thrown. Use the file mode option [FileMode.Create](#) rather than [FileMode.CreateNew](#) to always create a new file without throwing an exception.

Example

```
using System;
using System.IO;

class MyStream
{
    private const string FILE_NAME = "Test.data";

    public static void Main()
    {
        if (File.Exists(FILE_NAME))
        {
            Console.WriteLine($"{FILE_NAME} already exists!");
            return;
        }

        using (FileStream fs = new FileStream(FILE_NAME, FileMode.CreateNew))
        {
            using (BinaryWriter w = new BinaryWriter(fs))
            {
                for (int i = 0; i < 11; i++)
                {
                    w.Write(i);
                }
            }
        }

        using (FileStream fs = new FileStream(FILE_NAME, FileMode.Open, FileAccess.Read))
        {
            using (BinaryReader r = new BinaryReader(fs))
            {
                for (int i = 0; i < 11; i++)
                {
                    Console.WriteLine(r.ReadInt32());
                }
            }
        }
    }
}

// The example creates a file named "Test.data" and writes the integers 0 through 10 to it in binary format.
// It then writes the contents of Test.data to the console with each integer on a separate line.
```

```

Imports System.IO

Class MyStream
    Private Const FILE_NAME As String = "Test.data"

    Public Shared Sub Main()
        If File.Exists(FILE_NAME) Then
            Console.WriteLine($"{FILE_NAME} already exists!")
            Return
        End If

        Using fs As New FileStream(FILE_NAME, FileMode.CreateNew)
            Using w As New BinaryWriter(fs)
                For i As Integer = 0 To 10
                    w.Write(i)
                Next
            End Using
        End Using

        Using fs As New FileStream(FILE_NAME, FileMode.Open, FileAccess.Read)
            Using r As New BinaryReader(fs)
                For i As Integer = 0 To 10
                    Console.WriteLine(r.ReadInt32())
                Next
            End Using
        End Using
    End Sub
End Class

```

' The example creates a file named "Test.data" and writes the integers 0 through 10 to it in binary format.
' It then writes the contents of Test.data to the console with each integer on a separate line.

See also

- [BinaryReader](#)
- [BinaryWriter](#)
- [FileStream](#)
- [FileStream.Seek](#)
- [SeekOrigin](#)
- [How to: Enumerate directories and files](#)
- [How to: Open and append to a log file](#)
- [How to: Read text from a file](#)
- [How to: Write text to a file](#)
- [How to: Read characters from a string](#)
- [How to: Write characters to a string](#)
- [File and stream I/O](#)

How to: Open and append to a log file

9/20/2022 • 2 minutes to read • [Edit Online](#)

[StreamWriter](#) and [StreamReader](#) write characters to and read characters from streams. The following code example opens the *log.txt* file for input, or creates it if it doesn't exist, and appends log information to the end of the file. The example then writes the contents of the file to standard output for display.

As an alternative to this example, you could store the information as a single string or string array, and use the [File.WriteAllText](#) or [File.WriteAllLines](#) method to achieve the same functionality.

NOTE

Visual Basic users may choose to use the methods and properties provided by the [Log](#) class or [FileSystem](#) class for creating or writing to log files.

Example

```

using System;
using System.IO;

class DirAppend
{
    public static void Main()
    {
        using (StreamWriter w = File.AppendText("log.txt"))
        {
            Log("Test1", w);
            Log("Test2", w);
        }

        using (StreamReader r = File.OpenText("log.txt"))
        {
            DumpLog(r);
        }
    }

    public static void Log(string logMessage, TextWriter w)
    {
        w.WriteLine("\r\nLog Entry : ");
        w.WriteLine($"{DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss")}, {DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss")}");
        w.WriteLine(" :");
        w.WriteLine($" :{logMessage}");
        w.WriteLine ("-----");
    }

    public static void DumpLog(StreamReader r)
    {
        string line;
        while ((line = r.ReadLine()) != null)
        {
            Console.WriteLine(line);
        }
    }
}

// The example creates a file named "log.txt" and writes the following lines to it,
// or appends them to the existing "log.txt" file:

// Log Entry : <current long time string> <current long date string>
// :
// :Test1
// ----

// Log Entry : <current long time string> <current long date string>
// :
// :Test2
// ----

// It then writes the contents of "log.txt" to the console.

```

```

Imports System.IO

Class DirAppend
    Public Shared Sub Main()
        Using w As StreamWriter = File.AppendText("log.txt")
            Log("Test1", w)
            Log("Test2", w)
        End Using

        Using r As StreamReader = File.OpenText("log.txt")
            DumpLog(r)
        End Using
    End Sub

    Public Shared Sub Log(logMessage As String, w As TextWriter)
        w.WriteLine(vbCrLf + "Log Entry : ")
        w.WriteLine($"{DateTime.Now.ToString("yyyy-MM-ddTHH:mm:ss.fff")} {DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss.fff")}")
        w.WriteLine(" :")
        w.WriteLine($" :{logMessage}")
        w.WriteLine("-----")
    End Sub

    Public Shared Sub DumpLog(r As StreamReader)
        Dim line As String
        line = r.ReadLine()
        While Not (line Is Nothing)
            Console.WriteLine(line)
            line = r.ReadLine()
        End While
    End Sub
End Class

' The example creates a file named "log.txt" and writes the following lines to it,
' or appends them to the existing "log.txt" file:

' Log Entry : <current long time string> <current long date string>
' :
' :Test1
' ----

' Log Entry : <current long time string> <current long date string>
' :
' :Test2
' ----

' It then writes the contents of "log.txt" to the console.

```

See also

- [StreamWriter](#)
- [StreamReader](#)
- [File.AppendText](#)
- [File.OpenText](#)
- [StreamReader.ReadLine](#)
- [How to: Enumerate directories and files](#)
- [How to: Read and write to a newly created data file](#)
- [How to: Read text from a file](#)
- [How to: Write text to a file](#)
- [How to: Read characters from a string](#)
- [How to: Write characters to a string](#)

- File and stream I/O

How to: Write text to a file

9/20/2022 • 5 minutes to read • [Edit Online](#)

This topic shows different ways to write text to a file for a .NET app.

The following classes and methods are typically used to write text to a file:

- [StreamWriter](#) contains methods to write to a file synchronously ([Write](#) and [WriteLine](#)) or asynchronously ([WriteAsync](#) and [WriteLineAsync](#)).
- [File](#) provides static methods to write text to a file, such as [WriteAllLines](#) and [WriteAllText](#), or to append text to a file, such as [AppendAllLines](#), [AppendAllText](#), and [AppendText](#).
- [Path](#) is for strings that have file or directory path information. It contains the [Combine](#) method and, in .NET Core 2.1 and later, the [Join](#) and [TryJoin](#) methods, which allow concatenation of strings to build a file or directory path.

NOTE

The following examples show only the minimum amount of code needed. A real-world app usually provides more robust error checking and exception handling.

Example: Synchronously write text with StreamWriter

The following example shows how to use the [StreamWriter](#) class to synchronously write text to a new file one line at a time. Because the [StreamWriter](#) object is declared and instantiated in a [using](#) statement, the [Dispose](#) method is invoked, which automatically flushes and closes the stream.

```
using System;
using System.IO;

class Program
{
    static void Main(string[] args)
    {

        // Create a string array with the lines of text
        string[] lines = { "First line", "Second line", "Third line" };

        // Set a variable to the Documents path.
        string docPath =
            Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments);

        // Write the string array to a new file named "WriteLines.txt".
        using (StreamWriter outputFile = new StreamWriter(Path.Combine(docPath, "WriteLines.txt")))
        {
            foreach (string line in lines)
                outputFile.WriteLine(line);
        }
    }

    // The example creates a file named "WriteLines.txt" with the following contents:
    // First line
    // Second line
    // Third line
}
```

```

Imports System.IO

Class WriteText

    Public Shared Sub Main()

        ' Create a string array with the lines of text
        Dim lines() As String = {"First line", "Second line", "Third line"}

        ' Set a variable to the Documents path.
        Dim docPath As String = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments)

        ' Write the string array to a new file named "WriteLines.txt".
        Using outputFile As New StreamWriter(Path.Combine(docPath, Convert.ToString("WriteLines.txt")))
            For Each line As String In lines
                outputFile.WriteLine(line)
            Next
        End Using

    End Sub

End Class

' The example creates a file named "WriteLines.txt" with the following contents:
' First line
' Second line
' Third line

```

If you would like to see code comments translated to languages other than English, let us know in [this GitHub discussion issue](#).

Example: Synchronously append text with StreamWriter

The following example shows how to use the [StreamWriter](#) class to synchronously append text to the text file created in the first example.

```

using System;
using System.IO;

class Program
{
    static void Main(string[] args)
    {

        // Set a variable to the Documents path.
        string docPath = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments);

        // Append text to an existing file named "WriteLines.txt".
        using (StreamWriter outputFile = new StreamWriter(Path.Combine(docPath, "WriteLines.txt"), true))
        {
            outputFile.WriteLine("Fourth Line");
        }
    }
    // The example adds the following line to the contents of "WriteLines.txt":
    // Fourth Line

```

```

Imports System.IO

Class AppendText

    Public Shared Sub Main()

        ' Set a variable to the Documents path.
        Dim docPath As String =
            Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments)

        ' Append text to an existing file named "WriteLines.txt".
        Using outputFile As New StreamWriter(Path.Combine(docPath, Convert.ToString("WriteLines.txt")),
True)
            outputFile.WriteLine("Fourth Line")
        End Using

    End Sub

End Class

' The example adds the following line to the contents of "WriteLines.txt":
' Fourth Line

```

Example: Asynchronously write text with StreamWriter

The following example shows how to asynchronously write text to a new file using the [StreamWriter](#) class. To invoke the [WriteAsync](#) method, the method call must be within an `async` method. The C# example requires C# 7.1 or later, which adds support for the `async` modifier on the program entry point.

```

using System;
using System.IO;
using System.Threading.Tasks;

class Program
{
    static async Task Main()
    {
        // Set a variable to the Documents path.
        string docPath = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments);

        // Write the specified text asynchronously to a new file named "WriteTextAsync.txt".
        using (StreamWriter outputFile = new StreamWriter(Path.Combine(docPath, "WriteTextAsync.txt")))
        {
            await outputFile.WriteAsync("This is a sentence.");
        }
    }
    // The example creates a file named "WriteTextAsync.txt" with the following contents:
    // This is a sentence.
}

```

```

Imports System.IO

Public Module Example
    Public Sub Main()
        WriteTextAsync()
    End Sub

    Async Sub WriteTextAsync()
        ' Set a variable to the Documents path.
        Dim docPath As String = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments)

        ' Write the text asynchronously to a new file named "WriteTextAsync.txt".
        Using outputFile As New StreamWriter(Path.Combine(docPath, Convert.ToString("WriteTextAsync.txt")))
            Await outputFile.WriteLine("This is a sentence.")
        End Using
    End Sub
End Module

' The example creates a file named "WriteTextAsync.txt" with the following contents:
' This is a sentence.

```

Example: Write and append text with the File class

The following example shows how to write text to a new file and append new lines of text to the same file using the [File](#) class. The [WriteAllText](#) and [AppendAllLines](#) methods open and close the file automatically. If the path you provide to the [WriteAllText](#) method already exists, the file is overwritten.

```

using System;
using System.IO;

class Program
{
    static void Main(string[] args)
    {
        // Create a string with a line of text.
        string text = "First line" + Environment.NewLine;

        // Set a variable to the Documents path.
        string docPath = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments);

        // Write the text to a new file named "WriteFile.txt".
        File.WriteAllText(Path.Combine(docPath, "WriteFile.txt"), text);

        // Create a string array with the additional lines of text
        string[] lines = { "New line 1", "New line 2" };

        // Append new lines of text to the file
        File.AppendAllLines(Path.Combine(docPath, "WriteFile.txt"), lines);
    }
}

// The example creates a file named "WriteFile.txt" with the contents:
// First line
// And then appends the following contents:
// New line 1
// New line 2

```

```
Imports System.IO

Class WriteFile

    Public Shared Sub Main()

        ' Create a string array with the lines of text
        Dim text As String = "First line" & Environment.NewLine

        ' Set a variable to the Documents path.
        Dim docPath As String = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments)

        ' Write the text to a new file named "WriteFile.txt".
        File.WriteAllText(Path.Combine(docPath, Convert.ToString("WriteFile.txt")), text)

        ' Create a string array with the additional lines of text
        Dim lines() As String = {"New line 1", "New line 2"}

        ' Append new lines of text to the file
        File.AppendAllLines(Path.Combine(docPath, Convert.ToString("WriteFile.txt")), lines)

    End Sub

End Class

' The example creates a file named "WriteFile.txt" with the following contents:
' First line
' And then appends the following contents:
' New line 1
' New line 2
```

See also

- [StreamWriter](#)
- [Path](#)
- [File.CreateText](#)
- [How to: Enumerate directories and files](#)
- [How to: Read and write to a newly-created data file](#)
- [How to: Open and append to a log file](#)
- [How to: Read text from a file](#)
- [File and stream I/O](#)

How to: Read text from a file

9/20/2022 • 2 minutes to read • [Edit Online](#)

The following examples show how to read text synchronously and asynchronously from a text file using .NET for desktop apps. In both examples, when you create the instance of the [StreamReader](#) class, you provide the relative or absolute path to the file.

NOTE

These code examples do not apply to Universal Windows (UWP) apps, because the Windows Runtime provides different stream types for reading and writing to files. For an example that shows how to read text from a file in a UWP app, see [Quickstart: Reading and writing files](#). For examples that show how to convert between .NET Framework streams and Windows Runtime streams, see [How to: Convert between .NET Framework streams and Windows Runtime streams](#).

Example: Synchronous read in a console app

The following example shows a synchronous read operation within a console app. This example opens the text file using a stream reader, copies the contents to a string, and outputs the string to the console.

IMPORTANT

The example assumes that a file named *TestFile.txt* already exists in the same folder as the app.

```
using System;
using System.IO;

class Program
{
    public static void Main()
    {
        try
        {
            // Open the text file using a stream reader.
            using (var sr = new StreamReader("TestFile.txt"))
            {
                // Read the stream as a string, and write the string to the console.
                Console.WriteLine(sr.ReadToEnd());
            }
        }
        catch (IOException e)
        {
            Console.WriteLine("The file could not be read:");
            Console.WriteLine(e.Message);
        }
    }
}
```

```

Imports System.IO

Module Program
    Public Sub Main()
        Try
            ' Open the file using a stream reader.
            Using sr As New StreamReader("TestFile.txt")
                ' Read the stream as a string and write the string to the console.
                Console.WriteLine(sr.ReadToEnd())
            End Using
        Catch e As IOException
            Console.WriteLine("The file could not be read:")
            Console.WriteLine(e.Message)
        End Try
    End Sub
End Module

```

Example: Asynchronous read in a WPF app

The following example shows an asynchronous read operation in a Windows Presentation Foundation (WPF) app.

IMPORTANT

The example assumes that a file named *TestFile.txt* already exists in the same folder as the app.

```

using System.IO;
using System.Windows;

namespace TextFiles;

/// <summary>
/// Interaction logic for MainWindow.xaml
/// </summary>
public partial class MainWindow : Window
{
    public MainWindow() => InitializeComponent();

    private async void MainWindow_Loaded(object sender, RoutedEventArgs e)
    {
        try
        {
            using (var sr = new StreamReader("TestFile.txt"))
            {
                ResultBlock.Text = await sr.ReadToEndAsync();
            }
        }
        catch (FileNotFoundException ex)
        {
            ResultBlock.Text = ex.Message;
        }
    }
}

```

```
Imports System.IO
Imports System.Windows

''' <summary>
''' Interaction logic for MainWindow.xaml
''' </summary>

Partial Public Class MainWindow
    Inherits Window
    Public Sub New()
        InitializeComponent()
    End Sub

    Private Async Sub MainWindow_Loaded(sender As Object, e As RoutedEventArgs)
        Try
            Using sr As New StreamReader("TestFile.txt")
                ResultBlock.Text = Await sr.ReadToEndAsync()
            End Using
        Catch ex As FileNotFoundException
            ResultBlock.Text = ex.Message
        End Try
    End Sub
End Class
```

See also

- [StreamReader](#)
- [File.OpenText](#)
- [StreamReader.ReadLine](#)
- [Asynchronous file I/O](#)
- [How to: Create a directory listing](#)
- [Quickstart: Reading and writing files](#)
- [How to: Convert between .NET Framework streams and Windows Runtime streams](#)
- [How to: Read and write to a newly created data file](#)
- [How to: Open and append to a log file](#)
- [How to: Write text to a file](#)
- [How to: Read characters from a string](#)
- [How to: Write characters to a string](#)
- [File and stream I/O](#)

How to: Read characters from a string

9/20/2022 • 2 minutes to read • [Edit Online](#)

The following code examples show how to read characters synchronously or asynchronously from a string.

Example: Read characters synchronously

This example reads 13 characters synchronously from a string, stores them in an array, and displays them. The example then reads the rest of the characters in the string, stores them in the array starting at the sixth element, and displays the contents of the array.

```
using System;
using System.IO;

public class CharsFromStr
{
    public static void Main()
    {
        string str = "Some number of characters";
        char[] b = new char[str.Length];

        using (StringReader sr = new StringReader(str))
        {
            // Read 13 characters from the string into the array.
            sr.Read(b, 0, 13);
            Console.WriteLine(b);

            // Read the rest of the string starting at the current string position.
            // Put in the array starting at the 6th array member.
            sr.Read(b, 5, str.Length - 13);
            Console.WriteLine(b);
        }
    }
}

// The example has the following output:
//
// Some number o
// Some f characters
```

```
Imports System.IO

Public Class CharsFromStr
    Public Shared Sub Main()
        Dim str As String = "Some number of characters"
        Dim b(str.Length - 1) As Char

        Using sr As StringReader = New StringReader(str)
            ' Read 13 characters from the string into the array.
            sr.Read(b, 0, 13)
            Console.WriteLine(b)

            ' Read the rest of the string starting at the current string position.
            ' Put in the array starting at the 6th array member.
            sr.Read(b, 5, str.Length - 13)
            Console.WriteLine(b)
        End Using
    End Sub
End Class
' The example has the following output:
'
' Some number o
' Some f characters
```

Example: Read characters asynchronously

The next example is the code behind a WPF app. On window load, the example asynchronously reads all characters from a [TextBox](#) control and stores them in an array. It then asynchronously writes each letter or white-space character to a separate line of a [TextBlock](#) control.

```
using System;
using System.Text;
using System.Windows;
using System.IO;

namespace StringReaderWriter
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }

        private async void Window_Loaded(object sender, RoutedEventArgs e)
        {
            char[] charsRead = new char[UserInput.Text.Length];
            using (StringReader reader = new StringReader(UserInput.Text))
            {
                await reader.ReadAsync(charsRead, 0, UserInput.Text.Length);
            }

            StringBuilder reformattedText = new StringBuilder();
            using (StringWriter writer = new StringWriter(reformattedText))
            {
                foreach (char c in charsRead)
                {
                    if (char.IsLetter(c) || char.IsWhiteSpace(c))
                    {
                        await writer.WriteLineAsync(char.ToLower(c));
                    }
                }
            }
            Result.Text = reformattedText.ToString();
        }
    }
}
```

```
Imports System.IO
Imports System.Text

''' <summary>
''' Interaction logic for MainWindow.xaml
''' </summary>

Partial Public Class MainWindow
    Inherits Window
    Public Sub New()
        InitializeComponent()
    End Sub
    Private Async Sub Window_Loaded(sender As Object, e As RoutedEventArgs)
        Dim charsRead As Char() = New Char(UserInput.Text.Length) {}
        Using reader As StringReader = New StringReader(UserInput.Text)
            Await reader.ReadAsync(charsRead, 0, UserInput.Text.Length)
        End Using

        Dim reformattedText As StringBuilder = New StringBuilder()
        Using writer As StringWriter = New StringWriter(reformattedText)
            For Each c As Char In charsRead
                If Char.IsLetter(c) Or Char.IsWhiteSpace(c) Then
                    Await writer.WriteLineAsync(Char.ToLower(c))
                End If
            Next
        End Using
        Result.Text = reformattedText.ToString()
    End Sub
End Class
```

See also

- [StringReader](#)
- [StringReader.Read](#)
- [Asynchronous file I/O](#)
- [How to: Create a directory listing](#)
- [How to: Read and write to a newly created data file](#)
- [How to: Open and append to a log file](#)
- [How to: Read text from a file](#)
- [How to: Write text to a file](#)
- [How to: Write characters to a string](#)
- [File and stream I/O](#)

How to: Write characters to a string

9/20/2022 • 2 minutes to read • [Edit Online](#)

The following code examples write characters synchronously or asynchronously from a character array into a string.

Example: Write characters synchronously in a console app

The following example uses a [StringWriter](#) to write five characters synchronously to a [StringBuilder](#) object.

```
using System;
using System.IO;
using System.Text;

public class CharsToStr
{
    public static void Main()
    {
        StringBuilder sb = new StringBuilder("Start with a string and add from ");
        char[] b = { 'c', 'h', 'a', 'r', '.', ' ', 'B', 'u', 't', ' ', 'n', 'o', 't', ' ', 'a', 'l', 'l' };

        using (StringWriter sw = new StringWriter(sb))
        {
            // Write five characters from the array into the StringBuilder.
            sw.Write(b, 0, 5);
            Console.WriteLine(sb);
        }
    }
}

// The example has the following output:
//
// Start with a string and add from char.
```

```
Imports System.IO
Imports System.Text

Public Class CharsToStr
    Public Shared Sub Main()
        Dim sb As New StringBuilder("Start with a string and add from ")
        Dim b() As Char = {"c", "h", "a", "r", ".", " ", "B", "u", "t", " ", "n", "o", "t", " ", "a", "l",
"l"}

        Using sw As StringWriter = New StringWriter(sb)
            ' Write five characters from the array into the StringBuilder.
            sw.Write(b, 0, 5)
            Console.WriteLine(sb)
        End Using
    End Sub
End Class

' The example has the following output:
'
' Start with a string and add from char.
```

Example: Write characters asynchronously in a WPF app

The next example is the code behind a WPF app. On window load, the example asynchronously reads all characters from a [TextBox](#) control and stores them in an array. It then asynchronously writes each letter or white-space character to a separate line of a [TextBlock](#) control.

```
using System;
using System.Text;
using System.Windows;
using System.IO;

namespace StringReaderWriter
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }

        private async void Window_Loaded(object sender, RoutedEventArgs e)
        {
            char[] charsRead = new char[UserInput.Text.Length];
            using (StringReader reader = new StringReader(UserInput.Text))
            {
                await reader.ReadAsync(charsRead, 0, UserInput.Text.Length);
            }

            StringBuilder reformattedText = new StringBuilder();
            using (StringWriter writer = new StringWriter(reformattedText))
            {
                foreach (char c in charsRead)
                {
                    if (char.IsLetter(c) || char.IsWhiteSpace(c))
                    {
                        await writer.WriteLineAsync(char.ToLower(c));
                    }
                }
            }
            Result.Text = reformattedText.ToString();
        }
    }
}
```

```

Imports System.IO
Imports System.Text

''' <summary>
''' Interaction logic for MainWindow.xaml
''' </summary>

Partial Public Class MainWindow
    Inherits Window
    Public Sub New()
        InitializeComponent()
    End Sub
    Private Async Sub Window_Loaded(sender As Object, e As RoutedEventArgs)
        Dim charsRead As Char() = New Char(UserInput.Text.Length) {}
        Using reader As StringReader = New StringReader(UserInput.Text)
            Await reader.ReadAsync(charsRead, 0, UserInput.Text.Length)
        End Using

        Dim reformattedText As StringBuilder = New StringBuilder()
        Using writer As StringWriter = New StringWriter(reformattedText)
            For Each c As Char In charsRead
                If Char.IsLetter(c) Or Char.IsWhiteSpace(c) Then
                    Await writer.WriteLineAsync(Char.ToLower(c))
                End If
            Next
        End Using
        Result.Text = reformattedText.ToString()
    End Sub
End Class

```

See also

- [StringWriter](#)
- [StringWriter:Write](#)
- [StringBuilder](#)
- [File and stream I/O](#)
- [Asynchronous file I/O](#)
- [How to: Enumerate directories and files](#)
- [How to: Read and write to a newly created data file](#)
- [How to: Open and append to a log file](#)
- [How to: Read text from a file](#)
- [How to: Write text to a file](#)
- [How to: Read characters from a string](#)

How to: Add or remove Access Control List entries (.NET Framework only)

9/20/2022 • 2 minutes to read • [Edit Online](#)

To add or remove Access Control List (ACL) entries to or from a file or directory, get the [FileSecurity](#) or [DirectorySecurity](#) object from the file or directory. Modify the object, and then apply it back to the file or directory.

Add or remove an ACL entry from a file

1. Call the [File.GetAccessControl](#) method to get a [FileSecurity](#) object that contains the current ACL entries of a file.
2. Add or remove ACL entries from the [FileSecurity](#) object returned from step 1.
3. To apply the changes, pass the [FileSecurity](#) object to the [File.SetAccessControl](#) method.

Add or remove an ACL entry from a directory

1. Call the [Directory.GetAccessControl](#) method to get a [DirectorySecurity](#) object that contains the current ACL entries of a directory.
2. Add or remove ACL entries from the [DirectorySecurity](#) object returned from step 1.
3. To apply the changes, pass the [DirectorySecurity](#) object to the [Directory.SetAccessControl](#) method.

Example

You must use a valid user or group account to run this example. The example uses a [File](#) object. Use the same procedure for the [FileInfo](#), [Directory](#), and [DirectoryInfo](#) classes.

```
using System;
using System.IO;
using System.Security.AccessControl;

namespace FileSystemExample
{
    class FileExample
    {
        public static void Main()
        {
            try
            {
                string fileName = "test.xml";

                Console.WriteLine("Adding access control entry for "
                    + fileName);

                // Add the access control entry to the file.
                AddFileSecurity(fileName, @"DomainName\AccountName",
                    FileSystemRights.ReadData, AccessControlType.Allow);

                Console.WriteLine("Removing access control entry from "
                    + fileName);

                // Remove the access control entry from the file.
            }
        }
    }
}
```

```

        RemoveFileSecurity(fileName, @"DomainName\AccountName",
                           FileSystemRights.ReadData, AccessControlType.Allow);

        Console.WriteLine("Done.");
    }
    catch (Exception e)
    {
        Console.WriteLine(e);
    }
}

// Adds an ACL entry on the specified file for the specified account.
public static void AddFileSecurity(string fileName, string account,
                                    FileSystemRights rights, AccessControlType controlType)
{
    // Get a FileSecurity object that represents the
    // current security settings.
    FileSecurity fSecurity = File.GetAccessControl(fileName);

    // Add the FileSystemAccessRule to the security settings.
    fSecurity.AddAccessRule(new FileSystemAccessRule(account,
                                                       rights, controlType));

    // Set the new access settings.
    File.SetAccessControl(fileName, fSecurity);
}

// Removes an ACL entry on the specified file for the specified account.
public static void RemoveFileSecurity(string fileName, string account,
                                      FileSystemRights rights, AccessControlType controlType)
{
    // Get a FileSecurity object that represents the
    // current security settings.
    FileSecurity fSecurity = File.GetAccessControl(fileName);

    // Remove the FileSystemAccessRule from the security settings.
    fSecurity.RemoveAccessRule(new FileSystemAccessRule(account,
                                                       rights, controlType));

    // Set the new access settings.
    File.SetAccessControl(fileName, fSecurity);
}
}
}

```

```

Imports System.IO
Imports System.Security.AccessControl

Module FileExample

Sub Main()
    Try
        Dim fileName As String = "test.xml"

        Console.WriteLine("Adding access control entry for " & fileName)

        ' Add the access control entry to the file.
        AddFileSecurity(fileName, "DomainName\AccountName", _
                        FileSystemRights.ReadData, AccessControlType.Allow)

        Console.WriteLine("Removing access control entry from " & fileName)

        ' Remove the access control entry from the file.

```

```

        RemoveFileSecurity(fileName, "DomainName\AccountName", _
            FileSystemRights.ReadData, AccessControlType.Allow)

        Console.WriteLine("Done.")
    Catch e As Exception
        Console.WriteLine(e)
    End Try

End Sub

' Adds an ACL entry on the specified file for the specified account.
Sub AddFileSecurity(ByVal fileName As String, ByVal account As String, _
    ByVal rights As FileSystemRights, ByVal controlType As AccessControlType)

    ' Get a FileSecurity object that represents the
    ' current security settings.
    Dim fSecurity As FileSecurity = File.GetAccessControl(fileName)

    ' Add the FileSystemAccessRule to the security settings.
    Dim accessRule As FileSystemAccessRule = _
        New FileSystemAccessRule(account, rights, controlType)

    fSecurity.AddAccessRule(accessRule)

    ' Set the new access settings.
    File.SetAccessControl(fileName, fSecurity)

End Sub

' Removes an ACL entry on the specified file for the specified account.
Sub RemoveFileSecurity(ByVal fileName As String, ByVal account As String, _
    ByVal rights As FileSystemRights, ByVal controlType As AccessControlType)

    ' Get a FileSecurity object that represents the
    ' current security settings.
    Dim fSecurity As FileSecurity = File.GetAccessControl(fileName)

    ' Remove the FileSystemAccessRule from the security settings.
    fSecurity.RemoveAccessRule(New FileSystemAccessRule(account, _
        rights, controlType))

    ' Set the new access settings.
    File.SetAccessControl(fileName, fSecurity)

End Sub
End Module

```

How to: Compress and extract files

9/20/2022 • 5 minutes to read • [Edit Online](#)

The [System.IO.Compression](#) namespace contains the following classes for compressing and decompressing files and streams. You also can use these types to read and modify the contents of a compressed file:

- [ZipFile](#)
- [ZipArchive](#)
- [ZipArchiveEntry](#)
- [DeflateStream](#)
- [GZipStream](#)

The following examples show some of the operations you can perform with compressed files. These examples require the following NuGet packages to be added to your project:

- [System.IO.Compression](#)
- [System.IO.Compression.ZipFile](#)

If you're using .NET Framework, add references to these two libraries to your project:

- [System.IO.Compression](#)
- [System.IO.Compression.FileSystem](#)

Example 1: Create and extract a .zip file

The following example shows how to create and extract a compressed *.zip* file by using the [ZipFile](#) class. The example compresses the contents of a folder into a new *.zip* file, and then extracts the file to a new folder.

To run the sample, create a *start* folder in your program folder and populate it with files to zip.

```
using System;
using System.IO.Compression;

class Program
{
    static void Main(string[] args)
    {
        string startPath = @".\start";
        string zipPath = @".\result.zip";
        string extractPath = @".\extract";

        ZipFile.CreateFromDirectory(startPath, zipPath);

        ZipFile.ExtractToDirectory(zipPath, extractPath);
    }
}
```

```
Imports System.IO.Compression

Module Module1

    Sub Main()
        Dim startPath As String = ".\start"
        Dim zipPath As String = ".\result.zip"
        Dim extractPath As String = ".\extract"

        ZipFile.CreateFromDirectory(startPath, zipPath)

        ZipFile.ExtractToDirectory(zipPath, extractPath)
    End Sub

End Module
```

Example 2: Extract specific file extensions

The following example iterates through the contents of an existing `.zip` file and extracts files with a `.txt` extension. It uses the `ZipArchive` class to access the `.zip` file, and the `ZipArchiveEntry` class to inspect the individual entries. The extension method `ExtractToFile` for the `ZipArchiveEntry` object is available in the `System.IO.Compression.ZipFileExtensions` class.

To run the sample, place a `.zip` file called `result.zip` in your program folder. When prompted, provide a folder name to extract to.

IMPORTANT

When unzipping files, you must look for malicious file paths, which can escape from the directory you unzip into. This is known as a path traversal attack. The following example demonstrates how to check for malicious file paths and provides a safe way to unzip.

```
using System;
using System.IO;
using System.IO.Compression;

class Program
{
    static void Main(string[] args)
    {
        string zipPath = @".\result.zip";

        Console.WriteLine("Provide path where to extract the zip file:");
        string extractPath = Console.ReadLine();

        // Normalizes the path.
        extractPath = Path.GetFullPath(extractPath);

        // Ensures that the last character on the extraction path
        // is the directory separator char.
        // Without this, a malicious zip file could try to traverse outside of the expected
        // extraction path.
        if (!extractPath.EndsWith(Path.DirectorySeparatorChar.ToString(), StringComparison.OrdinalIgnoreCase))
            extractPath += Path.DirectorySeparatorChar;

        using (ZipArchive archive = ZipFile.OpenRead(zipPath))
        {
            foreach (ZipArchiveEntry entry in archive.Entries)
            {
                if (entry.FullName.EndsWith(".txt", StringComparison.OrdinalIgnoreCase))
                {
                    // Gets the full path to ensure that relative segments are removed.
                    string destinationPath = Path.GetFullPath(Path.Combine(extractPath, entry.FullName));

                    // Ordinal match is safest, case-sensitive volumes can be mounted within volumes that
                    // are case-insensitive.
                    if (destinationPath.StartsWith(extractPath, StringComparison.OrdinalIgnoreCase))
                        entry.ExtractToFile(destinationPath);
                }
            }
        }
    }
}
```

```

Imports System.IO
Imports System.IO.Compression

Module Module1

Sub Main()
    Dim zipPath As String = ".\result.zip"

    Console.WriteLine("Provide path where to extract the zip file:")
    Dim extractPath As String = Console.ReadLine()

    ' Normalizes the path.
    extractPath = Path.GetFullPath(extractPath)

    ' Ensures that the last character on the extraction path
    ' is the directory separator char.
    ' Without this, a malicious zip file could try to traverse outside of the expected
    ' extraction path.
    If Not extractPath.EndsWith(Path.DirectorySeparatorChar.ToString(), StringComparison.OrdinalIgnoreCase) Then
        extractPath += Path.DirectorySeparatorChar
    End If

    Using archive As ZipArchive = ZipFile.OpenRead(zipPath)
        For Each entry As ZipArchiveEntry In archive.Entries
            If entry.FullName.EndsWith(".txt", StringComparison.OrdinalIgnoreCaseIgnoreCase) Then

                ' Gets the full path to ensure that relative segments are removed.
                Dim destinationPath As String = Path.GetFullPath(Path.Combine(extractPath,
entry.FullName))

                ' Ordinal match is safest, case-sensitive volumes can be mounted within volumes that
                ' are case-insensitive.
                If destinationPath.StartsWith(extractPath, StringComparison.OrdinalIgnoreCase) Then
                    entry.ExtractToFile(destinationPath)
                End If

            End If
        Next
    End Using
End Sub

End Module

```

Example 3: Add a file to an existing .zip file

The following example uses the [ZipArchive](#) class to access an existing `.zip` file, and adds a file to it. The new file gets compressed when you add it to the existing `.zip` file.

```

using System;
using System.IO;
using System.IO.Compression;

namespace ConsoleApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            using (FileStream zipToOpen = new FileStream(@"c:\users\exampleuser\release.zip",
FileMode.Open))
            {
                using (ZipArchive archive = new ZipArchive(zipToOpen, ZipArchiveMode.Update))
                {
                    ZipArchiveEntry readmeEntry = archive.CreateEntry("Readme.txt");
                    using (StreamWriter writer = new StreamWriter(readmeEntry.Open()))
                    {
                        writer.WriteLine("Information about this package.");
                        writer.WriteLine("=====");
                    }
                }
            }
        }
    }
}

```

```

Imports System.IO
Imports System.IO.Compression

Module Module1

Sub Main()
    Using zipToOpen As FileStream = New FileStream("c:\users\exampleuser\release.zip", FileMode.Open)
        Using archive As ZipArchive = New ZipArchive(zipToOpen, ZipArchiveMode.Update)
            Dim readmeEntry As ZipArchiveEntry = archive.CreateEntry("Readme.txt")
            Using writer As StreamWriter = New StreamWriter(readmeEntry.Open())
                writer.WriteLine("Information about this package.")
                writer.WriteLine("=====")
            End Using
        End Using
    End Using
End Sub

End Module

```

Example 4: Compress and decompress .gz files

You can also use the [GZipStream](#) and [DeflateStream](#) classes to compress and decompress data. They use the same compression algorithm. You can decompress [GZipStream](#) objects that are written to a .gz file by using many common tools. The following example shows how to compress and decompress a directory of files by using the [GZipStream](#) class:

```

using System;
using System.IO;
using System.IO.Compression;

public class Program
{
    private static string directoryPath = @"..\temp";
    public static void Main()
    {
        DirectoryInfo directorySelected = new DirectoryInfo(directoryPath);
        Compress(directorySelected);

        foreach (FileInfo fileToDecompress in directorySelected.GetFiles("*.gz"))
        {
            Decompress(fileToDecompress);
        }
    }

    public static void Compress(DirectoryInfo directorySelected)
    {
        foreach (FileInfo fileToCompress in directorySelected.GetFiles())
        {
            using (FileStream originalFileStream = fileToCompress.OpenRead())
            {
                if ((File.GetAttributes(fileToCompress.FullName) &
                    FileAttributes.Hidden) != FileAttributes.Hidden & fileToCompress.Extension != ".gz")
                {
                    using (FileStream compressedFileStream = File.Create(fileToCompress.FullName + ".gz"))
                    {
                        using (GZipStream compressionStream = new GZipStream(compressedFileStream,
                            CompressionMode.Compress))
                        {
                            originalFileStream.CopyTo(compressionStream);
                        }
                    }
                    FileInfo info = new FileInfo(directoryPath + Path.DirectorySeparatorChar +
fileToCompress.Name + ".gz");
                    Console.WriteLine($"Compressed {fileToCompress.Name} from
{fileToCompress.Length.ToString()} to {info.Length.ToString()} bytes.");
                }
            }
        }
    }

    public static void Decompress(FileInfo fileToDecompress)
    {
        using (FileStream originalFileStream = fileToDecompress.OpenRead())
        {
            string currentFileName = fileToDecompress.FullName;
            string newFileName = currentFileName.Remove(currentFileName.Length -
fileToDecompress.Extension.Length);

            using (FileStream decompressedFileStream = File.Create(newFileName))
            {
                using (GZipStream decompressionStream = new GZipStream(originalFileStream,
CompressionMode.Decompress))
                {
                    decompressionStream.CopyTo(decompressedFileStream);
                    Console.WriteLine($"Decompressed: {fileToDecompress.Name}");
                }
            }
        }
    }
}

```

```

Imports System.IO
Imports System.IO.Compression

Module Module1

    Private directoryPath As String = ".\temp"
    Public Sub Main()
        Dim directorySelected As New DirectoryInfo(directoryPath)
        Compress(directorySelected)

        For Each fileToDecompress As FileInfo In directorySelected.GetFiles("*.gz")
            Decompress(fileToDecompress)
        Next
    End Sub

    Public Sub Compress(directorySelected As DirectoryInfo)
        For Each fileToCompress As FileInfo In directorySelected.GetFiles()
            Using originalFileStream As FileStream = fileToCompress.OpenRead()
                If (File.GetAttributes(fileToCompress.FullName) And FileAttributes.Hidden) <>
                    FileAttributes.Hidden And fileToCompress.Extension <> ".gz" Then
                    Using compressedFileStream As FileStream = File.Create(fileToCompress.FullName & ".gz")
                        Using compressionStream As New GZipStream(compressedFileStream,
                            CompressionMode.Compress)

                            originalFileStream.CopyTo(compressionStream)
                        End Using
                    End Using
                    Dim info As New FileInfo(directoryPath & Path.DirectorySeparatorChar &
                        fileToCompress.Name & ".gz")
                    Console.WriteLine($"Compressed {fileToCompress.Name} from
{fileToCompress.Length.ToString()} to {info.Length.ToString()} bytes.")
                End If
            End Using
        Next
    End Sub

    Private Sub Decompress(ByVal fileToDecompress As FileInfo)
        Using originalFileStream As FileStream = fileToDecompress.OpenRead()
            Dim currentFileName As String = fileToDecompress.FullName
            Dim newFileName = currentFileName.Remove(currentFileName.Length -
                fileToDecompress.Extension.Length)

            Using decompressedFileStream As FileStream = File.Create(newFileName)
                Using decompressionStream As GZipStream = New GZipStream(originalFileStream,
                    CompressionMode.Decompress)
                    decompressionStream.CopyTo(decompressedFileStream)
                    Console.WriteLine($"Decompressed: {fileToDecompress.Name}")
                End Using
            End Using
        End Sub
    End Module

```

See also

- [ZipArchive](#)
- [ZipFile](#)
- [ZipArchiveEntry](#)
- [DeflateStream](#)
- [GZipStream](#)
- [File and stream I/O](#)

Compose streams

9/20/2022 • 3 minutes to read • [Edit Online](#)

A *backing store* is a storage medium, such as a disk or memory. Each different backing store implements its own stream as an implementation of the [Stream](#) class.

Each stream type reads and writes bytes from and to its given backing store. Streams that connect to backing stores are called *base streams*. Base streams have constructors with the parameters necessary to connect the stream to the backing store. For example, [FileStream](#) has constructors that specify a path parameter, which specifies how the file will be shared by processes.

The design of the [System.IO](#) classes provides simplified stream composition. You can attach base streams to one or more pass-through streams that provide the functionality you want. You can attach a reader or writer to the end of the chain, so the preferred types can be read or written easily.

The following code examples create a [FileStream](#) around the existing *MyFile.txt* in order to buffer *MyFile.txt*. Note that [FileStreams](#) are buffered by default.

IMPORTANT

The examples assume that a file named *MyFile.txt* already exists in the same folder as the app.

Example: Use StreamReader

The following example creates a [StreamReader](#) to read characters from the [FileStream](#), which is passed to the [StreamReader](#) as its constructor argument. [StreamReader.ReadLine](#) then reads until [StreamReader.Peek](#) finds no more characters.

```

using System;
using System.IO;

public class CompBuf
{
    private const string FILE_NAME = "MyFile.txt";

    public static void Main()
    {
        if (!File.Exists(FILE_NAME))
        {
            Console.WriteLine($"{FILE_NAME} does not exist!");
            return;
        }
        FileStream fsIn = new FileStream(FILE_NAME, FileMode.Open,
            FileAccess.Read, FileShare.Read);
        // Create an instance of StreamReader that can read
        // characters from the FileStream.
        using (StreamReader sr = new StreamReader(fsIn))
        {
            string input;
            // While not at the end of the file, read lines from the file.
            while (sr.Peek() > -1)
            {
                input = sr.ReadLine();
                Console.WriteLine(input);
            }
        }
    }
}

```

```

Imports System.IO

Public Class CompBuf
    Private Const FILE_NAME As String = "MyFile.txt"

    Public Shared Sub Main()
        If Not File.Exists(FILE_NAME) Then
            Console.WriteLine($"{FILE_NAME} does not exist!")
            Return
        End If
        Dim fsIn As New FileStream(FILE_NAME, FileMode.Open, _
            FileAccess.Read, FileShare.Read)
        ' Create an instance of StreamReader that can read
        ' characters from the FileStream.
        Using sr As New StreamReader(fsIn)
            Dim input As String
            ' While not at the end of the file, read lines from the file.
            While sr.Peek() > -1
                input = sr.ReadLine()
                Console.WriteLine(input)
            End While
        End Using
    End Sub
End Class

```

Example: Use BinaryReader

The following example creates a [BinaryReader](#) to read bytes from the [FileStream](#), which is passed to the [BinaryReader](#) as its constructor argument. [ReadByte](#) then reads until [PeekChar](#) finds no more bytes.

```

using System;
using System.IO;

public class ReadBuf
{
    private const string FILE_NAME = "MyFile.txt";

    public static void Main()
    {
        if (!File.Exists(FILE_NAME))
        {
            Console.WriteLine($"{FILE_NAME} does not exist.");
            return;
        }
        FileStream f = new FileStream(FILE_NAME, FileMode.Open,
            FileAccess.Read, FileShare.Read);
        // Create an instance of BinaryReader that can
        // read bytes from the FileStream.
        using (BinaryReader br = new BinaryReader(f))
        {
            byte input;
            // While not at the end of the file, read lines from the file.
            while (br.PeekChar() > -1 )
            {
                input = br.ReadByte();
                Console.WriteLine(input);
            }
        }
    }
}

```

```

Imports System.IO

Public Class ReadBuf
    Private Const FILE_NAME As String = "MyFile.txt"

    Public Shared Sub Main()
        If Not File.Exists(FILE_NAME) Then
            Console.WriteLine($"{FILE_NAME} does not exist.")
            Return
        End If
        Dim f As New FileStream(FILE_NAME, FileMode.Open, _
            FileAccess.Read, FileShare.Read)
        ' Create an instance of BinaryReader that can
        ' read bytes from the FileStream.
        Using br As New BinaryReader(f)
            Dim input As Byte
            ' While not at the end of the file, read lines from the file.
            While br.PeekChar() > -1
                input = br.ReadByte()
                Console.WriteLine(input)
            End While
        End Using
    End Sub
End Class

```

See also

- [StreamReader](#)
- [StreamReader.ReadLine](#)
- [StreamReader.Peek](#)
- [FileStream](#)

- [BinaryReader](#)
- [BinaryReader.ReadByte](#)
- [BinaryReader.PeekChar](#)

How to: Convert between .NET Framework and Windows Runtime streams (Windows only)

9/20/2022 • 6 minutes to read • [Edit Online](#)

.NET Framework for UWP apps is a subset of the full .NET Framework. Because of security and other requirements for UWP apps, you can't use the full set of .NET Framework APIs to open and read files. For more information, see [.NET for UWP apps overview](#). However, you may want to use .NET Framework APIs for other stream manipulation operations. To manipulate these streams, you can convert between a .NET Framework stream type, such as [MemoryStream](#) or [FileStream](#), and a Windows Runtime stream, such as [IInputStream](#), [IOOutputStream](#), or [IRandomAccessStream](#).

The [System.IO.WindowsRuntimeStreamExtensions](#) class contains methods that make these conversions easy. However, underlying differences between .NET Framework and Windows Runtime streams affect the results of using these methods, as described in the following sections:

Convert from a Windows Runtime to a .NET Framework stream

To convert from a Windows Runtime stream to a .NET Framework stream, use one of the following [System.IO.WindowsRuntimeStreamExtensions](#) methods:

- [WindowsRuntimeStreamExtensionsAsStream](#) converts a random-access stream in the Windows Runtime to a managed stream in .NET for UWP apps.
- [WindowsRuntimeStreamExtensionsAsStreamForWrite](#) converts an output stream in the Windows Runtime to a managed stream in .NET for UWP apps.
- [WindowsRuntimeStreamExtensionsAsStreamForRead](#) converts an input stream in the Windows Runtime to a managed stream in .NET for UWP apps.

The Windows Runtime offers stream types that support reading only, writing only, or reading and writing. These capabilities are maintained when you convert a Windows Runtime stream to a .NET Framework stream.

Furthermore, if you convert a Windows Runtime stream to a .NET Framework stream and back, you get the original Windows Runtime instance back.

It's best practice to use the conversion method that matches the capabilities of the Windows Runtime stream you want to convert. However, since [IRandomAccessStream](#) is readable and writeable (it implements both [IOOutputStream](#) and [IInputStream](#)), the conversion methods maintain the capabilities of the original stream. For example, using [WindowsRuntimeStreamExtensionsAsStreamForRead](#) to convert an [IRandomAccessStream](#) doesn't limit the converted .NET Framework stream to being readable. It's also writable.

Example: Convert Windows Runtime random-access to .NET Framework stream

To convert from a Windows Runtime random-access stream to a .NET Framework stream, use the [WindowsRuntimeStreamExtensionsAsStream](#) method.

The following code example prompts you to select a file, opens it with Windows Runtime APIs, and then converts it to a .NET Framework stream. It reads the stream and outputs it to a text block. You would typically manipulate the stream with .NET Framework APIs before outputting the results.

To run this example, create a UWP XAML app that contains a text block named `TextBlock1` and a button named

Button1. Associate the button click event with the `button1_Click` method shown in the example.

```
using System;
using System.IO;
using System.Runtime.InteropServices.WindowsRuntime;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Media.Imaging;
using Windows.Storage;
using System.Net.Http;
using Windows.Storage.Pickers;

private async void button1_Click(object sender, RoutedEventArgs e)
{
    // Create a file picker.
    FileOpenPicker picker = new FileOpenPicker();

    // Set properties on the file picker such as start location and the type
    // of files to display.
    picker.SuggestedStartLocation = PickerLocationId.DocumentsLibrary;
    picker.ViewMode = PickerViewMode.List;
    picker.FileTypeFilter.Add(".txt");

    // Show picker enabling user to pick one file.
    StorageFile result = await picker.PickSingleFileAsync();

    if (result != null)
    {
        try
        {
            // Retrieve the stream. This method returns a IRandomAccessStreamWithContentType.
            var stream = await result.OpenReadAsync();

            // Convert the stream to a .NET stream using AsStream, pass to a
            // StreamReader and read the stream.
            using (StreamReader sr = new StreamReader(stream.AsStream()))
            {
                TextBlock1.Text = sr.ReadToEnd();
            }
        }
        catch (Exception ex)
        {
            TextBlock1.Text = "Error occurred reading the file. " + ex.Message;
        }
    }
    else
    {
        TextBlock1.Text = "User did not pick a file";
    }
}
```

```

Imports System.IO
Imports System.Runtime.InteropServices.WindowsRuntime
Imports Windows.UI.Xaml
Imports Windows.UI.Xaml.Controls
Imports Windows.UI.Xaml.Media.Imaging
Imports Windows.Storage
Imports System.Net.Http
Imports Windows.Storage.Pickers

Private Async Sub button1_Click(sender As Object, e As RoutedEventArgs)
    ' Create a file picker.
    Dim picker As New FileOpenPicker()

    ' Set properties on the file picker such as start location and the type of files to display.
    picker.SuggestedStartLocation = PickerLocationId.DocumentsLibrary
    picker.ViewMode = PickerViewMode.List
    picker.FileTypeFilter.Add(".txt")

    ' Show picker that enable user to pick one file.
    Dim result As StorageFile = Await picker.PickSingleFileAsync()

    If result IsNot Nothing Then
        Try
            ' Retrieve the stream. This method returns a IRandomAccessStreamWithContentType.
            Dim stream = Await result.OpenReadAsync()

            ' Convert the stream to a .NET stream using AsStreamForRead, pass to a
            ' StreamReader and read the stream.
            Using sr As New StreamReader(stream.AsStream())
                TextBlock1.Text = sr.ReadToEnd()
            End Using
        Catch ex As Exception
            TextBlock1.Text = "Error occurred reading the file. " + ex.Message
        End Try
    Else
        TextBlock1.Text = "User did not pick a file"
    End If
End Sub

```

Convert from a .NET Framework to a Windows Runtime stream

To convert from a .NET Framework stream to a Windows Runtime stream, use one of the following [System.IO.WindowsRuntimeStreamExtensions](#) methods:

- [WindowsRuntimeStreamExtensions.AsInputStream](#) converts a managed stream in .NET for UWP apps to an input stream in the Windows Runtime.
- [WindowsRuntimeStreamExtensions.AsOutputStream](#) converts a managed stream in .NET for UWP apps to an output stream in the Windows Runtime.
- [WindowsRuntimeStreamExtensions.AsRandomAccessStream](#) converts a managed stream in .NET for UWP apps to a random-access stream that the Windows Runtime can use for reading or writing.

When you convert a .NET Framework stream to a Windows Runtime stream, the capabilities of the converted stream depend on the original stream. For example, if the original stream supports both reading and writing, and you call [WindowsRuntimeStreamExtensions.AsInputStream](#) to convert the stream, the returned type is an [IRandomAccessStream](#). [IRandomAccessStream](#) implements [IInputStream](#) and [IOutputStream](#), and supports reading and writing.

.NET Framework streams don't support cloning, even after conversion. If you convert a .NET Framework stream to a Windows Runtime stream and call [GetInputStreamAt](#) or [GetOutputStreamAt](#), which call [CloneStream](#), or if you call [CloneStream](#) directly, an exception occurs.

Example: Convert .NET Framework to Windows Runtime random-access stream

To convert from a .NET Framework stream to a Windows Runtime random-access stream, use the [AsRandomAccessStream](#) method, as shown in the following example:

IMPORTANT

Make sure that the .NET Framework stream you are using supports seeking, or copy it to a stream that does. You can use the [Stream.CanSeek](#) property to determine this.

To run this example, create a UWP XAML app that targets the .NET Framework 4.5.1 and contains a text block named `TextBlock2` and a button named `Button2`. Associate the button click event with the `button2_Click` method shown in the example.

```
using System;
using System.IO;
using System.Runtime.InteropServices.WindowsRuntime;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Media.Imaging;
using Windows.Storage;
using System.Net.Http;
using Windows.Storage.Pickers;

private async void button2_Click(object sender, RoutedEventArgs e)
{
    // Create an HttpClient and access an image as a stream.
    var client = new HttpClient();
    Stream stream = await client.GetStreamAsync("https://docs.microsoft.com/en-us/dotnet/images/hub/featured-1.png");
    // Create a .NET memory stream.
    var memStream = new MemoryStream();
    // Convert the stream to the memory stream, because a memory stream supports seeking.
    await stream.CopyToAsync(memStream);
    // Set the start position.
    memStream.Position = 0;
    // Create a new bitmap image.
    var bitmap = new BitmapImage();
    // Set the bitmap source to the stream, which is converted to a IRandomAccessStream.
    bitmap.SetSource(memStream.AsRandomAccessStream());
    // Set the image control source to the bitmap.
    image1.Source = bitmap;
}
```

```
Imports System.IO
Imports System.Runtime.InteropServices.WindowsRuntime
Imports Windows.UI.Xaml
Imports Windows.UI.Xaml.Controls
Imports Windows.UI.Xaml.Media.Imaging
Imports Windows.Storage
Imports System.Net.Http
Imports Windows.Storage.Pickers

Private Async Sub button2_Click(sender As Object, e As RoutedEventArgs)

    ' Create an HttpClient and access an image as a stream.
    Dim client = New HttpClient()
    Dim stream As Stream = Await client.GetStreamAsync("https://docs.microsoft.com/en-us/dotnet/images/hub/featured-1.png")
    ' Create a .NET memory stream.
    Dim memStream = New MemoryStream()

    ' Convert the stream to the memory stream, because a memory stream supports seeking.
    Await stream.CopyToAsync(memStream)

    ' Set the start position.
    memStream.Position = 0

    ' Create a new bitmap image.
    Dim bitmap = New BitmapImage()

    ' Set the bitmap source to the stream, which is converted to a IRandomAccessStream.
    bitmap.SetSource(memStream.AsRandomAccessStream())

    ' Set the image control source to the bitmap.
    image1.Source = bitmap
End Sub
```

See also

- [Quickstart: Read and write a file \(Windows\)](#)
- [.NET for Windows Store apps overview](#)
- [.NET for Windows Store apps APIs](#)

Asynchronous File I/O

9/20/2022 • 4 minutes to read • [Edit Online](#)

Asynchronous operations enable you to perform resource-intensive I/O operations without blocking the main thread. This performance consideration is particularly important in a Windows 8.x Store app or desktop app where a time-consuming stream operation can block the UI thread and make your app appear as if it is not working.

Starting with .NET Framework 4.5, the I/O types include `async` methods to simplify asynchronous operations. An `async` method contains `Async` in its name, such as `ReadAsync`, `WriteAsync`, `CopyToAsync`, `FlushAsync`, `ReadLineAsync`, and `ReadToEndAsync`. These `async` methods are implemented on stream classes, such as `Stream`, `FileStream`, and `MemoryStream`, and on classes that are used for reading from or writing to streams, such `TextReader` and `TextWriter`.

In .NET Framework 4 and earlier versions, you have to use methods such as `BeginRead` and `EndRead` to implement asynchronous I/O operations. These methods are still available in current .NET versions to support legacy code; however, the `async` methods help you implement asynchronous I/O operations more easily.

C# and Visual Basic each have two keywords for asynchronous programming:

- `Async` (Visual Basic) or `async` (C#) modifier, which is used to mark a method that contains an asynchronous operation.
- `Await` (Visual Basic) or `await` (C#) operator, which is applied to the result of an `async` method.

To implement asynchronous I/O operations, use these keywords in conjunction with the `async` methods, as shown in the following examples. For more information, see [Asynchronous programming with `async` and `await` \(C#\)](#) or [Asynchronous Programming with `Async` and `Await` \(Visual Basic\)](#).

The following example demonstrates how to use two `FileStream` objects to copy files asynchronously from one directory to another. Notice that the `Click` event handler for the `Button` control is marked with the `async` modifier because it calls an asynchronous method.

```

using System;
using System.Threading.Tasks;
using System.Windows;
using System.IO;

namespace WpfApplication
{
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }

        private async void Button_Click(object sender, RoutedEventArgs e)
        {
            string startDirectory = @"c:\Users\exampleuser\start";
            string endDirectory = @"c:\Users\exampleuser\end";

            foreach (string filename in Directory.EnumerateFiles(startDirectory))
            {
                using (FileStream sourceStream = File.Open(filename, FileMode.Open))
                {
                    using (FileStream destinationStream = File.Create(Path.Combine(endDirectory,
Path.GetFileName(filename))))
                    {
                        await sourceStream.CopyToAsync(destinationStream);
                    }
                }
            }
        }
    }
}

```

```

Imports System.IO

Class MainWindow

    Private Async Sub Button_Click(sender As Object, e As RoutedEventArgs)
        Dim StartDirectory As String = "c:\Users\exampleuser\start"
        Dim EndDirectory As String = "c:\Users\exampleuser\end"

        For Each filename As String In Directory.EnumerateFiles(StartDirectory)
            Using SourceStream As FileStream = File.Open(filename, FileMode.Open)
                Using DestinationStream As FileStream = File.Create(EndDirectory +
filename.Substring(filename.LastIndexOf("\\")))
                    Await SourceStream.CopyToAsync(DestinationStream)
                End Using

            End Using
        Next
    End Sub

End Class

```

The next example is similar to the previous one but uses [StreamReader](#) and [StreamWriter](#) objects to read and write the contents of a text file asynchronously.

```

private async void Button_Click(object sender, RoutedEventArgs e)
{
    string UserDirectory = @"c:\Users\exampleuser\";

    using (StreamReader SourceReader = File.OpenText(UserDirectory + "BigFile.txt"))
    {
        using (StreamWriter DestinationWriter = File.CreateText(UserDirectory + "CopiedFile.txt"))
        {
            await CopyFilesAsync(SourceReader, DestinationWriter);
        }
    }
}

public async Task CopyFilesAsync(StreamReader Source, StreamWriter Destination)
{
    char[] buffer = new char[0x1000];
    int numRead;
    while ((numRead = await Source.ReadAsync(buffer, 0, buffer.Length)) != 0)
    {
        await Destination.WriteAsync(buffer, 0, numRead);
    }
}

```

```

Private Async Sub Button_Click(sender As Object, e As RoutedEventArgs)
    Dim UserDirectory As String = "c:\Users\exampleuser\"

    Using SourceReader As StreamReader = File.OpenText(UserDirectory + "BigFile.txt")
        Using DestinationWriter As StreamWriter = File.CreateText(UserDirectory + "CopiedFile.txt")
            Await CopyFilesAsync(SourceReader, DestinationWriter)
        End Using
    End Using
End Sub

Public Async Function CopyFilesAsync(Source As StreamReader, Destination As StreamWriter) As Task
    Dim buffer(4095) As Char
    Dim numRead As Integer

    numRead = Await Source.ReadAsync(buffer, 0, buffer.Length)
    Do While numRead <> 0
        Await Destination.WriteAsync(buffer, 0, numRead)
        numRead = Await Source.ReadAsync(buffer, 0, buffer.Length)
    Loop
End Function

```

The next example shows the code-behind file and the XAML file that are used to open a file as a [Stream](#) in a Windows 8.x Store app, and read its contents by using an instance of the [StreamReader](#) class. It uses asynchronous methods to open the file as a stream and to read its contents.

```
using System;
using System.IO;
using System.Text;
using Windows.Storage.Pickers;
using Windows.Storage;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;

namespace ExampleApplication
{
    public sealed partial class MainPage : Page
    {
        public MainPage()
        {
            this.InitializeComponent();
        }

        private async void Button_Click_1(object sender, RoutedEventArgs e)
        {
            StringBuilder contents = new StringBuilder();
            string nextLine;
            int lineCounter = 1;

            var openPicker = new FileOpenPicker();
            openPicker.SuggestedStartLocation = PickerLocationId.DocumentsLibrary;
            openPicker.FileTypeFilter.Add(".txt");
            StorageFile selectedFile = await openPicker.PickSingleFileAsync();

            using (StreamReader reader = new StreamReader(await selectedFile.OpenStreamForReadAsync()))
            {
                while ((nextLine = await reader.ReadLineAsync()) != null)
                {
                    contents.AppendFormat("{0}. ", lineCounter);
                    contents.Append(nextLine);
                    contents.AppendLine();
                    lineCounter++;
                    if (lineCounter > 3)
                    {
                        contents.AppendLine("Only first 3 lines shown.");
                        break;
                    }
                }
            }
            DisplayContentsBlock.Text = contents.ToString();
        }
    }
}
```

```

Imports System.Text
Imports System.IO
Imports Windows.Storage.Pickers
Imports Windows.Storage

NotInheritable Public Class BlankPage
    Inherits Page


    Private Async Sub Button_Click_1(sender As Object, e As RoutedEventArgs)
        Dim contents As StringBuilder = New StringBuilder()
        Dim nextLine As String
        Dim lineCounter As Integer = 1

        Dim openPicker = New FileOpenPicker()
        openPicker.SuggestedStartLocation = PickerLocationId.DocumentsLibrary

        openPicker.FileTypeFilter.Add(".txt")
        Dim selectedFile As StorageFile = Await openPicker.PickSingleFileAsync()

        Using reader As StreamReader = New StreamReader(Await selectedFile.OpenStreamForReadAsync())
            nextLine = Await reader.ReadLineAsync()
            While (nextLine <> Nothing)
                contents.AppendFormat("{0}. ", lineCounter)
                contents.Append(nextLine)
                contents.AppendLine()
                lineCounter = lineCounter + 1
                If (lineCounter > 3) Then
                    contents.AppendLine("Only first 3 lines shown.")
                    Exit While
                End If
                nextLine = Await reader.ReadLineAsync()
            End While
        End Using
        DisplayContentsBlock.Text = contents.ToString()
    End Sub
End Class

```

```

<Page
    x:Class="ExampleApplication.BlankPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:ExampleApplication"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d">

    <StackPanel Background="{StaticResource ApplicationPageBackgroundBrush}" VerticalAlignment="Center"
    HorizontalAlignment="Center">
        <TextBlock Text="Display lines from a file."></TextBlock>
        <Button Content="Load File" Click="Button_Click_1"></Button>
        <TextBlock Name="DisplayContentsBlock"></TextBlock>
    </StackPanel>
</Page>

```

See also

- [Stream](#)
- [File and Stream I/O](#)
- [Asynchronous programming with async and await \(C#\)](#)
- [Asynchronous Programming with Async and Await \(Visual Basic\)](#)

Handling I/O errors in .NET

9/20/2022 • 5 minutes to read • [Edit Online](#)

In addition to the exceptions that can be thrown in any method call (such as an [OutOfMemoryException](#) when a system is stressed or an [NullReferenceException](#) due to programmer error), .NET file system methods can throw the following exceptions:

- [System.IO.IOException](#), the base class of all [System.IO](#) exception types. It is thrown for errors whose return codes from the operating system don't directly map to any other exception type.
- [System.IO.FileNotFoundException](#).
- [System.IO.DirectoryNotFoundException](#).
- [DriveNotFoundException](#).
- [System.IO.PathTooLongException](#).
- [System.OperationCanceledException](#).
- [System.UnauthorizedAccessException](#).
- [System.ArgumentException](#), which is thrown for invalid path characters on .NET Framework and on .NET Core 2.0 and previous versions.
- [System.NotSupportedException](#), which is thrown for invalid colons in .NET Framework.
- [System.Security.SecurityException](#), which is thrown for applications running in limited trust that lack the necessary permissions on .NET Framework only. (Full trust is the default on .NET Framework.)

Mapping error codes to exceptions

Because the file system is an operating system resource, I/O methods in both .NET Core and .NET Framework wrap calls to the underlying operating system. When an I/O error occurs in code executed by the operating system, the operating system returns error information to the .NET I/O method. The method then translates the error information, typically in the form of an error code, into a .NET exception type. In most cases, it does this by directly translating the error code into its corresponding exception type; it does not perform any special mapping of the error based on the context of the method call.

For example, on the Windows operating system, a method call that returns an error code of

`ERROR_FILE_NOT_FOUND` (or 0x02) maps to a [FileNotFoundException](#), and an error code of `ERROR_PATH_NOT_FOUND` (or 0x03) maps to a [DirectoryNotFoundException](#).

However, the precise conditions under which the operating system returns particular error codes is often undocumented or poorly documented. As a result, unexpected exceptions can occur. For example, because you are working with a directory rather than a file, you would expect that providing an invalid directory path to the [DirectoryInfo](#) constructor throws a [DirectoryNotFoundException](#). However, it may also throw a [FileNotFoundException](#).

Exception handling in I/O operations

Because of this reliance on the operating system, identical exception conditions (such as the directory not found error in our example) can result in an I/O method throwing any one of the entire class of I/O exceptions. This means that, when calling I/O APIs, your code should be prepared to handle most or all of these exceptions, as shown in the following table:

EXCEPTION TYPE	.NET CORE/.NET 5+	.NET FRAMEWORK
IOException	Yes	Yes
FileNotFoundException	Yes	Yes
DirectoryNotFoundException	Yes	Yes
DriveNotFoundException	Yes	Yes
PathTooLongException	Yes	Yes
OperationCanceledException	Yes	Yes
UnauthorizedAccessException	Yes	Yes
ArgumentException	.NET Core 2.0 and earlier	Yes
NotSupportedException	No	Yes
SecurityException	No	Limited trust only

Handling IOException

As the base class for exceptions in the [System.IO](#) namespace, [IOException](#) is also thrown for any error code that does not map to a predefined exception type. This means that it can be thrown by any I/O operation.

IMPORTANT

Because [IOException](#) is the base class of the other exception types in the [System.IO](#) namespace, you should handle in a `catch` block after you've handled the other I/O-related exceptions.

In addition, starting with .NET Core 2.1, validation checks for path correctness (for example, to ensure that invalid characters are not present in a path) have been removed, and the runtime throws an exception mapped from an operating system error code rather than from its own validation code. The most likely exception to be thrown in this case is an [IOException](#), although any other exception type could also be thrown.

Note that, in your exception handling code, you should always handle the [IOException](#) last. Otherwise, because it is the base class of all other IO exceptions, the catch blocks of derived classes will not be evaluated.

In the case of an [IOException](#), you can get additional error information from the [IOException.HResult](#) property. To convert the HResult value to a Win32 error code, you strip out the upper 16 bits of the 32-bit value. The following table lists error codes that may be wrapped in an [IOException](#).

HRESULT	CONSTANT	DESCRIPTION
ERROR_SHARING_VIOLATION	32	The file name is missing, or the file or directory is in use.
ERROR_FILE_EXISTS	80	The file already exists.
ERROR_INVALID_PARAMETER	87	An argument supplied to the method is invalid.

HRESULT	CONSTANT	DESCRIPTION
ERROR_ALREADY_EXISTS	183	The file or directory already exists.

You can handle these using a `When` clause in a catch statement, as the following example shows.

```

using System;
using System.IO;
using System.Text;

class Program
{
    static void Main()
    {
        var sw = OpenStream(@".\textfile.txt");
        if (sw is null)
            return;
        sw.WriteLine("This is the first line.");
        sw.WriteLine("This is the second line.");
        sw.Close();
    }

    static StreamWriter OpenStream(string path)
    {
        if (path is null) {
            Console.WriteLine("You did not supply a file path.");
            return null;
        }

        try {
            var fs = new FileStream(path, FileMode.CreateNew);
            return new StreamWriter(fs);
        }
        catch (FileNotFoundException) {
            Console.WriteLine("The file or directory cannot be found.");
        }
        catch (DirectoryNotFoundException) {
            Console.WriteLine("The file or directory cannot be found.");
        }
        catch (DriveNotFoundException) {
            Console.WriteLine("The drive specified in 'path' is invalid.");
        }
        catch (PathTooLongException) {
            Console.WriteLine("'path' exceeds the maximum supported path length.");
        }
        catch (UnauthorizedAccessException) {
            Console.WriteLine("You do not have permission to create this file.");
        }
        catch (IOException e) when ((e.HResult & 0x0000FFFF) == 32 ) {
            Console.WriteLine("There is a sharing violation.");
        }
        catch (IOException e) when ((e.HResult & 0x0000FFFF) == 80) {
            Console.WriteLine("The file already exists.");
        }
        catch (IOException e) {
            Console.WriteLine($"An exception occurred:\nError code: " +
                           $"{e.HResult & 0x0000FFFF}\nMessage: {e.Message}");
        }
        return null;
    }
}

```

```

Imports System.IO

Module Program
    Sub Main(args As String())
        Dim sw = OpenStream(".\textfile.txt")
        If sw Is Nothing Then Return

        sw.WriteLine("This is the first line.")
        sw.WriteLine("This is the second line.")
        sw.Close()
    End Sub

    Function OpenStream(path As String) As StreamWriter
        If path Is Nothing Then
            Console.WriteLine("You did not supply a file path.")
            Return Nothing
        End If

        Try
            Dim fs As New FileStream(path, FileMode.CreateNew)
            Return New StreamWriter(fs)
        Catch e As FileNotFoundException
            Console.WriteLine("The file or directory cannot be found.")
        Catch e As DirectoryNotFoundException
            Console.WriteLine("The file or directory cannot be found.")
        Catch e As DriveNotFoundException
            Console.WriteLine("The drive specified in 'path' is invalid.")
        Catch e As PathTooLongException
            Console.WriteLine("'path' exceeds the maximum supported path length.")
        Catch e As UnauthorizedAccessException
            Console.WriteLine("You do not have permission to create this file.")
        Catch e As IOException When (e.HResult And &h0000FFFF) = 32
            Console.WriteLine("There is a sharing violation.")
        Catch e As IOException When (e.HResult And &h0000FFFF) = 80
            Console.WriteLine("The file already exists.")
        Catch e As IOException
            Console.WriteLine($"An exception occurred:{vbCrLf}Error code: " +
                $"{e.HResult And &h0000FFFF}{vbCrLf}Message: {e.Message}")
        End Try
        Return Nothing
    End Function
End Module

```

See also

- [Handling and throwing exceptions in .NET](#)
- [Exception handling \(Task Parallel Library\)](#)
- [Best practices for exceptions](#)
- [How to use specific exceptions in a catch block](#)

Isolated storage

9/20/2022 • 17 minutes to read • [Edit Online](#)

For desktop apps, isolated storage is a data storage mechanism that provides isolation and safety by defining standardized ways of associating code with saved data. Standardization provides other benefits as well. Administrators can use tools designed to manipulate isolated storage to configure file storage space, set security policies, and delete unused data. With isolated storage, your code no longer needs unique paths to specify safe locations in the file system, and data is protected from other applications that only have isolated storage access. Hard-coded information that indicates where an application's storage area is located is unnecessary.

IMPORTANT

Isolated storage is not available for Windows 8.x Store apps. Instead, use the application data classes in the `Windows.Storage` namespaces included in the Windows Runtime API to store local data and files. For more information, see [Application data](#) in the Windows Dev Center.

Data Compartments and Stores

When an application stores data in a file, the file name and storage location must be carefully chosen to minimize the possibility that the storage location will be known to another application and, therefore, vulnerable to corruption. Without a standard system in place to manage these problems, improvising techniques that minimize storage conflicts can be complex, and the results can be unreliable.

With isolated storage, data is always isolated by user and by assembly. Credentials such as the origin or the strong name of the assembly determine assembly identity. Data can also be isolated by application domain, using similar credentials.

When you use isolated storage, your application saves data to a unique data compartment that is associated with some aspect of the code's identity, such as its publisher or signature. The data compartment is an abstraction, not a specific storage location; it consists of one or more isolated storage files, called stores, which contain the actual directory locations where data is stored. For example, an application might have a data compartment associated with it, and a directory in the file system would implement the store that actually preserves the data for that application. The data saved in the store can be any kind of data, from user preference information to application state. For the developer, the location of the data compartment is transparent. Stores usually reside on the client, but a server application could use isolated stores to store information by impersonating the user on whose behalf it is functioning. Isolated storage can also store information on a server with a user's roaming profile so that the information will travel with the roaming user.

Quotas for Isolated Storage

A quota is a limit on the amount of isolated storage that can be used. The quota includes bytes of file space as well as the overhead associated with the directory and other information in the store. Isolated storage uses permission quotas, which are storage limits that are set by using `IsolatedStoragePermission` objects. If you try to write data that exceeds the quota, an `IsolatedStorageException` exception is thrown. Security policy, which can be modified using the .NET Framework Configuration Tool (Mscorcfg.msc), determines which permissions are granted to code. Code that has been granted `IsolatedStoragePermission` is restricted to using no more storage than the `UserQuota` property allows. However, because code can bypass permission quotas by presenting different user identities, permission quotas serve as guidelines for how code should behave rather than as a

firm limit on code behavior.

Quotas are not enforced on roaming stores. Because of this, a slightly higher level of permission is required for code to use them. The enumeration values [AssemblyIsolationByRoamingUser](#) and [DomainIsolationByRoamingUser](#) specify a permission to use isolated storage for a roaming user.

Secure Access

Using isolated storage enables partially trusted applications to store data in a manner that is controlled by the computer's security policy. This is especially useful for downloaded components that a user might want to run cautiously. Security policy rarely grants this kind of code permission when you access the file system by using standard I/O mechanisms. However, by default, code running from the local computer, a local network, or the Internet is granted the right to use isolated storage.

Administrators can limit how much isolated storage an application or a user has available, based on an appropriate trust level. In addition, administrators can remove a user's persisted data completely. To create or access isolated storage, code must be granted the appropriate [IsolatedStorageFilePermission](#) permission.

To access isolated storage, code must have all necessary native platform operating system rights. The access control lists (ACLs) that control which users have the rights to use the file system must be satisfied. .NET applications already have operating system rights to access isolated storage unless they perform (platform-specific) impersonation. In this case, the application is responsible for ensuring that the impersonated user identity has the proper operating system rights to access isolated storage. This access provides a convenient way for code that is run or downloaded from the web to read and write to a storage area related to a particular user.

To control access to isolated storage, the common language runtime uses [IsolatedStorageFilePermission](#) objects. Each object has properties that specify the following values:

- Allowed usage, which indicates the type of access that is allowed. The values are members of the [IsolatedStorageContainment](#) enumeration. For more information about these values, see the table in the next section.
- Storage quota, as discussed in the preceding section.

The runtime demands [IsolatedStorageFilePermission](#) permission when code first attempts to open a store. It decides whether to grant this permission, based on how much the code is trusted. If the permission is granted, the allowed usage and storage quota values are determined by security policy and by the code's request for [IsolatedStorageFilePermission](#). Security policy is set by using the .NET Framework Configuration Tool (Mscorcfg.msc). All callers in the call stack are checked to ensure that each caller has at least the appropriate allowed usage. The runtime also checks the quota imposed on the code that opened or created the store in which the file is to be saved. If these conditions are satisfied, permission is granted. The quota is checked again every time a file is written to the store.

Application code is not required to request permission because the common language runtime will grant whatever [IsolatedStorageFilePermission](#) is appropriate based on security policy. However, there are good reasons to request specific permissions that your application needs, including [IsolatedStorageFilePermission](#).

Allowed Usage and Security Risks

The allowed usage specified by [IsolatedStorageFilePermission](#) determines the degree to which code will be allowed to create and use isolated storage. The following table shows how the allowed usage specified in the permission corresponds to types of isolation and summarizes the security risks associated with each allowed usage.

ALLOWED USAGE	ISOLATION TYPES	SECURITY IMPACT
None	No isolated storage use is allowed.	There is no security impact.
DomainIsolationByUser	Isolation by user, domain, and assembly. Each assembly has a separate substore within the domain. Stores that use this permission are also implicitly isolated by computer.	This permission level leaves resources open to unauthorized overuse, although enforced quotas make it more difficult. This is called a denial of service attack.
DomainIsolationByRoamingUser	Same as <code>DomainIsolationByUser</code> , but store is saved to a location that will roam if roaming user profiles are enabled and quotas are not enforced.	Because quotas must be disabled, storage resources are more vulnerable to a denial of service attack.
AssemblyIsolationByUser	Isolation by user and assembly. Stores that use this permission are also implicitly isolated by computer.	Quotas are enforced at this level to help prevent a denial of service attack. The same assembly in another domain can access this store, opening the possibility that information could be leaked between applications.
AssemblyIsolationByRoamingUser	Same as <code>AssemblyIsolationByUser</code> , but store is saved to a location that will roam if roaming user profiles are enabled and quotas are not enforced.	Same as in <code>AssemblyIsolationByUser</code> , but without quotas, the risk of a denial of service attack increases.
AdministerIsolatedStorageByUser	Isolation by user. Typically, only administrative or debugging tools use this level of permission.	Access with this permission allows code to view or delete any of a user's isolated storage files or directories (regardless of assembly isolation). Risks include, but are not limited to, leaking information and data loss.
UnrestrictedIsolatedStorage	Isolation by all users, domains, and assemblies. Typically, only administrative or debugging tools use this level of permission.	This permission creates the potential for a total compromise of all isolated stores for all users.

Safety of isolated storage components with regard to untrusted data

This section applies to the following frameworks:

- .NET Framework (all versions)
- .NET Core 2.1+
- .NET 5+

.NET Framework and .NET Core offer isolated storage as a mechanism to persist data for a user, an application, or a component. This is a legacy component primarily designed for now-deprecated Code Access Security scenarios.

Various isolated storage APIs and tools can be used to read data across trust boundaries. For example, reading data from a machine-wide scope can aggregate data from other, possibly less-trusted user accounts on the machine. Components or applications that read from machine-wide isolated storage scopes should be aware of the consequences of reading this data.

Security-sensitive APIs that can read from the machine-wide scope

Components or applications that call any of the following APIs read from the machine-wide scope:

- `IsolatedStorageFile.GetEnumerator`, passing a scope that includes the `IsolatedStorageScope.Machine` flag
- `IsolatedStorageFile.GetMachineStoreForApplication`
- `IsolatedStorageFile.GetMachineStoreForAssembly`
- `IsolatedStorageFile.GetMachineStoreForDomain`
- `IsolatedStorageFile.GetStore`, passing a scope that includes the `IsolatedStorageScope.Machine` flag
- `IsolatedStorageFile.Remove`, passing a scope that includes the `IsolatedStorageScope.Machine` flag

The isolated storage tool `storeadm.exe` is impacted if called with the `/machine` switch, as shown in the following code:

```
storeadm.exe /machine [any-other-switches]
```

The isolated storage tool is provided as part of Visual Studio and the .NET Framework SDK.

If the application doesn't involve calls to the preceding APIs, or if the workflow doesn't involve calling `storeadm.exe` in this manner, this document doesn't apply.

Impact in multi-user environments

As mentioned previously, the security impact from these APIs results from data written from one trust environment is read from a different trust environment. Isolated storage generally uses one of three locations to read and write data:

1. `%LOCALAPPDATA%\IsolatedStorage\` : For example, `C:\Users\<username>\AppData\Local\IsolatedStorage\`, for `User` scope.
2. `%APPDATA%\IsolatedStorage\` : For example, `C:\Users\<username>\AppData\Roaming\IsolatedStorage\`, for `User|Roaming` scope.
3. `%PROGRAMDATA%\IsolatedStorage\` : For example, `C:\ProgramData\IsolatedStorage\`, for `Machine` scope.

The first two locations are isolated per-user. Windows ensures that different user accounts on the same machine cannot access each other's user profile folders. Two different user accounts who use the `User` or `User|Roaming` stores will not see each other's data and cannot interfere with each other's data.

The third location is shared across all user accounts on the machine. Different accounts can read from and write to this location, and they're able to see each other's data.

The preceding paths may differ based on the version of Windows in use.

Now consider a multi-user system with two registered users *Mallory* and *Bob*. Mallory has the ability to access her user profile directory `C:\Users\Mallory\`, and she can access the shared machine-wide storage location `C:\ProgramData\IsolatedStorage\`. She cannot access Bob's user profile directory `C:\Users\Bob\`.

If Mallory wishes to attack Bob, she might write data to the machine-wide storage location, then attempt to influence Bob into reading from the machine-wide store. When Bob runs an app that reads from this store, that app will operate on the data Mallory placed there, but from within the context of Bob's user account. The remainder of this document contemplates various attack vectors and what steps apps can do to minimize their risk to these attacks.

NOTE

In order for such an attack to take place, Mallory requires:

- A user account on the machine.
- The ability to place a file into a known location on the file system.
- Knowledge that Bob will at some point run an app that attempts to read this data.

These are not threat vectors that apply to standard single-user desktop environments like home PCs or single-employee enterprise workstations.

Elevation of privilege

An **elevation of privilege** attack occurs when Bob's app reads Mallory's file and automatically tries to take some action based on the contents of that payload. Consider an app that reads the contents of a startup script from the machine-wide store and passes those contents to `Process.Start`. If Mallory can place a malicious script inside the machine-wide store, when Bob launches his app:

- His app parses and launches Mallory's malicious script *under the context of Bob's user profile*.
- Mallory gains access to Bob's account on the local machine.

Denial of service

A **denial of service** attack occurs when Bob's app reads Mallory's file and crashes or otherwise stops functioning correctly. Consider again the app mentioned previously, which attempts to parse a startup script from the machine-wide store. If Mallory can place a file with malformed contents inside the machine-wide store, she might:

- Cause Bob's app to throw an exception early in the startup path.
- Prevent the app from launching successfully because of the exception.

She has then denied Bob the ability to launch the app under his own user account.

Information disclosure

An **information disclosure** attack occurs when Mallory can trick Bob into disclosing the contents of a file that Mallory does not normally have access to. Consider that Bob has a secret file `C:\Users\Bob\secret.txt` that Mallory wants to read. She knows the path to this file, but she cannot read it because Windows forbids her from gaining access to Bob's user profile directory.

Instead, Mallory places a hard link into the machine-wide store. This is a special kind of file that itself does not contain any contents, rather, it points to another file on disk. Attempting to read the hard link file will instead read the contents of the file targeted by the link. After creating the hard link, Mallory still cannot read the file contents because she does not have access to the target (`C:\Users\Bob\secret.txt`) of the link. However, Bob *does* have access to this file.

When Bob's app reads from the machine-wide store, it now inadvertently reads the contents of his `secret.txt` file, just as if the file itself had been present in the machine-wide store. When Bob's app exits, if it attempts to resave the file to the machine-wide store, it will end up placing an actual copy of the file in the `*C:\ProgramData\IsolatedStorage*` directory. Since this directory is readable by any user on the machine, Mallory can now read the contents of the file.

Best practices to defend against these attacks

Important: If your environment has multiple mutually untrusted users, **do not** call the API

`IsolatedStorageFile.GetEnumerator(IsolatedStorageScope.Machine)` or invoke the tool

`storeadm.exe /machine /list`. Both of these assume that they're operating on trusted data. If an attacker can seed a malicious payload in the machine-wide store, that payload can lead to an elevation of privilege attack under the context of the user who runs these commands.

If operating in a multi-user environment, reconsider use of isolated storage features that target the *Machine* scope. If an app must read data from a machine-wide location, prefer to read the data from a location that's writable only by admin accounts. The `%PROGRAMFILES%` directory and the `HKLM` registry hive are examples of locations that are writable by only administrators and readable by everyone. Data read from those locations is therefore considered trustworthy.

If an app must use the *Machine* scope in a multi-user environment, validate the contents of any file that you read from the machine-wide store. If the app deserializing object graphs from these files, consider using safer serializers like `XmlSerializer` instead of dangerous serializers like `BinaryFormatter` or `NetDataContractSerializer`. Use caution with deeply nested object graphs or object graphs that perform resource allocation based on the file contents.

Isolated Storage Locations

Sometimes it is helpful to verify a change to isolated storage by using the file system of the operating system. You might also want to know the location of isolated storage files. This location is different depending on the operating system. The following table shows the root locations where isolated storage is created on a few common operating systems. Look for Microsoft\IsolatedStorage directories under this root location. You must change folder settings to show hidden files and folders in order to see isolated storage in the file system.

OPERATING SYSTEM	LOCATION IN FILE SYSTEM
Windows 2000, Windows XP, Windows Server 2003 (upgrade from Windows NT 4.0)	<p>Roaming-enabled stores = <code><SYSTEMROOT>\Profiles\<user>\Application Data</code></p> <p>Nonroaming stores = <code><SYSTEMROOT>\Profiles\<user>\Local Settings\Application Data</code></p>
Windows 2000 - clean installation (and upgrades from Windows 98 and Windows NT 3.51)	<p>Roaming-enabled stores = <code><SYSTEMDRIVE>\Documents and Settings\<user>\Application Data</code></p> <p>Nonroaming stores = <code><SYSTEMDRIVE>\Documents and Settings\<user>\Local Settings\Application Data</code></p>
Windows XP, Windows Server 2003 - clean installation (and upgrades from Windows 2000 and Windows 98)	<p>Roaming-enabled stores = <code><SYSTEMDRIVE>\Documents and Settings\<user>\Application Data</code></p> <p>Nonroaming stores = <code><SYSTEMDRIVE>\Documents and Settings\<user>\Local Settings\Application Data</code></p>

OPERATING SYSTEM	LOCATION IN FILE SYSTEM
Windows 8, Windows 7, Windows Server 2008, Windows Vista	<p>Roaming-enabled stores = <SYSTEMDRIVE>\Users\<user>\AppData\Roaming</p> <p>Nonroaming stores = <SYSTEMDRIVE>\Users\<user>\AppData\Local</p>

Creating, Enumerating, and Deleting Isolated Storage

.NET provides three classes in the [System.IO.IsolatedStorage](#) namespace to help you perform tasks that involve isolated storage:

- [IsolatedStorageFile](#), derives from [System.IO.IsolatedStorage.IsolatedStorage](#) and provides basic management of stored assembly and application files. An instance of the [IsolatedStorageFile](#) class represents a single store located in the file system.
- [IsolatedStorageFileStream](#) derives from [System.IO.FileStream](#) and provides access to the files in a store.
- [IsolatedStorageScope](#) is an enumeration that enables you to create and select a store with the appropriate isolation type.

The isolated storage classes enable you to create, enumerate, and delete isolated storage. The methods for performing these tasks are available through the [IsolatedStorageFile](#) object. Some operations require you to have the [IsolatedStorageFilePermission](#) permission that represents the right to administer isolated storage; you might also need to have operating system rights to access the file or directory.

For a series of examples that demonstrate common isolated storage tasks, see the how-to topics listed in [Related Topics](#).

Scenarios for Isolated Storage

Isolated storage is useful in many situations, including these four scenarios:

- Downloaded controls. Managed code controls downloaded from the Internet are not allowed to write to the hard drive through normal I/O classes, but they can use isolated storage to persist users' settings and application states.
- Shared component storage. Components that are shared between applications can use isolated storage to provide controlled access to data stores.
- Server storage. Server applications can use isolated storage to provide individual stores for a large number of users making requests to the application. Because isolated storage is always segregated by user, the server must impersonate the user making the request. In this case, data is isolated based on the identity of the principal, which is the same identity the application uses to distinguish between its users.
- Roaming. Applications can also use isolated storage with roaming user profiles. This allows a user's isolated stores to roam with the profile.

Do not use isolated storage in the following situations:

- To store high-value secrets, such as unencrypted keys or passwords, because isolated storage is not protected from highly trusted code, from unmanaged code, or from trusted users of the computer.
- To store code.

- To store configuration and deployment settings, which administrators control. (User preferences are not considered to be configuration settings because administrators do not control them.)

Many applications use a database to store and isolate data, in which case one or more rows in a database might represent storage for a specific user. You might choose to use isolated storage instead of a database when the number of users is small, when the overhead of using a database is significant, or when no database facility exists. Also, when the application requires storage that is more flexible and complex than what a row in a database provides, isolated storage can provide a viable alternative.

Related articles

TITLE	DESCRIPTION
Types of Isolation	Describes the different types of isolation.
How to: Obtain Stores for Isolated Storage	Provides an example of using the <code>IsolatedStorageFile</code> class to obtain a store isolated by user and assembly.
How to: Enumerate Stores for Isolated Storage	Shows how to use the <code>IsolatedStorageFile.GetEnumerator</code> method to calculate the size of all isolated storage for the user.
How to: Delete Stores in Isolated Storage	Shows how to use the <code>IsolatedStorageFile.Remove</code> method in two different ways to delete isolated stores.
How to: Anticipate Out-of-Space Conditions with Isolated Storage	Shows how to measure the remaining space in an isolated store.
How to: Create Files and Directories in Isolated Storage	Provides some examples of creating files and directories in an isolated store.
How to: Find Existing Files and Directories in Isolated Storage	Demonstrates how to read the directory structure and files in isolated storage.
How to: Read and Write to Files in Isolated Storage	Provides an example of writing a string to an isolated storage file and reading it back.
How to: Delete Files and Directories in Isolated Storage	Demonstrates how to delete isolated storage files and directories.
File and Stream I/O	Explains how you can perform synchronous and asynchronous file and data stream access.

Reference

- [System.IO.IsolatedStorage.IsolatedStorage](#)
- [System.IO.IsolatedStorage.IsolatedStorageFile](#)
- [System.IO.IsolatedStorage.IsolatedStorageFileStream](#)
- [System.IO.IsolatedStorage.IsolatedStorageScope](#)

Types of isolation

9/20/2022 • 5 minutes to read • [Edit Online](#)

Access to isolated storage is always restricted to the user who created it. To implement this type of isolation, the common language runtime uses the same notion of user identity that the operating system recognizes, which is the identity associated with the process in which the code is running when the store is opened. This identity is an authenticated user identity, but impersonation can cause the identity of the current user to change dynamically.

Access to isolated storage is also restricted according to the identity associated with the application's domain and assembly, or with the assembly alone. The runtime obtains these identities in the following ways:

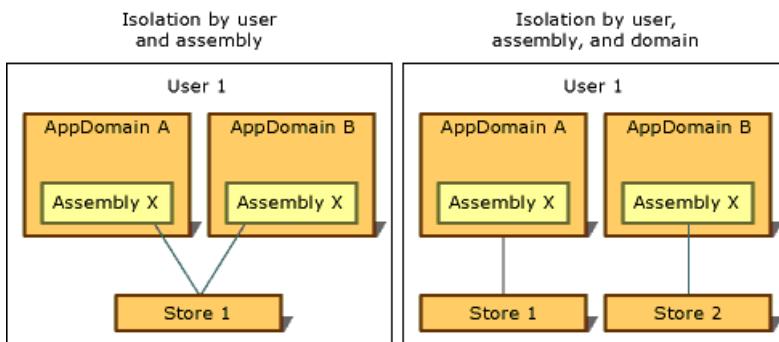
- Domain identity represents the evidence of the application, which in the case of a web application might be the full URL. For shell-hosted code, the domain identity might be based on the application directory path. For example, if the executable runs from the path C:\Office\MyApp.exe, the domain identity would be C:\Office\MyApp.exe.
- Assembly identity is the evidence of the assembly. This might come from a cryptographic digital signature, which can be the assembly's [strong name](#), the software publisher of the assembly, or its URL identity. If an assembly has both a strong name and a software publisher identity, then the software publisher identity is used. If the assembly comes from the Internet and is unsigned, the URL identity is used. For more information about assemblies and strong names, see [Programming with Assemblies](#).
- Roaming stores move with a user that has a roaming user profile. Files are written to a network directory and are downloaded to any computer the user logs into. For more information about roaming user profiles, see [IsolatedStorageScope.Roaming](#).

By combining the concepts of user, domain, and assembly identity, isolated storage can isolate data in the following ways, each of which has its own usage scenarios:

- [Isolation by user and assembly](#)
- [Isolation by user, domain, and assembly](#)

Either of these isolations can be combined with a roaming user profile. For more information, see the section [Isolated Storage and Roaming](#).

The following illustration demonstrates how stores are isolated in different scopes:



Except for roaming stores, isolated storage is always implicitly isolated by computer because it uses the storage facilities that are local to a given computer.

IMPORTANT

Isolated storage is not available for Windows 8.x Store apps. Instead, use the application data classes in the `Windows.Storage` namespaces included in the Windows Runtime API to store local data and files. For more information, see [Application data](#) in the Windows Dev Center.

Isolation by User and Assembly

When the assembly that uses the data store needs to be accessible from any application's domain, isolation by user and assembly is appropriate. Typically, in this situation, isolated storage is used to store data that applies across multiple applications and is not tied to any particular application, such as the user's name or license information. To access storage isolated by user and assembly, code must be trusted to transfer information between applications. Typically, isolation by user and assembly is allowed on intranets but not on the Internet. Calling the static `IsolatedStorageFile.GetStore` method and passing in a user and an assembly `IsolatedStorageScope` returns storage with this kind of isolation.

The following code example retrieves a store that is isolated by user and assembly. The store can be accessed through the `isoFile` object.

```
IsolatedStorageFile^ isoFile =
    IsolatedStorageFile::GetStore(IsolatedStorageScope::User |
        IsolatedStorageScope::Assembly, (Type^)nullptr, (Type^)nullptr);
```

```
IsolatedStorageFile isoFile =
    IsolatedStorageFile::GetStore(IsolatedStorageScope::User |
        IsolatedStorageScope::Assembly, null, null);
```

```
Dim isoFile As IsolatedStorageFile = _
    IsolatedStorageFile::GetStore(IsolatedStorageScope::User Or _
        IsolatedStorageScope::Assembly, Nothing, Nothing)
```

For an example that uses the evidence parameters, see [GetStore\(IsolatedStorageScope, Evidence, Type, Evidence, Type\)](#).

The `GetUserStoreForAssembly` method is available as a shortcut, as shown in the following code example. This shortcut cannot be used to open stores that are capable of roaming; use `GetStore` in such cases.

```
IsolatedStorageFile^ isoFile = IsolatedStorageFile:: GetUserStoreForAssembly();
```

```
IsolatedStorageFile isoFile = IsolatedStorageFile:: GetUserStoreForAssembly();
```

```
Dim isoFile As IsolatedStorageFile = _
    IsolatedStorageFile:: GetUserStoreForAssembly()
```

Isolation by User, Domain, and Assembly

If your application uses a third-party assembly that requires a private data store, you can use isolated storage to store the private data. Isolation by user, domain, and assembly ensures that only code in a given assembly can

access the data, and only when the assembly is used by the application that was running when the assembly created the store, and only when the user for whom the store was created runs the application. Isolation by user, domain, and assembly keeps the third-party assembly from leaking data to other applications. This isolation type should be your default choice if you know that you want to use isolated storage but are not sure which type of isolation to use. Calling the static [GetStore](#) method of [IsolatedStorageFile](#) and passing in a user, domain, and assembly [IsolatedStorageScope](#) returns storage with this kind of isolation.

The following code example retrieves a store isolated by user, domain, and assembly. The store can be accessed through the `isoFile` object.

```
IsolatedStorageFile^ isoFile =
    IsolatedStorageFile::GetStore(IsolatedStorageScope::User |
        IsolatedStorageScope::Domain |
        IsolatedStorageScope::Assembly, (Type^)nullptr, (Type^)nullptr);
```

```
IsolatedStorageFile isoFile =
    IsolatedStorageFile::GetStore(IsolatedStorageScope::User |
        IsolatedStorageScope::Domain |
        IsolatedStorageScope::Assembly, null, null);
```

```
Dim isoFile As IsolatedStorageFile = _
    IsolatedStorageFile::GetStore(IsolatedStorageScope::User Or _
        IsolatedStorageScope::Domain Or _
        IsolatedStorageScope::Assembly, Nothing, Nothing)
```

Another method is available as a shortcut, as shown in the following code example. This shortcut cannot be used to open stores that are capable of roaming; use [GetStore](#) in such cases.

```
IsolatedStorageFile^ isoFile = IsolatedStorageFile:: GetUserStoreForDomain();
```

```
IsolatedStorageFile isoFile = IsolatedStorageFile:: GetUserStoreForDomain();
```

```
Dim isoFile As IsolatedStorageFile = _
    IsolatedStorageFile:: GetUserStoreForDomain()
```

Isolated Storage and Roaming

Roaming user profiles are a Windows feature that enables a user to set up an identity on a network and use that identity to log into any network computer, carrying over all personalized settings. An assembly that uses isolated storage can specify that the user's isolated storage should move with the roaming user profile. Roaming can be used in conjunction with isolation by user and assembly or with isolation by user, domain, and assembly. If a roaming scope is not used, stores will not roam even if a roaming user profile is used.

The following code example retrieves a roaming store isolated by user and assembly. The store can be accessed through the `isoFile` object.

```
IsolatedStorageFile^ isoFile =
    IsolatedStorageFile::GetStore(IsolatedStorageScope::User |
        IsolatedStorageScope::Assembly |
        IsolatedStorageScope::Roaming, (Type^)nullptr, (Type^)nullptr);
```

```
IsolatedStorageFile isoFile =
    IsolatedStorageFile.GetStore(IsolatedStorageScope.User |
        IsolatedStorageScope.Assembly |
        IsolatedStorageScope.Roaming, null, null);
```

```
Dim isoFile As IsolatedStorageFile = _
    IsolatedStorageFile.GetStore(IsolatedStorageScope.User Or _
        IsolatedStorageScope.Assembly Or _
        IsolatedStorageScope.Roaming, Nothing, Nothing)
```

A domain scope can be added to create a roaming store isolated by user, domain, and application. The following code example demonstrates this.

```
IsolatedStorageFile^ isoFile =
    IsolatedStorageFile::GetStore(IsolatedStorageScope::User |
        IsolatedStorageScope::Assembly | IsolatedStorageScope::Domain |
        IsolatedStorageScope::Roaming, (Type^)nullptr, (Type^)nullptr);
```

```
IsolatedStorageFile isoFile =
    IsolatedStorageFile.GetStore(IsolatedStorageScope.User |
        IsolatedStorageScope.Assembly | IsolatedStorageScope.Domain |
        IsolatedStorageScope.Roaming, null, null);
```

```
Dim isoFile As IsolatedStorageFile = _
    IsolatedStorageFile.GetStore(IsolatedStorageScope.User Or _
        IsolatedStorageScope.Assembly Or IsolatedStorageScope.Domain Or _
        IsolatedStorageScope.Roaming, Nothing, Nothing)
```

See also

- [IsolatedStorageScope](#)
- [Isolated Storage](#)

How to: Obtain Stores for Isolated Storage

9/20/2022 • 2 minutes to read • [Edit Online](#)

An isolated store exposes a virtual file system within a data compartment. The [IsolatedStorageFile](#) class supplies a number of methods for interacting with an isolated store. To create and retrieve stores, [IsolatedStorageFile](#) provides three static methods:

- [GetUserStoreForAssembly](#) returns storage that is isolated by user and assembly.
- [GetUserStoreForDomain](#) returns storage that is isolated by domain and assembly.

Both methods retrieve a store that belongs to the code from which they are called.

- The static method [GetStore](#) returns an isolated store that is specified by passing in a combination of scope parameters.

The following code returns a store that is isolated by user, assembly, and domain.

```
IsolatedStorageFile^ isoStore = IsolatedStorageFile::GetStore(IsolatedStorageScope::User |  
    IsolatedStorageScope::Assembly | IsolatedStorageScope::Domain, (Type ^)nullptr, (Type ^)nullptr);
```

```
IsolatedStorageFile isoStore = IsolatedStorageFile.GetStore(IsolatedStorageScope.User |  
    IsolatedStorageScope.Assembly | IsolatedStorageScope.Domain, null, null);
```

```
Dim isoStore As IsolatedStorageFile = IsolatedStorageFile.GetStore(IsolatedStorageScope.User Or  
    IsolatedStorageScope.Assembly Or IsolatedStorageScope.Domain, Nothing, Nothing)
```

You can use the [GetStore](#) method to specify that a store should roam with a roaming user profile. For details on how to set this up, see [Types of Isolation](#).

Isolated stores obtained from within different assemblies are, by default, different stores. You can access the store of a different assembly or domain by passing in the assembly or domain evidence in the parameters of the [GetStore](#) method. This requires permission to access isolated storage by application domain identity. For more information, see the [GetStore](#) method overloads.

The [GetUserStoreForAssembly](#), [GetUserStoreForDomain](#), and [GetStore](#) methods return an [IsolatedStorageFile](#) object. To help you decide which isolation type is most appropriate for your situation, see [Types of Isolation](#). When you have an isolated storage file object, you can use the isolated storage methods to read, write, create, and delete files and directories.

There is no mechanism that prevents code from passing an [IsolatedStorageFile](#) object to code that does not have sufficient access to get the store itself. Domain and assembly identities and isolated storage permissions are checked only when a reference to an [IsolatedStorage](#) object is obtained, typically in the [GetUserStoreForAssembly](#), [GetUserStoreForDomain](#), or [GetStore](#) method. Protecting references to [IsolatedStorageFile](#) objects is, therefore, the responsibility of the code that uses these references.

Example

The following code provides a simple example of a class obtaining a store that is isolated by user and assembly. The code can be changed to retrieve a store that is isolated by user, domain, and assembly by adding [IsolatedStorageScope.Domain](#) to the arguments that the [GetStore](#) method passes.

After you run the code, you can confirm that a store was created by typing `StoreAdm /LIST` at the command line. This runs the [Isolated Storage tool \(Storeadm.exe\)](#) and lists all the current isolated stores for the user.

```
using namespace System;
using namespace System::IO::IsolatedStorage;

public ref class ObtainingAStore
{
public:
    static void Main()
    {
        // Get a new isolated store for this assembly and put it into an
        // isolated store object.

        IsolatedStorageFile^ isoStore = IsolatedStorageFile::GetStore(IsolatedStorageScope::User |
            IsolatedStorageScope::Assembly, (Type ^)nullptr, (Type ^)nullptr);
    }
};
```

```
using System;
using System.IO.IsolatedStorage;

public class ObtainingAStore
{
    public static void Main()
    {
        // Get a new isolated store for this assembly and put it into an
        // isolated store object.

        IsolatedStorageFile isoStore = IsolatedStorageFile.GetStore(IsolatedStorageScope.User |
            IsolatedStorageScope.Assembly, null, null);
    }
}
```

```
Imports System.IO.IsolatedStorage

Public Class ObtainingAStore
    Public Shared Sub Main()
        ' Get a new isolated store for this assembly and put it into an
        ' isolated store object.

        Dim isoStore As IsolatedStorageFile = IsolatedStorageFile.GetStore(IsolatedStorageScope.User Or
            IsolatedStorageScope.Assembly, Nothing, Nothing)
    End Sub
End Class
```

See also

- [IsolatedStorageFile](#)
- [IsolatedStorageScope](#)
- [Isolated Storage](#)
- [Types of Isolation](#)
- [Assemblies in .NET](#)

How to: Enumerate Stores for Isolated Storage

9/20/2022 • 2 minutes to read • [Edit Online](#)

You can enumerate all isolated stores for the current user by using the `IsolatedStorageFile.GetEnumerator` static method. This method takes an `IsolatedStorageScope` value and returns an `IsolatedStorageFile` enumerator. To enumerate stores, you must have the `IsolatedStorageFilePermission` permission that specifies the `AdministerIsolatedStorageByUser` value. If you call the `GetEnumerator` method with the `User` value, it returns an array of `IsolatedStorageFile` objects that are defined for the current user.

Example

The following code example obtains a store that is isolated by user and assembly, creates a few files, and retrieves those files by using the `GetEnumerator` method.

```
using System;
using System.IO;
using System.IO.IsolatedStorage;
using System.Collections;

public class EnumeratingStores
{
    public static void Main()
    {
        using (IsolatedStorageFile isoStore = IsolatedStorageFile.GetStore(IsolatedStorageScope.User |
IsolatedStorageScope.Assembly, null, null))
        {
            isoStore.CreateFile("TestFileA.Txt");
            isoStore.CreateFile("TestFileB.Txt");
            isoStore.CreateFile("TestFileC.Txt");
            isoStore.CreateFile("TestFileD.Txt");
        }

        Ienumerator allFiles = IsolatedStorageFile.GetEnumerator(IsolatedStorageScope.User);
        long totalsize = 0;

        while (allFiles.MoveNext())
        {
            IsolatedStorageFile storeFile = (IsolatedStorageFile)allFiles.Current;
            totalsize += (long)storeFile.UsedSize;
        }

        Console.WriteLine("The total size = " + totalsize);
    }
}
```

```
Imports System.IO
Imports System.IO.IsolatedStorage

Module Module1
    Sub Main()
        Using isoStore As IsolatedStorageFile = IsolatedStorageFile.GetStore(IsolatedStorageScope.User Or
IsolatedStorageScope.Assembly, Nothing, Nothing)
            isoStore.CreateFile("TestFileA.Txt")
            isoStore.CreateFile("TestFileB.Txt")
            isoStore.CreateFile("TestFileC.Txt")
            isoStore.CreateFile("TestFileD.Txt")
        End Using

        Dim allFiles As IEnumerable = IsolatedStorageFile.GetEnumerator(IsolatedStorageScope.User)
        Dim totalsize As Long = 0

        While (allFiles.MoveNext())
            Dim storeFile As IsolatedStorageFile = CType(allFiles.Current, IsolatedStorageFile)
            totalsize += CType(storeFile.UsedSize, Long)
        End While

        Console.WriteLine("The total size = " + totalsize.ToString())
    End Sub
End Module
```

See also

- [IsolatedStorageFile](#)
- [Isolated Storage](#)

How to: Delete Stores in Isolated Storage

9/20/2022 • 3 minutes to read • [Edit Online](#)

The [IsolatedStorageFile](#) class supplies two methods for deleting isolated storage files:

- The instance method [Remove\(\)](#) does not take any arguments and deletes the store that calls it. No permissions are required for this operation. Any code that can access the store can delete any or all the data inside it.
- The static method [Remove\(IsolatedStorageScope\)](#) takes the [User](#) enumeration value, and deletes all the stores for the user who is running the code. This operation requires [IsolatedStorageFilePermission](#) permission for the [AdministerIsolatedStorageByUser](#) value.

Example

The following code example demonstrates the use of the static and instance [Remove](#) methods. The class obtains two stores; one is isolated for user and assembly and the other is isolated for user, domain, and assembly. The user, domain, and assembly store is then deleted by calling the [Remove\(\)](#) method of the isolated storage file `isoStore1`. Then, all remaining stores for the user are deleted by calling the static method [Remove\(IsolatedStorageScope\)](#).

```
using namespace System;
using namespace System::IO::IsolatedStorage;

public ref class DeletingStores
{
public:
    static void Main()
    {
        // Get a new isolated store for this user, domain, and assembly.
        // Put the store into an IsolatedStorageFile object.

        IsolatedStorageFile^ isoStore1 = IsolatedStorageFile::GetStore(IsolatedStorageScope::User |
            IsolatedStorageScope::Domain | IsolatedStorageScope::Assembly, (Type ^)nullptr, (Type
            ^)nullptr);
        Console::WriteLine("A store isolated by user, assembly, and domain has been obtained.");

        // Get a new isolated store for user and assembly.
        // Put that store into a different IsolatedStorageFile object.

        IsolatedStorageFile^ isoStore2 = IsolatedStorageFile::GetStore(IsolatedStorageScope::User |
            IsolatedStorageScope::Assembly, (Type ^)nullptr, (Type ^)nullptr);
        Console::WriteLine("A store isolated by user and assembly has been obtained.");

        // The Remove method deletes a specific store, in this case the
        // isoStore1 file.

        isoStore1->Remove();
        Console::WriteLine("The user, domain, and assembly isolated store has been deleted.");

        // This static method deletes all the isolated stores for this user.

        IsolatedStorageFile::Remove(IsolatedStorageScope::User);
        Console::WriteLine("All isolated stores for this user have been deleted.");
    } // End of Main.
};

int main()
{
    DeletingStores::Main();
}
```

```

using System;
using System.IO.IsolatedStorage;

public class DeletingStores
{
    public static void Main()
    {
        // Get a new isolated store for this user, domain, and assembly.
        // Put the store into an IsolatedStorageFile object.

        IsolatedStorageFile isoStore1 = IsolatedStorageFile.GetStore(IsolatedStorageScope.User |
            IsolatedStorageScope.Domain | IsolatedStorageScope.Assembly, null, null);
        Console.WriteLine("A store isolated by user, assembly, and domain has been obtained.");

        // Get a new isolated store for user and assembly.
        // Put that store into a different IsolatedStorageFile object.

        IsolatedStorageFile isoStore2 = IsolatedStorageFile.GetStore(IsolatedStorageScope.User |
            IsolatedStorageScope.Assembly, null, null);
        Console.WriteLine("A store isolated by user and assembly has been obtained.");

        // The Remove method deletes a specific store, in this case the
        // isoStore1 file.

        isoStore1.Remove();
        Console.WriteLine("The user, domain, and assembly isolated store has been deleted.");

        // This static method deletes all the isolated stores for this user.

        IsolatedStorageFile.Remove(IsolatedStorageScope.User);
        Console.WriteLine("All isolated stores for this user have been deleted.");
    } // End of Main.
}

```

```

Imports System.IO.IsolatedStorage

Public Class DeletingStores
    Public Shared Sub Main()
        ' Get a new isolated store for this user, domain, and assembly.
        ' Put the store into an IsolatedStorageFile object.

        Dim isoStore1 As IsolatedStorageFile = IsolatedStorageFile.GetStore(IsolatedStorageScope.User Or
            IsolatedStorageScope.Domain Or IsolatedStorageScope.Assembly, Nothing, Nothing)
        Console.WriteLine("A store isolated by user, assembly, and domain has been obtained.")

        ' Get a new isolated store for user and assembly.
        ' Put that store into a different IsolatedStorageFile object.

        Dim isoStore2 As IsolatedStorageFile = IsolatedStorageFile.GetStore(IsolatedStorageScope.User Or
            IsolatedStorageScope.Assembly, Nothing, Nothing)
        Console.WriteLine("A store isolated by user and assembly has been obtained.")

        ' The Remove method deletes a specific store, in this case the
        ' isoStore1 file.

        isoStore1.Remove()
        Console.WriteLine("The user, domain, and assembly isolated store has been deleted.")

        ' This static method deletes all the isolated stores for this user.

        IsolatedStorageFile.Remove(IsolatedStorageScope.User)
        Console.WriteLine("All isolated stores for this user have been deleted.")

    End Sub
End Class

```

See also

- [IsolatedStorageFile](#)
- [Isolated Storage](#)

How to: Anticipate Out-of-Space Conditions with Isolated Storage

9/20/2022 • 2 minutes to read • [Edit Online](#)

Code that uses isolated storage is constrained by a [quota](#) that specifies the maximum size for the data compartment in which isolated storage files and directories exist. The quota is defined by security policy and is configurable by administrators. If the maximum allowed size is exceeded when you try to write data, an [IsolatedStorageException](#) exception is thrown and the operation fails. This helps prevent malicious denial-of-service attacks that could cause the application to refuse requests because data storage is filled.

To help you determine whether a given write attempt is likely to fail for this reason, the [IsolatedStorage](#) class provides three read-only properties: [AvailableFreeSpace](#), [UsedSize](#), and [Quota](#). You can use these properties to determine whether writing to the store will cause the maximum allowed size of the store to be exceeded. Keep in mind that isolated storage can be accessed concurrently; therefore, when you compute the amount of remaining storage, the storage space could be consumed by the time you try to write to the store. However, you can use the maximum size of the store to help determine whether the upper limit on available storage is about to be reached.

The [Quota](#) property depends on evidence from the assembly to work properly. For this reason, you should retrieve this property only on [IsolatedStorageFile](#) objects that were created by using the [GetUserStoreForAssembly](#), [GetUserStoreForDomain](#), or [GetStore](#) method. [IsolatedStorageFile](#) objects that were created in any other way (for example, objects that were returned from the [GetEnumerator](#) method) will not return an accurate maximum size.

Example

The following code example obtains an isolated store, creates a few files, and retrieves the [AvailableFreeSpace](#) property. The remaining space is reported in bytes.

```

using namespace System;
using namespace System::IO;
using namespace System::IO::IsolatedStorage;

public ref class CheckingSpace
{
public:
    static void Main()
    {
        // Get an isolated store for this assembly and put it into an
        // IsolatedStoreFile object.
        IsolatedStorageFile^ isoStore =  IsolatedStorageFile::GetStore(IsolatedStorageScope::User |
            IsolatedStorageScope::Assembly, (Type ^)nullptr, (Type ^)nullptr);

        // Create a few placeholder files in the isolated store.
        gcnew IsolatedStorageFileStream("InTheRoot.txt", FileMode::Create, isoStore);
        gcnew IsolatedStorageFileStream("Another.txt", FileMode::Create, isoStore);
        gcnew IsolatedStorageFileStream("AThird.txt", FileMode::Create, isoStore);
        gcnew IsolatedStorageFileStream("AFourth.txt", FileMode::Create, isoStore);
        gcnew IsolatedStorageFileStream("AFifth.txt", FileMode::Create, isoStore);

        Console::WriteLine(isoStore->AvailableFreeSpace + " bytes of space remain in this isolated store.");
    } // End of Main.
};

int main()
{
    CheckingSpace::Main();
}

```

```

using System;
using System.IO;
using System.IO.IsolatedStorage;

public class CheckingSpace
{
    public static void Main()
    {
        // Get an isolated store for this assembly and put it into an
        // IsolatedStoreFile object.
        IsolatedStorageFile isoStore =  IsolatedStorageFile.GetStore(IsolatedStorageScope.User |
            IsolatedStorageScope.Assembly, null, null);

        // Create a few placeholder files in the isolated store.
        new IsolatedStorageFileStream("InTheRoot.txt", FileMode.Create, isoStore);
        new IsolatedStorageFileStream("Another.txt", FileMode.Create, isoStore);
        new IsolatedStorageFileStream("AThird.txt", FileMode.Create, isoStore);
        new IsolatedStorageFileStream("AFourth.txt", FileMode.Create, isoStore);
        new IsolatedStorageFileStream("AFifth.txt", FileMode.Create, isoStore);

        Console.WriteLine(isoStore.AvailableFreeSpace + " bytes of space remain in this isolated store.");
    } // End of Main.
}

```

```
Imports System.IO
Imports System.IO.IsolatedStorage

Public Class CheckingSpace
    Public Shared Sub Main()
        ' Get an isolated store for this assembly and put it into an
        ' IsolatedStorageFile object.
        Dim isoStore As IsolatedStorageFile = _
            IsolatedStorageFile.GetStore(IsolatedStorageScope.User Or _
            IsolatedStorageScope.Assembly, Nothing, Nothing)

        ' Create a few placeholder files in the isolated store.
        Dim aStream As New IsolatedStorageFileStream("InTheRoot.txt", FileMode.Create, isoStore)
        Dim bStream As New IsolatedStorageFileStream("Another.txt", FileMode.Create, isoStore)
        Dim cStream As New IsolatedStorageFileStream("AThird.txt", FileMode.Create, isoStore)
        Dim dStream As New IsolatedStorageFileStream("AFourth.txt", FileMode.Create, isoStore)
        Dim eStream As New IsolatedStorageFileStream("AFifth.txt", FileMode.Create, isoStore)

        Console.WriteLine(isoStore.AvailableFreeSpace + " bytes of space remain in this isolated store.")
    End Sub
End Class
```

See also

- [IsolatedStorageFile](#)
- [Isolated Storage](#)
- [How to: Obtain Stores for Isolated Storage](#)

How to: Create Files and Directories in Isolated Storage

9/20/2022 • 2 minutes to read • [Edit Online](#)

After you've obtained an isolated store, you can create directories and files for storing data. Within a store, file and directory names are specified with respect to the root of the virtual file system.

To create a directory, use the [IsolatedStorageFile.CreateDirectory](#) instance method. If you specify a subdirectory of a directory that doesn't exist, both directories are created. If you specify a directory that already exists, the method returns without creating a directory, and no exception is thrown. However, if you specify a directory name that contains invalid characters, an [IsolatedStorageException](#) exception is thrown.

To create a file, use the [IsolatedStorageFile.CreateFile](#) method.

In the Windows operating system, isolated storage file and directory names are case-insensitive. That is, if you create a file named `ThisFile.txt`, and then create another file named `THISFILE.TXT`, only one file is created. The file name keeps its original casing for display purposes.

Isolated storage file creation will throw an [IsolatedStorageException](#) if the path contains a directory that does not exist.

Example

The following code example illustrates how to create files and directories in an isolated store.

```
using System;
using System.IO;
using System.IO.IsolatedStorage;

public class CreatingFilesDirectories
{
    public static void Main()
    {
        using (IsolatedStorageFile isoStore = IsolatedStorageFile.GetStore(IsolatedStorageScope.User |
IsolatedStorageScope.Domain | IsolatedStorageScope.Assembly, null, null))
        {
            isoStore.CreateDirectory("TopLevelDirectory");
            isoStore.CreateDirectory("TopLevelDirectory/SecondLevel");
            isoStore.CreateDirectory("AnotherTopLevelDirectory/InsideDirectory");
            Console.WriteLine("Created directories.");

            isoStore.CreateFile("InTheRoot.txt");
            Console.WriteLine("Created a new file in the root.");

            isoStore.CreateFile("AnotherTopLevelDirectory/InsideDirectory/HereIAm.txt");
            Console.WriteLine("Created a new file in the InsideDirectory.");
        }
    }
}
```

```
Imports System.IO
Imports System.IO.IsolatedStorage

Module Module1
    Sub Main()
        Using isoStore As IsolatedStorageFile = IsolatedStorageFile.GetStore(IsolatedStorageScope.User Or
IsolatedStorageScope.Assembly Or IsolatedStorageScope.Domain, Nothing, Nothing)

            isoStore.CreateDirectory("TopLevelDirectory")
            isoStore.CreateDirectory("TopLevelDirectory/SecondLevel")
            isoStore.CreateDirectory("AnotherTopLevelDirectory/InsideDirectory")
            Console.WriteLine("Created directories.")

            isoStore.CreateFile("InTheRoot.txt")
            Console.WriteLine("Created a new file in the root.")

            isoStore.CreateFile("AnotherTopLevelDirectory/InsideDirectory/HereIAm.txt")
            Console.WriteLine("Created a new file in the InsideDirectory.")
        End Using
    End Sub
End Module
```

See also

- [IsolatedStorageFile](#)
- [IsolatedStorageFileStream](#)
- [Isolated Storage](#)

How to: Find Existing Files and Directories in Isolated Storage

9/20/2022 • 6 minutes to read • [Edit Online](#)

To search for a directory in isolated storage, use the [IsolatedStorageFile.GetDirectoryNames](#) method. This method takes a string that represents a search pattern. You can use both single-character (?) and multi-character (*) wildcard characters in the search pattern, but the wildcard characters must appear in the final portion of the name. For example, `directory1/*ect*` is a valid search string, but `*ect*/directory2` is not.

To search for a file, use the [IsolatedStorageFile.GetFileNames](#) method. The restriction for wildcard characters in search strings that applies to [GetDirectoryNames](#) also applies to [GetFileNames](#).

Neither of these methods is recursive; the [IsolatedStorageFile](#) class does not supply any methods for listing all directories or files in your store. However, recursive methods are shown in the following code example.

Example

The following code example illustrates how to create files and directories in an isolated store. First, a store that is isolated for user, domain, and assembly is retrieved and placed in the `isoStore` variable. The [CreateDirectory](#) method is used to set up a few different directories, and the [IsolatedStorageFileStream\(String, FileMode, IsolatedStorageFile\)](#) constructor creates some files in these directories. The code then loops through the results of the [GetAllDirectories](#) method. This method uses [GetDirectoryNames](#) to find all the directory names in the current directory. These names are stored in an array, and then [GetAllDirectories](#) calls itself, passing in each directory it has found. As a result, all the directory names are returned in an array. Next, the code calls the [GetAllFiles](#) method. This method calls [GetAllDirectories](#) to find out the names of all the directories, and then it checks each directory for files by using the [GetFileNames](#) method. The result is returned in an array for display.

```
using namespace System;
using namespace System::IO;
using namespace System::IO::IsolatedStorage;
using namespace System::Collections;
using namespace System::Collections::Generic;

public class FindingExistingFilesAndDirectories
{
public:
    // Retrieves an array of all directories in the store, and
    // displays the results.
    static void Main()
    {
        // This part of the code sets up a few directories and files in the
        // store.
        IsolatedStorageFile^ isoStore = IsolatedStorageFile::GetStore(IsolatedStorageScope::User |
            IsolatedStorageScope::Assembly, (Type ^)nullptr, (Type ^)nullptr);
        isoStore->CreateDirectory("TopLevelDirectory");
        isoStore->CreateDirectory("TopLevelDirectory/SecondLevel");
        isoStore->CreateDirectory("AnotherTopLevelDirectory/InsideDirectory");
        gcnew IsolatedStorageFileStream("InTheRoot.txt", FileMode::Create, isoStore);
        gcnew IsolatedStorageFileStream("AnotherTopLevelDirectory/InsideDirectory/HereIAm.txt",
            FileMode::Create, isoStore);
        // End of setup.

        Console::WriteLine('\r');
        Console::WriteLine("Here is a list of all directories in this isolated store:");
    }
}
```

```

        for each (String^ directory in GetAllDirectories("*, isoStore))
    {
        Console::WriteLine(directory);
    }
    Console::WriteLine('\r');

    // Retrieve all the files in the directory by calling the GetFiles
    // method.

    Console::WriteLine("Here is a list of all the files in this isolated store:");
    for each (String^ file in GetAllFiles("*, isoStore))
    {
        Console::WriteLine(file);
    }

} // End of Main.

// Method to retrieve all directories, recursively, within a store.
static List<String^>^ GetAllDirectories(String^ pattern, IsolatedStorageFile^ storeFile)
{
    // Get the root of the search string.
    String^ root = Path::GetDirectoryName(pattern);

    if (root != "")
    {
        root += "/";
    }

    // Retrieve directories.
    array<String^>^ directories = storeFile->GetDirectoryNames(pattern);

    List<String^>^ directoryList = gcnew List<String^>(directories);

    // Retrieve subdirectories of matches.
    for (int i = 0, max = directories->Length; i < max; i++)
    {
        String^ directory = directoryList[i] + "/";
        List<String^>^ more = GetAllDirectories (root + directory + "*", storeFile);

        // For each subdirectory found, add in the base path.
        for (int j = 0; j < more->Count; j++)
        {
            more[j] = directory + more[j];
        }

        // Insert the subdirectories into the list and
        // update the counter and upper bound.
        directoryList->InsertRange(i + 1, more);
        i += more->Count;
        max += more->Count;
    }

    return directoryList;
}

static List<String^>^ GetAllFiles(String^ pattern, IsolatedStorageFile^ storeFile)
{
    // Get the root and file portions of the search string.
    String^ fileString = Path::GetFileName(pattern);
    array<String^>^ files = storeFile->GetFileNames(pattern);

    List<String^>^ fileList = gcnew List<String^>(files);

    // Loop through the subdirectories, collect matches,
    // and make separators consistent.
    for each (String^ directory in GetAllDirectories( "*", storeFile))
    {
        for each (String^ file in storeFile->GetFileNames(directory + "/" + fileString))

```

```

        {
            fileList->Add((directory + "/" + file));
        }
    }

    return fileList;
} // End of GetFiles.
};

int main()
{
    FindingExistingFilesAndDirectories::Main();
}

```

```

using System;
using System.IO;
using System.IO.IsolatedStorage;
using System.Collections;
using System.Collections.Generic;

public class FindingExistingFilesAndDirectories
{
    // Retrieves an array of all directories in the store, and
    // displays the results.
    public static void Main()
    {
        // This part of the code sets up a few directories and files in the
        // store.
        IsolatedStorageFile isoStore = IsolatedStorageFile.GetStore(IsolatedStorageScope.User |
            IsolatedStorageScope.Assembly, null, null);
        isoStore.CreateDirectory("TopLevelDirectory");
        isoStore.CreateDirectory("TopLevelDirectory/SecondLevel");
        isoStore.CreateDirectory("AnotherTopLevelDirectory/InsideDirectory");
        isoStore.CreateFile("InTheRoot.txt");
        isoStore.CreateFile("AnotherTopLevelDirectory/InsideDirectory/HereIAm.txt");
        // End of setup.

        Console.WriteLine('\r');
        Console.WriteLine("Here is a list of all directories in this isolated store:");

        foreach (string directory in GetAllDirectories("*", isoStore))
        {
            Console.WriteLine(directory);
        }
        Console.WriteLine('\r');

        // Retrieve all the files in the directory by calling the GetFiles
        // method.

        Console.WriteLine("Here is a list of all the files in this isolated store:");
        foreach (string file in GetAllFiles("*", isoStore)){
            Console.WriteLine(file);
        }
    } // End of Main.

    // Method to retrieve all directories, recursively, within a store.
    public static List<String> GetAllDirectories(string pattern, IsolatedStorageFile storeFile)
    {
        // Get the root of the search string.
        string root = Path.GetDirectoryName(pattern);

        if (root != "")
        {
            root += "/";
        }

        // Retrieve directories.

```

```
List<String> directoryList = new List<String>(storeFile.GetDirectoryNames(pattern));

// Retrieve subdirectories of matches.
for (int i = 0, max = directoryList.Count; i < max; i++)
{
    string directory = directoryList[i] + "/";
    List<String> more = GetAllDirectories(root + directory + "*", storeFile);

    // For each subdirectory found, add in the base path.
    for (int j = 0; j < more.Count; j++)
    {
        more[j] = directory + more[j];
    }

    // Insert the subdirectories into the list and
    // update the counter and upper bound.
    directoryList.InsertRange(i + 1, more);
    i += more.Count;
    max += more.Count;
}

return directoryList;
}

public static List<String> GetAllFiles(string pattern, IsolatedStorageFile storeFile)
{
    // Get the root and file portions of the search string.
    string fileString = Path.GetFileName(pattern);

    List<String> fileList = new List<String>(storeFile.GetFileNames(pattern));

    // Loop through the subdirectories, collect matches,
    // and make separators consistent.
    foreach (string directory in GetAllDirectories("*", storeFile))
    {
        foreach (string file in storeFile.GetFileNames(directory + "/" + fileString))
        {
            fileList.Add((directory + "/" + file));
        }
    }

    return fileList;
} // End of GetFiles.
}
```

```

Imports System.IO
Imports System.IO.IsolatedStorage
Imports System.Collections
Imports System.Collections.Generic

Public class FindingExistingFilesAndDirectories
    ' These arrayLists hold the directory and file names as they are found.

    Private Shared directoryList As New List(Of String)
    Private Shared fileList As New List(Of String)

    ' Retrieves an array of all directories in the store, and
    ' displays the results.

    Public Shared Sub Main()
        ' This part of the code sets up a few directories and files in the store.
        Dim isoStore As IsolatedStorageFile = IsolatedStorageFile.GetStore(IsolatedStorageScope.User Or _
            IsolatedStorageScope.Assembly Or IsolatedStorageScope.Domain, Nothing, Nothing)
        isoStore.CreateDirectory("TopLevelDirectory")
        isoStore.CreateDirectory("TopLevelDirectory/SecondLevel")
        isoStore.CreateDirectory("AnotherTopLevelDirectory/InsideDirectory")
        isoStore.CreateFile("InTheRoot.txt")
        isoStore.CreateFile("AnotherTopLevelDirectory/InsideDirectory/HereIAm.txt")
        ' End of setup.

        Console.WriteLine()
        Console.WriteLine("Here is a list of all directories in this isolated store:")

        GetAllDirectories("*", isoStore)
        For Each directory As String In directoryList
            Console.WriteLine(directory)
        Next

        Console.WriteLine()
        Console.WriteLine("Retrieve all the files in the directory by calling the GetFiles method.")

        GetAllFiles(isoStore)
        For Each file As String In fileList
            Console.WriteLine(file)
        Next
    End Sub

    Public Shared Sub GetAllDirectories(ByVal pattern As String, ByVal storeFile As IsolatedStorageFile)
        ' Retrieve directories.
        Dim directories As String() = storeFile.GetDirectoryNames(pattern)

        For Each directory As String In directories
            ' Add the directory to the final list.
            directoryList.Add((pattern.TrimEnd(CChar("*")) + directory + "/"))
            ' Call the method again using directory.
            GetAllDirectories((pattern.TrimEnd(CChar("*")) + directory + "/*"), storeFile)
        Next
    End Sub

    Public Shared Sub GetAllFiles(ByVal storefile As IsolatedStorageFile)
        ' This adds the root to the directory list.
        directoryList.Add("*")
        For Each directory As String In directoryList
            Dim files As String() = storefile.GetFileNames(directory + "*")
            For Each dirfile As String In files
                fileList.Add(dirfile)
            Next
        Next
    End Sub
End Class

```

See also

- [IsolatedStorageFile](#)
- [Isolated Storage](#)

How to: Read and Write to Files in Isolated Storage

9/20/2022 • 2 minutes to read • [Edit Online](#)

To read from, or write to, a file in an isolated store, use an [IsolatedStorageFileStream](#) object with a stream reader ([StreamReader](#) object) or stream writer ([StreamWriter](#) object).

Example

The following code example obtains an isolated store and checks whether a file named TestStore.txt exists in the store. If it doesn't exist, it creates the file and writes "Hello Isolated Storage" to the file. If TestStore.txt already exists, the example code reads from the file.

```
using System;
using System.IO;
using System.IO.IsolatedStorage;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            IsolatedStorageFile isoStore = IsolatedStorageFile.GetStore(IsolatedStorageScope.User |
IsolatedStorageScope.Assembly, null, null);

            if (isoStore.FileExists("TestStore.txt"))
            {
                Console.WriteLine("The file already exists!");
                using (IsolatedStorageFileStream isoStream = new IsolatedStorageFileStream("TestStore.txt",
 FileMode.Open, isoStore))
                {
                    using (StreamReader reader = new StreamReader(isoStream))
                    {
                        Console.WriteLine("Reading contents:");
                        Console.WriteLine(reader.ReadToEnd());
                    }
                }
            }
            else
            {
                using (IsolatedStorageFileStream isoStream = new IsolatedStorageFileStream("TestStore.txt",
 FileMode.CreateNew, isoStore))
                {
                    using (StreamWriter writer = new StreamWriter(isoStream))
                    {
                        writer.WriteLine("Hello Isolated Storage");
                        Console.WriteLine("You have written to the file.");
                    }
                }
            }
        }
    }
}
```

```
Imports System.IO
Imports System.IO.IsolatedStorage

Module Module1

    Sub Main()
        Dim isoStore As IsolatedStorageFile = IsolatedStorageFile.GetStore(IsolatedStorageScope.User Or
IsolatedStorageScope.Assembly, Nothing, Nothing)

        If (isoStore.FileExists("TestStore.txt")) Then
            Console.WriteLine("The file already exists!")
            Using isoStream As IsolatedStorageFileStream = New IsolatedStorageFileStream("TestStore.txt",
 FileMode.Open, isoStore)
                Using reader As StreamReader = New StreamReader(isoStream)
                    Console.WriteLine("Reading contents:")
                    Console.WriteLine(reader.ReadToEnd())
                End Using
            End Using
        Else
            Using isoStream As IsolatedStorageFileStream = New IsolatedStorageFileStream("TestStore.txt",
 FileMode.CreateNew, isoStore)
                Using writer As StreamWriter = New StreamWriter(isoStream)
                    writer.WriteLine("Hello Isolated Storage")
                    Console.WriteLine("You have written to the file.")
                End Using
            End Using
        End If
    End Sub

End Module
```

See also

- [IsolatedStorageFile](#)
- [IsolatedStorageFileStream](#)
- [System.IO.FileMode](#)
- [System.IO.FileAccess](#)
- [System.IO.StreamReader](#)
- [System.IO.StreamWriter](#)
- [File and Stream I/O](#)
- [Isolated Storage](#)

How to: Delete Files and Directories in Isolated Storage

9/20/2022 • 3 minutes to read • [Edit Online](#)

You can delete directories and files within an isolated storage file. Within a store, file and directory names are operating-system dependent and are specified as relative to the root of the virtual file system. They are not case-sensitive on Windows operating systems.

The [System.IO.IsolatedStorage.IsolatedStorageFile](#) class supplies two methods for deleting directories and files: [DeleteDirectory](#) and [DeleteFile](#). An [IsolatedStorageException](#) exception is thrown if you try to delete a file or directory that does not exist. If you include a wildcard character in the name, [DeleteDirectory](#) throws an [IsolatedStorageException](#) exception, and [DeleteFile](#) throws an [ArgumentException](#) exception.

The [DeleteDirectory](#) method fails if the directory contains any files or subdirectories. You can use the [GetFileNames](#) and [GetDirectoryName](#)s methods to retrieve the existing files and directories. For more information about searching the virtual file system of a store, see [How to: Find Existing Files and Directories in Isolated Storage](#).

Example

The following code example creates and then deletes several directories and files.

```
using namespace System;
using namespace System::IO::IsolatedStorage;
using namespace System::IO;

public ref class DeletingFilesDirectories
{
public:
    static void Main()
    {
        // Get a new isolated store for this user domain and assembly.
        // Put the store into an isolatedStorageFile object.

        IsolatedStorageFile^ isoStore = IsolatedStorageFile::GetStore(IsolatedStorageScope::User |
            IsolatedStorageScope::Domain | IsolatedStorageScope::Assembly, (Type ^)nullptr, (Type ^
)nullptr);

        Console::WriteLine("Creating Directories:");

        // This code creates several different directories.

        isoStore->CreateDirectory("TopLevelDirectory");
        Console::WriteLine("TopLevelDirectory");
        isoStore->CreateDirectory("TopLevelDirectory/SecondLevel");
        Console::WriteLine("TopLevelDirectory/SecondLevel");

        // This code creates two new directories, one inside the other.

        isoStore->CreateDirectory("AnotherTopLevelDirectory/InsideDirectory");
        Console::WriteLine("AnotherTopLevelDirectory/InsideDirectory");
        Console::WriteLine();

        // This code creates a few files and places them in the directories.

        Console::WriteLine("Creating Files:");
    }
}
```

```
// This file is placed in the root.

IsolatedStorageFileStream^ isoStream1 = gcnew IsolatedStorageFileStream("InTheRoot.txt",
    FileMode::Create, isoStore);
Console::WriteLine("InTheRoot.txt");

isoStream1->Close();

// This file is placed in the InsideDirectory.

IsolatedStorageFileStream^ isoStream2 = gcnew IsolatedStorageFileStream(
    "AnotherTopLevelDirectory/InsideDirectory/HereIAm.txt", FileMode::Create, isoStore);
Console::WriteLine("AnotherTopLevelDirectory/InsideDirectory/HereIAm.txt");
Console::WriteLine();

isoStream2->Close();

Console::WriteLine("Deleting File:");

// This code deletes the HereIAm.txt file.
isoStore->DeleteFile("AnotherTopLevelDirectory/InsideDirectory/HereIAm.txt");
Console::WriteLine("AnotherTopLevelDirectory/InsideDirectory/HereIAm.txt");
Console::WriteLine();

Console::WriteLine("Deleting Directory:");

// This code deletes the InsideDirectory.

isoStore->DeleteDirectory("AnotherTopLevelDirectory/InsideDirectory/");
Console::WriteLine("AnotherTopLevelDirectory/InsideDirectory/");
Console::WriteLine();

} // End of main.
};

int main()
{
    DeletingFilesDirectories::Main();
}
```

```
using System;
using System.IO.IsolatedStorage;
using System.IO;

public class DeletingFilesDirectories
{
    public static void Main()
    {
        // Get a new isolated store for this user domain and assembly.
        // Put the store into an isolatedStorageFile object.

        IsolatedStorageFile isoStore = IsolatedStorageFile.GetStore(IsolatedStorageScope.User |
            IsolatedStorageScope.Domain | IsolatedStorageScope.Assembly, null, null);

        Console.WriteLine("Creating Directories:");

        // This code creates several different directories.

        isoStore.CreateDirectory("TopLevelDirectory");
        Console.WriteLine("TopLevelDirectory");
        isoStore.CreateDirectory("TopLevelDirectory/SecondLevel");
        Console.WriteLine("TopLevelDirectory/SecondLevel");

        // This code creates two new directories, one inside the other.

        isoStore.CreateDirectory("AnotherTopLevelDirectory/InsideDirectory");
        Console.WriteLine("AnotherTopLevelDirectory/InsideDirectory");
        Console.WriteLine();

        // This code creates a few files and places them in the directories.

        Console.WriteLine("Creating Files:");

        // This file is placed in the root.

        IsolatedStorageFileStream isoStream1 = new IsolatedStorageFileStream("InTheRoot.txt",
            FileMode.Create, isoStore);
        Console.WriteLine("InTheRoot.txt");

        isoStream1.Close();

        // This file is placed in the InsideDirectory.

        IsolatedStorageFileStream isoStream2 = new IsolatedStorageFileStream(
            "AnotherTopLevelDirectory/InsideDirectory/HereIAm.txt", FileMode.Create, isoStore);
        Console.WriteLine("AnotherTopLevelDirectory/InsideDirectory/HereIAm.txt");
        Console.WriteLine();

        isoStream2.Close();

        Console.WriteLine("Deleting File:");

        // This code deletes the HereIAm.txt file.

        isoStore.DeleteFile("AnotherTopLevelDirectory/InsideDirectory/HereIAm.txt");
        Console.WriteLine("AnotherTopLevelDirectory/InsideDirectory/HereIAm.txt");
        Console.WriteLine();

        Console.WriteLine("Deleting Directory:");

        // This code deletes the InsideDirectory.

        isoStore.DeleteDirectory("AnotherTopLevelDirectory/InsideDirectory/");
        Console.WriteLine("AnotherTopLevelDirectory/InsideDirectory/");
        Console.WriteLine();
    } // End of main.
}
```

```

Imports System.IO.IsolatedStorage
Imports System.IO

Public Class DeletingFilesDirectories
    Public Shared Sub Main()
        ' Get a new isolated store for this user domain and assembly.
        ' Put the store into an isolatedStorageFile object.

        Dim isoStore As IsolatedStorageFile = IsolatedStorageFile.GetStore(IsolatedStorageScope.User Or
            IsolatedStorageScope.Domain Or IsolatedStorageScope.Assembly, Nothing, Nothing)

        Console.WriteLine("Creating Directories:")

        ' This code creates several different directories.

        isoStore.CreateDirectory("TopLevelDirectory")
        Console.WriteLine("TopLevelDirectory")
        isoStore.CreateDirectory("TopLevelDirectory/SecondLevel")
        Console.WriteLine("TopLevelDirectory/SecondLevel")

        ' This code creates two new directories, one inside the other.

        isoStore.CreateDirectory("AnotherTopLevelDirectory/InsideDirectory")
        Console.WriteLine("AnotherTopLevelDirectory/InsideDirectory")
        Console.WriteLine()

        ' This code creates a few files and places them in the directories.

        Console.WriteLine("Creating Files:")

        ' This file is placed in the root.

        Dim isoStream1 As New IsolatedStorageFileStream("InTheRoot.txt", FileMode.Create, isoStore)
        Console.WriteLine("InTheRoot.txt")

        isoStream1.Close()

        ' This file is placed in the InsideDirectory.

        Dim isoStream2 As New IsolatedStorageFileStream(
            "AnotherTopLevelDirectory/InsideDirectory/HereIAm.txt", FileMode.Create, isoStore)
        Console.WriteLine("AnotherTopLevelDirectory/InsideDirectory/HereIAm.txt")
        Console.WriteLine()

        isoStream2.Close()

        Console.WriteLine("Deleting File:")

        ' This code deletes the HereIAm.txt file.

        isoStore.DeleteFile("AnotherTopLevelDirectory/InsideDirectory/HereIAm.txt")
        Console.WriteLine("AnotherTopLevelDirectory/InsideDirectory/HereIAm.txt")
        Console.WriteLine()

        Console.WriteLine("Deleting Directory:")

        ' This code deletes the InsideDirectory.

        isoStore.DeleteDirectory("AnotherTopLevelDirectory/InsideDirectory/")
        Console.WriteLine("AnotherTopLevelDirectory/InsideDirectory/")
        Console.WriteLine()

    End Sub
End Class

```

See also

- [System.IO.IsolatedStorage.IsolatedStorageFile](#)
- [Isolated Storage](#)

Pipe Operations in .NET

9/20/2022 • 2 minutes to read • [Edit Online](#)

Pipes provide a means for interprocess communication. There are two types of pipes:

- Anonymous pipes.

Anonymous pipes provide interprocess communication on a local computer. Anonymous pipes require less overhead than named pipes but offer limited services. Anonymous pipes are one-way and cannot be used over a network. They support only a single server instance. Anonymous pipes are useful for communication between threads, or between parent and child processes where the pipe handles can be easily passed to the child process when it is created.

In .NET, you implement anonymous pipes by using the [AnonymousPipeServerStream](#) and [AnonymousPipeClientStream](#) classes.

See [How to: Use Anonymous Pipes for Local Interprocess Communication](#).

- Named pipes.

Named pipes provide interprocess communication between a pipe server and one or more pipe clients. Named pipes can be one-way or duplex. They support message-based communication and allow multiple clients to connect simultaneously to the server process using the same pipe name. Named pipes also support impersonation, which enables connecting processes to use their own permissions on remote servers.

In .NET, you implement named pipes by using the [NamedPipeServerStream](#) and [NamedPipeClientStream](#) classes.

See [How to: Use Named Pipes for Network Interprocess Communication](#).

See also

- [File and Stream I/O](#)
- [How to: Use Anonymous Pipes for Local Interprocess Communication](#)
- [How to: Use Named Pipes for Network Interprocess Communication](#)

How to: Use Anonymous Pipes for Local Interprocess Communication

9/20/2022 • 4 minutes to read • [Edit Online](#)

Anonymous pipes provide interprocess communication on a local computer. They offer less functionality than named pipes, but also require less overhead. You can use anonymous pipes to make interprocess communication on a local computer easier. You cannot use anonymous pipes for communication over a network.

To implement anonymous pipes, use the [AnonymousPipeServerStream](#) and [AnonymousPipeClientStream](#) classes.

Example 1

The following example demonstrates a way to send a string from a parent process to a child process using anonymous pipes. This example creates an [AnonymousPipeServerStream](#) object in a parent process with a [PipeDirection](#) value of [Out](#). The parent process then creates a child process by using a client handle to create an [AnonymousPipeClientStream](#) object. The child process has a [PipeDirection](#) value of [In](#).

The parent process then sends a user-supplied string to the child process. The string is displayed to the console in the child process.

The following example shows the server process.

```

#using <System.dll>
#using <System.Core.dll>

using namespace System;
using namespace System::IO;
using namespace System::IO::Pipes;
using namespace System::Diagnostics;

ref class PipeServer
{
public:
    static void Main()
    {
        Process^ pipeClient = gcnew Process();

        pipeClient->StartInfo->FileName = "pipeClient.exe";

        AnonymousPipeServerStream^ pipeServer =
            gcnew AnonymousPipeServerStream(PipeDirection::Out,
                HandleInheritability::Inheritable);

        Console::WriteLine("[SERVER] Current TransmissionMode: {0}.",
            pipeServer->TransmissionMode);

        // Pass the client process a handle to the server.
        pipeClient->StartInfo->Arguments =
            pipeServer->GetClientHandleAsString();
        pipeClient->StartInfo->UseShellExecute = false;
        pipeClient->Start();

        pipeServer->DisposeLocalCopyOfClientHandle();

        try
        {
            // Read user input and send that to the client process.
            StreamWriter^ sw = gcnew StreamWriter(pipeServer);

            sw->AutoFlush = true;
            // Send a 'sync message' and wait for client to receive it.
            sw->WriteLine("SYNC");
            pipeServer->WaitForPipeDrain();
            // Send the console input to the client process.
            Console::Write("[SERVER] Enter text: ");
            sw->WriteLine(Console::ReadLine());
            sw->Close();
        }
        // Catch the IOException that is raised if the pipe is broken
        // or disconnected.
        catch (IOException^ e)
        {
            Console::WriteLine("[SERVER] Error: {0}", e->Message);
        }
        pipeServer->Close();
        pipeClient->WaitForExit();
        pipeClient->Close();
        Console::WriteLine("[SERVER] Client quit. Server terminating.");
    }
};

int main()
{
    PipeServer::Main();
}

```

```

using System;
using System.IO;
using System.IO.Pipes;
using System.Diagnostics;

class PipeServer
{
    static void Main()
    {
        Process pipeClient = new Process();

        pipeClient.StartInfo.FileName = "pipeClient.exe";

        using (AnonymousPipeServerStream pipeServer =
            new AnonymousPipeServerStream(PipeDirection.Out,
                HandleInheritance.Inheritable))
        {
            Console.WriteLine("[SERVER] Current TransmissionMode: {0}.",
                pipeServer.TransmissionMode);

            // Pass the client process a handle to the server.
            pipeClient.StartInfo.Arguments =
                pipeServer.GetClientHandleAsString();
            pipeClient.StartInfo.UseShellExecute = false;
            pipeClient.Start();

            pipeServer.DisposeLocalCopyOfClientHandle();

            try
            {
                // Read user input and send that to the client process.
                using (StreamWriter sw = new StreamWriter(pipeServer))
                {
                    sw.AutoFlush = true;
                    // Send a 'sync message' and wait for client to receive it.
                    sw.WriteLine("SYNC");
                    pipeServer.WaitForPipeDrain();
                    // Send the console input to the client process.
                    Console.Write("[SERVER] Enter text: ");
                    sw.WriteLine(Console.ReadLine());
                }
            }
            // Catch the IOException that is raised if the pipe is broken
            // or disconnected.
            catch (IOException e)
            {
                Console.WriteLine("[SERVER] Error: {0}", e.Message);
            }
        }

        pipeClient.WaitForExit();
        pipeClient.Close();
        Console.WriteLine("[SERVER] Client quit. Server terminating.");
    }
}

```

```

Imports System.IO
Imports System.IO.Pipes
Imports System.Diagnostics

Class PipeServer
    Shared Sub Main()
        Dim pipeClient As New Process()

        pipeClient.StartInfo.FileName = "pipeClient.exe"

        Using pipeServer As New AnonymousPipeServerStream(PipeDirection.Out, _
            HandleInheritability.Inheritable)

            Console.WriteLine("[SERVER] Current TransmissionMode: {0}.",
                pipeServer.TransmissionMode)

            ' Pass the client process a handle to the server.
            pipeClient.StartInfo.Arguments = pipeServer.GetClientHandleAsString()
            pipeClient.StartInfo.UseShellExecute = false
            pipeClient.Start()

            pipeServer.DisposeLocalCopyOfClientHandle()

        Try
            ' Read user input and send that to the client process.
            Using sw As New StreamWriter(pipeServer)
                sw.AutoFlush = true
                ' Send a 'sync message' and wait for client to receive it.
                sw.WriteLine("SYNC")
                pipeServer.WaitForPipeDrain()
                ' Send the console input to the client process.
                Console.Write("[SERVER] Enter text: ")
                sw.WriteLine(Console.ReadLine())
            End Using
            Catch e As IOException
                ' Catch the IOException that is raised if the pipe is broken
                ' or disconnected.
                Console.WriteLine("[SERVER] Error: {0}", e.Message)
            End Try
        End Using

        pipeClient.WaitForExit()
        pipeClient.Close()
        Console.WriteLine("[SERVER] Client quit. Server terminating.")
    End Sub
End Class

```

If you would like to see code comments translated to languages other than English, let us know in [this GitHub discussion issue](#).

Example 2

The following example shows the client process. The server process starts the client process and gives that process a client handle. The resulting executable from the client code should be named `pipeClient.exe` and be copied to the same directory as the server executable before running the server process.

```

#using <System.Core.dll>

using namespace System;
using namespace System::IO;
using namespace System::IO::Pipes;

ref class PipeClient
{
public:
    static void Main(array<String^>^ args)
    {
        if (args->Length > 1)
        {
            PipeStream^ pipeClient = gcnew AnonymousPipeClientStream(PipeDirection::In, args[1]);

            Console::WriteLine("[CLIENT] Current TransmissionMode: {0}.",
                pipeClient->TransmissionMode);

            StreamReader^ sr = gcnew StreamReader(pipeClient);

            // Display the read text to the console
            String^ temp;

            // Wait for 'sync message' from the server.
            do
            {
                Console::WriteLine("[CLIENT] Wait for sync...");  

                temp = sr->ReadLine();
            }
            while (!temp->StartsWith("SYNC"));

            // Read the server data and echo to the console.
            while ((temp = sr->ReadLine()) != nullptr)
            {
                Console::WriteLine("[CLIENT] Echo: " + temp);
            }
            sr->Close();
            pipeClient->Close();
        }
        Console::Write("[CLIENT] Press Enter to continue...");  

        Console::ReadLine();
    }
};

int main()
{
    array<String^>^ args = Environment::GetCommandLineArgs();
    PipeClient::Main(args);
}

```

```
using System;
using System.IO;
using System.IO.Pipes;

class PipeClient
{
    static void Main(string[] args)
    {
        if (args.Length > 0)
        {
            using (PipeStream pipeClient =
                new AnonymousPipeClientStream(PipeDirection.In, args[0]))
            {
                Console.WriteLine("[CLIENT] Current TransmissionMode: {0}.",
                    pipeClient.TransmissionMode);

                using (StreamReader sr = new StreamReader(pipeClient))
                {
                    // Display the read text to the console
                    string temp;

                    // Wait for 'sync message' from the server.
                    do
                    {
                        Console.WriteLine("[CLIENT] Wait for sync...");
                        temp = sr.ReadLine();
                    }
                    while (!temp.StartsWith("SYNC"));

                    // Read the server data and echo to the console.
                    while ((temp = sr.ReadLine()) != null)
                    {
                        Console.WriteLine("[CLIENT] Echo: " + temp);
                    }
                }
            }
            Console.Write("[CLIENT] Press Enter to continue...");  

            Console.ReadLine();
        }
    }
}
```

```
Imports System.IO
Imports System.IO.Pipes

Class PipeClient
    Shared Sub Main(args() As String)
        If args.Length > 0 Then
            Using pipeClient As New AnonymousPipeClientStream(PipeDirection.In, args(0))
                Console.WriteLine("[CLIENT] Current TransmissionMode: {0}.", _
                    pipeClient.TransmissionMode)

                Using sr As New StreamReader(pipeClient)
                    ' Display the read text to the console
                    Dim temp As String

                    ' Wait for 'sync message' from the server.
                    Do
                        Console.WriteLine("[CLIENT] Wait for sync...")
                        temp = sr.ReadLine()
                    Loop While temp.StartsWith("SYNC") = False

                    ' Read the server data and echo to the console.
                    temp = sr.ReadLine()
                    While Not temp = Nothing
                        Console.WriteLine("[CLIENT] Echo: " + temp)
                        temp = sr.ReadLine()
                    End While
                End Using
            End Using
        End If
        Console.Write("[CLIENT] Press Enter to continue...")
        Console.ReadLine()
    End Sub
End Class
```

See also

- [Pipes](#)
- [How to: Use Named Pipes for Network Interprocess Communication](#)

How to: Use Named Pipes for Network Interprocess Communication

9/20/2022 • 11 minutes to read • [Edit Online](#)

Named pipes provide interprocess communication between a pipe server and one or more pipe clients. They offer more functionality than anonymous pipes, which provide interprocess communication on a local computer. Named pipes support full duplex communication over a network and multiple server instances, message-based communication, and client impersonation, which enables connecting processes to use their own set of permissions on remote servers.

To implement name pipes, use the [NamedPipeServerStream](#) and [NamedPipeClientStream](#) classes.

Example 1

The following example demonstrates how to create a named pipe by using the [NamedPipeServerStream](#) class. In this example, the server process creates four threads. Each thread can accept a client connection. The connected client process then supplies the server with a file name. If the client has sufficient permissions, the server process opens the file and sends its contents back to the client.

```
#using <System.Core.dll>

using namespace System;
using namespace System::IO;
using namespace System::IO::Pipes;
using namespace System::Text;
using namespace System::Threading;

// Defines the data protocol for reading and writing strings on our stream
public ref class StreamString
{
private:
    Stream^ ioStream;
    UnicodeEncoding^ streamEncoding;

public:
    StreamString(Stream^ ioStream)
    {
        this->ioStream = ioStream;
        streamEncoding = gcnew UnicodeEncoding();
    }

    String^ ReadString()
    {
        int len;

        len = ioStream->.ReadByte() * 256;
        len += ioStream->.ReadByte();
        array<Byte>^ inBuffer = gcnew array<Byte>(len);
        ioStream->Read(inBuffer, 0, len);

        return streamEncoding->GetString(inBuffer);
    }

    int WriteString(String^ outString)
    {
        array<Byte>^ outBuffer = streamEncoding->GetBytes(outString);
        int len = outBuffer->Length;
        if (len > UInt16::MaxValue)
```

```

        if (len > UInt16::MaxValue)
    {
        len = (int)UInt16::MaxValue;
    }
    ioStream->WriteByte((Byte)(len / 256));
    ioStream->WriteByte((Byte)(len & 255));
    ioStream->Write(outBuffer, 0, len);
    ioStream->Flush();

    return outBuffer->Length + 2;
}
};

// Contains the method executed in the context of the impersonated user
public ref class ReadFileToStream
{
private:
    String^ fn;
    StreamString ^ss;

public:
    ReadFileToStream(StreamString^ str, String^ filename)
    {
        fn = filename;
        ss = str;
    }

    void Start()
    {
        String^ contents = File::ReadAllText(fn);
        ss->WriteString(contents);
    }
};

public ref class PipeServer
{
private:
    static int numThreads = 4;

public:
    static void Main()
    {
        int i;
        array<Thread^>^ servers = gcnew array<Thread^>(numThreads);

        Console::WriteLine("\n*** Named pipe server stream with impersonation example ***\n");
        Console::WriteLine("Waiting for client connect...\n");
        for (i = 0; i < numThreads; i++)
        {
            servers[i] = gcnew Thread(gcnew ThreadStart(&ServerThread));
            servers[i]->Start();
        }
        Thread::Sleep(250);
        while (i > 0)
        {
            for (int j = 0; j < numThreads; j++)
            {
                if (servers[j] != nullptr)
                {
                    if (servers[j]->Join(250))
                    {
                        Console::WriteLine("Server thread[{0}] finished.", servers[j]->ManagedThreadId);
                        servers[j] = nullptr;
                        i--; // decrement the thread watch count
                    }
                }
            }
        }
        Console::WriteLine("\nServer threads exhausted, exiting.");
    }
};

```

```

}

private:
    static void ServerThread()
    {
        NamedPipeServerStream^ pipeServer =
            gcnew NamedPipeServerStream("testpipe", PipeDirection::InOut, numThreads);

        int threadId = Thread::CurrentThread->ManagedThreadId;

        // Wait for a client to connect
        pipeServer->WaitForConnection();

        Console::WriteLine("Client connected on thread[{0}].", threadId);
        try
        {
            // Read the request from the client. Once the client has
            // written to the pipe its security token will be available.

            StreamString^ ss = gcnew StreamString(pipeServer);

            // Verify our identity to the connected client using a
            // string that the client anticipates.

            ss->WriteString("I am the one true server!");
            String^ filename = ss->ReadString();

            // Read in the contents of the file while impersonating the client.
            ReadFileToStream^ fileReader = gcnew ReadFileToStream(ss, filename);

            // Display the name of the user we are impersonating.
            Console::WriteLine("Reading file: {0} on thread[{1}] as user: {2}.",
                filename, threadId, pipeServer->GetImpersonationUserName());
            pipeServer->RunAsClient(gcnew PipeStreamImpersonationWorker(fileReader,
&ReadFileToStream::Start));
        }
        // Catch the IOException that is raised if the pipe is broken
        // or disconnected.
        catch (IOException^ e)
        {
            Console::WriteLine("ERROR: {0}", e->Message);
        }
        pipeServer->Close();
    }
};

int main()
{
    PipeServer::Main();
}

```

```

using System;
using System.IO;
using System.IO.Pipes;
using System.Text;
using System.Threading;

public class PipeServer
{
    private static int numThreads = 4;

    public static void Main()
    {
        int i;
        Thread[] servers = new Thread[numThreads];

        Console.WriteLine("\n*** Named pipe server stream with impersonation example ***\n");
        Console.WriteLine("Waiting for client connect...\n");

```

```

        Console.WriteLine("Waiting for client connect...{0}");  

        for (i = 0; i < numThreads; i++)  

        {  

            servers[i] = new Thread(ServerThread);  

            servers[i].Start();  

        }  

        Thread.Sleep(250);  

        while (i > 0)  

        {  

            for (int j = 0; j < numThreads; j++)  

            {  

                if (servers[j] != null)  

                {  

                    if (servers[j].Join(250))  

                    {  

                        Console.WriteLine("Server thread[{0}] finished.", servers[j].ManagedThreadId);  

                        servers[j] = null;  

                        i--; // decrement the thread watch count  

                    }  

                }  

            }  

        }  

        Console.WriteLine("\nServer threads exhausted, exiting.");
    }

    private static void ServerThread(object data)
    {
        NamedPipeServerStream pipeServer =
            new NamedPipeServerStream("testpipe", PipeDirection.InOut, numThreads);

        int threadId = Thread.CurrentThread.ManagedThreadId;

        // Wait for a client to connect
        pipeServer.WaitForConnection();

        Console.WriteLine("Client connected on thread[{0}].", threadId);
        try
        {
            // Read the request from the client. Once the client has
            // written to the pipe its security token will be available.

            StreamString ss = new StreamString(pipeServer);

            // Verify our identity to the connected client using a
            // string that the client anticipates.

            ss.WriteString("I am the one true server!");
            string filename = ss.ReadString();

            // Read in the contents of the file while impersonating the client.
            ReadFileToStream fileReader = new ReadFileToStream(ss, filename);

            // Display the name of the user we are impersonating.
            Console.WriteLine("Reading file: {0} on thread[{1}] as user: {2}.",
                filename, threadId, pipeServer.GetImpersonationUserName());
            pipeServer.RunAsClient(fileReader.Start());
        }
        // Catch the IOException that is raised if the pipe is broken
        // or disconnected.
        catch (IOException e)
        {
            Console.WriteLine("ERROR: {0}", e.Message);
        }
        pipeServer.Close();
    }

    // Defines the data protocol for reading and writing strings on our stream
    public class StreamString
    {

```

```

{
    private Stream ioStream;
    private UnicodeEncoding streamEncoding;

    public StreamString(Stream ioStream)
    {
        this.ioStream = ioStream;
        streamEncoding = new UnicodeEncoding();
    }

    public string ReadString()
    {
        int len = 0;

        len = ioStream.ReadByte() * 256;
        len += ioStream.ReadByte();
        byte[] inBuffer = new byte[len];
        ioStream.Read(inBuffer, 0, len);

        return streamEncoding.GetString(inBuffer);
    }

    public int WriteString(string outString)
    {
        byte[] outBuffer = streamEncoding.GetBytes(outString);
        int len = outBuffer.Length;
        if (len > UInt16.MaxValue)
        {
            len = (int)UInt16.MaxValue;
        }
        ioStream.WriteByte((byte)(len / 256));
        ioStream.WriteByte((byte)(len & 255));
        ioStream.Write(outBuffer, 0, len);
        ioStream.Flush();

        return outBuffer.Length + 2;
    }
}

// Contains the method executed in the context of the impersonated user
public class ReadFileToStream
{
    private string fn;
    private StreamString ss;

    public ReadFileToStream(StreamString str, string filename)
    {
        fn = filename;
        ss = str;
    }

    public void Start()
    {
        string contents = File.ReadAllText(fn);
        ss.WriteString(contents);
    }
}

```

```

Imports System.IO
Imports System.IO.Pipes
Imports System.Text
Imports System.Threading

Public Class PipeServer
    Private Shared numThreads As Integer = 4

    Public Shared Sub Main()
        Dim s As String

```

```

    Dim i As Integer
    Dim servers(numThreads) As Thread

    Console.WriteLine(vbCrLf + "*** Named pipe server stream with impersonation example ***" + vbCrLf)
    Console.WriteLine("Waiting for client connect..." + vbCrLf)
    For i = 0 To numThreads - 1
        servers(i) = New Thread(AddressOf ServerThread)
        servers(i).Start()
    Next i
    Thread.Sleep(250)
    While i > 0
        For j As Integer = 0 To numThreads - 1
            If Not (servers(j) Is Nothing) Then
                If servers(j).Join(250) Then
                    Console.WriteLine("Server thread[{0}] finished.", servers(j).ManagedThreadId)
                    servers(j) = Nothing
                    i -= 1      ' decrement the thread watch count
                End If
            End If
        Next j
    End While
    Console.WriteLine(vbCrLf + "Server threads exhausted, exiting.")
End Sub

Private Shared Sub ServerThread(data As Object)
    Dim pipeServer As New _
        NamedPipeServerStream("testpipe", PipeDirection.InOut, numThreads)

    Dim threadId As Integer = Thread.CurrentThread.ManagedThreadId

    ' Wait for a client to connect
    pipeServer.WaitForConnection()

    Console.WriteLine("Client connected on thread[{0}].", threadId)
    Try
        ' Read the request from the client. Once the client has
        ' written to the pipe its security token will be available.

        Dim ss As New StreamString(pipeServer)

        ' Verify our identity to the connected client using a
        ' string that the client anticipates.

        ss.WriteString("I am the one true server!")
        Dim filename As String = ss.ReadString()

        ' Read in the contents of the file while impersonating the client.
        Dim fileReader As New ReadFileToStream(ss, filename)

        ' Display the name of the user we are impersonating.
        Console.WriteLine("Reading file: {0} on thread[{1}] as user: {2}.",
            filename, threadId, pipeServer.GetImpersonationUserName())
        pipeServer.RunAsClient(AddressOf fileReader.Start)
        ' Catch the IOException that is raised if the pipe is broken
        ' or disconnected.
        Catch e As IOException
            Console.WriteLine("ERROR: {0}", e.Message)
        End Try
        pipeServer.Close()
    End Sub
End Class

' Defines the data protocol for reading and writing strings on our stream
Public Class StreamString
    Private ioStream As Stream
    Private streamEncoding As UnicodeEncoding

    Public Sub New(ioStream As Stream)
        Me.ioStream = ioStream
    End Sub
End Class

```

```

        streamEncoding = New UnicodeEncoding(False, False)
    End Sub

    Public Function ReadString() As String
        Dim len As Integer = 0
        len = CType(ioStream.ReadByte(), Integer) * 256
        len += CType(ioStream.ReadByte(), Integer)
        Dim inBuffer As Array = Array.CreateInstance(GetType(Byte), len)
        ioStream.Read(inBuffer, 0, len)

        Return streamEncoding.GetString(CType(inBuffer, Byte()))
    End Function

    Public Function WriteString(outString As String) As Integer
        Dim outBuffer() As Byte = streamEncoding.GetBytes(outString)
        Dim len As Integer = outBuffer.Length
        If len > UInt16.MaxValue Then
            len = CType(UInt16.MaxValue, Integer)
        End If
        ioStream.WriteByte(CType(len \ 256, Byte))
        ioStream.WriteByte(CType(len And 255, Byte))
        ioStream.Write(outBuffer, 0, outBuffer.Length)
        ioStream.Flush()

        Return outBuffer.Length + 2
    End Function
End Class

' Contains the method executed in the context of the impersonated user
Public Class ReadFileToStream
    Private fn As String
    Private ss As StreamString

    Public Sub New(str As StreamString, filename As String)
        fn = filename
        ss = str
    End Sub

    Public Sub Start()
        Dim contents As String = File.ReadAllText(fn)
        ss.WriteString(contents)
    End Sub
End Class

```

Example 2

The following example shows the client process, which uses the [NamedPipeClientStream](#) class. The client connects to the server process and sends a file name to the server. The example uses impersonation, so the identity that is running the client application must have permission to access the file. The server then sends the contents of the file back to the client. The file contents are then displayed to the console.

```

using System;
using System.Diagnostics;
using System.IO;
using System.IO.Pipes;
using System.Security.Principal;
using System.Text;
using System.Threading;

public class PipeClient
{
    private static int numClients = 4;

    public static void Main(string[] args)
    {
        ...
    }
}

```

```

if (args.Length > 0)
{
    if (args[0] == "spawnclient")
    {
        var pipeClient =
            new NamedPipeClientStream(".", "testpipe",
                PipeDirection.InOut, PipeOptions.None,
                TokenImpersonationLevel.Impersonation);

        Console.WriteLine("Connecting to server...\\n");
        pipeClient.Connect();

        var ss = new StreamString(pipeClient);
        // Validate the server's signature string.
        if (ss.ReadString() == "I am the one true server!")
        {
            // The client security token is sent with the first write.
            // Send the name of the file whose contents are returned
            // by the server.
            ss.WriteString("c:\\textfile.txt");

            // Print the file to the screen.
            Console.Write(ss.ReadString());
        }
        else
        {
            Console.WriteLine("Server could not be verified.");
        }
        pipeClient.Close();
        // Give the client process some time to display results before exiting.
        Thread.Sleep(4000);
    }
}
else
{
    Console.WriteLine("\n*** Named pipe client stream with impersonation example ***\\n");
    StartClients();
}
}

// Helper function to create pipe client processes
private static void StartClients()
{
    string currentProcessName = Environment.CommandLine;

    // Remove extra characters when launched from Visual Studio
    currentProcessName = currentProcessName.Trim(' ', ' ');

    currentProcessName = Path.ChangeExtension(currentProcessName, ".exe");
    Process[] plist = new Process[numClients];

    Console.WriteLine("Spawning client processes...\\n");

    if (currentProcessName.Contains(Environment.CurrentDirectory))
    {
        currentProcessName = currentProcessName.Replace(Environment.CurrentDirectory, String.Empty);
    }

    // Remove extra characters when launched from Visual Studio
    currentProcessName = currentProcessName.Replace("\\", String.Empty);
    currentProcessName = currentProcessName.Replace("\\", String.Empty);

    int i;
    for (i = 0; i < numClients; i++)
    {
        // Start 'this' program but spawn a named pipe client.
        plist[i] = Process.Start(currentProcessName, "spawnclient");
    }
    while (i > 0)
}

```

```

    {
        for (int j = 0; j < numClients; j++)
        {
            if (plist[j] != null)
            {
                if (plist[j].HasExited)
                {
                    Console.WriteLine($"Client process[{plist[j].Id}] has exited.");
                    plist[j] = null;
                    i--; // decrement the process watch count
                }
                else
                {
                    Thread.Sleep(250);
                }
            }
        }
        Console.WriteLine("\nClient processes finished, exiting.");
    }
}

// Defines the data protocol for reading and writing strings on our stream.
public class StreamString
{
    private Stream ioStream;
    private UnicodeEncoding streamEncoding;

    public StreamString(Stream ioStream)
    {
        this.ioStream = ioStream;
        streamEncoding = new UnicodeEncoding();
    }

    public string ReadString()
    {
        int len;
        len = ioStream.ReadByte() * 256;
        len += ioStream.ReadByte();
        var inBuffer = new byte[len];
        ioStream.Read(inBuffer, 0, len);

        return streamEncoding.GetString(inBuffer);
    }

    public int WriteString(string outString)
    {
        byte[] outBuffer = streamEncoding.GetBytes(outString);
        int len = outBuffer.Length;
        if (len > UInt16.MaxValue)
        {
            len = (int)UInt16.MaxValue;
        }
        ioStream.WriteByte((byte)(len / 256));
        ioStream.WriteByte((byte)(len & 255));
        ioStream.Write(outBuffer, 0, len);
        ioStream.Flush();

        return outBuffer.Length + 2;
    }
}

```

```

Imports System.Diagnostics
Imports System.IO
Imports System.IO.Pipes
Imports System.Security.Principal
Imports System.Text

```

```

Imports System.Threading

Public Class PipeClient
    Private Shared numClients As Integer = 4

    Public Shared Sub Main(args() As String)
        If args.Length > 0 Then
            If args(0) = "spawnclient" Then
                Dim pipeClient As New NamedPipeClientStream( _
                    ".", "testpipe", _
                    PipeDirection.InOut, PipeOptions.None, _
                    TokenImpersonationLevel.Impersonation)

                Console.WriteLine("Connecting to server..." + vbCrLf)
                pipeClient.Connect()

                Dim ss As New StreamString(pipeClient)
                ' Validate the server's signature string.
                If ss.ReadString() = "I am the one true server!" Then
                    ' The client security token is sent with the first write.
                    ' Send the name of the file whose contents are returned
                    ' by the server.
                    ss.WriteString("c:\textfile.txt")

                    ' Print the file to the screen.
                    Console.Write(ss.ReadString())
                Else
                    Console.WriteLine("Server could not be verified.")
                End If
                pipeClient.Close()
                ' Give the client process some time to display results before exiting.
                Thread.Sleep(4000)
            End If
        Else
            Console.WriteLine(vbCrLf + "*** Named pipe client stream with impersonation example ***" +
vbCrLf)
            StartClients()
        End If
    End Sub

    ' Helper function to create pipe client processes
    Private Shared Sub StartClients()
        Dim currentProcessName As String = Environment.CommandLine
        Dim plist(numClients - 1) As Process

        Console.WriteLine("Spawning client processes..." + vbCrLf)

        If currentProcessName.Contains(Environment.CurrentDirectory) Then
            currentProcessName = currentProcessName.Replace(Environment.CurrentDirectory, String.Empty)
        End If

        ' Remove extra characters when launched from Visual Studio.
        currentProcessName = currentProcessName.Replace("\", String.Empty)
        currentProcessName = currentProcessName.Replace("""", String.Empty)

        ' Change extension for .NET Core "dotnet run" returns the DLL, not the host exe.
        currentProcessName = Path.ChangeExtension(currentProcessName, ".exe")

        Dim i As Integer
        For i = 0 To numClients - 1
            ' Start 'this' program but spawn a named pipe client.
            plist(i) = Process.Start(currentProcessName, "spawnclient")
        Next
        While i > 0
            For j As Integer = 0 To numClients - 1
                If plist(j) IsNot Nothing Then
                    If plist(j).HasExited Then
                        Console.WriteLine($"Client process[{plist(j).Id}] has exited.")
                        plist(j) = Nothing
                    End If
                End If
            Next
        End While
    End Sub

```

```

        i -= 1      ' decrement the process watch count
    Else
        Thread.Sleep(250)
    End If
End If
Next
End While
Console.WriteLine(vbCrLf + "Client processes finished, exiting.")
End Sub
End Class

' Defines the data protocol for reading and writing strings on our stream
Public Class StreamString
    Private ioStream As Stream
    Private streamEncoding As UnicodeEncoding

    Public Sub New(ioStream As Stream)
        Me.ioStream = ioStream
        streamEncoding = New UnicodeEncoding(False, False)
    End Sub

    Public Function ReadString() As String
        Dim len As Integer = 0
        len = CType(ioStream.ReadByte(), Integer) * 256
        len += CType(ioStream.ReadByte(), Integer)
        Dim inBuffer As Array = Array.CreateInstance(GetType(Byte), len)
        ioStream.Read(inBuffer, 0, len)

        Return streamEncoding.GetString(CType(inBuffer, Byte()))
    End Function

    Public Function WriteString(outString As String) As Integer
        Dim outBuffer As Byte() = streamEncoding.GetBytes(outString)
        Dim len As Integer = outBuffer.Length
        If len > UInt16.MaxValue Then
            len = CType(UInt16.MaxValue, Integer)
        End If
        ioStream.WriteByte(CType(len \ 256, Byte))
        ioStream.WriteByte(CType(len And 255, Byte))
        ioStream.Write(outBuffer, 0, outBuffer.Length)
        ioStream.Flush()

        Return outBuffer.Length + 2
    End Function
End Class

```

Robust Programming

The client and server processes in this example are intended to run on the same computer, so the server name provided to the [NamedPipeClientStream](#) object is `".."`. If the client and server processes were on separate computers, `".."` would be replaced with the network name of the computer that runs the server process.

See also

- [TokenImpersonationLevel](#)
- [GetImpersonationUserName](#)
- [Pipes](#)
- [How to: Use Anonymous Pipes for Local Interprocess Communication](#)

System.IO.Pipelines in .NET

9/20/2022 • 22 minutes to read • [Edit Online](#)

[System.IO.Pipelines](#) is a library that is designed to make it easier to do high-performance I/O in .NET. It's a library targeting .NET Standard that works on all .NET implementations.

The library is available in the [System.IO.Pipelines](#) Nuget package.

What problem does System.IO.Pipelines solve

Apps that parse streaming data are composed of boilerplate code having many specialized and unusual code flows. The boilerplate and special case code is complex and difficult to maintain.

`System.IO.Pipelines` was architected to:

- Have high performance parsing streaming data.
- Reduce code complexity.

The following code is typical for a TCP server that receives line-delimited messages (delimited by `'\n'`) from a client:

```
async Task ProcessLinesAsync(NetworkStream stream)
{
    var buffer = new byte[1024];
    await stream.ReadAsync(buffer, 0, buffer.Length);

    // Process a single line from the buffer
    ProcessLine(buffer);
}
```

The preceding code has several problems:

- The entire message (end of line) might not be received in a single call to `ReadAsync`.
- It's ignoring the result of `stream.ReadAsync`. `stream.ReadAsync` returns how much data was read.
- It doesn't handle the case where multiple lines are read in a single `ReadAsync` call.
- It allocates a `byte` array with each read.

To fix the preceding problems, the following changes are required:

- Buffer the incoming data until a new line is found.
- Parse all the lines returned in the buffer.
- It's possible that the line is bigger than 1 KB (1024 bytes). The code needs to resize the input buffer until the delimiter is found in order to fit the complete line inside the buffer.
 - If the buffer is resized, more buffer copies are made as longer lines appear in the input.
 - To reduce wasted space, compact the buffer used for reading lines.
- Consider using buffer pooling to avoid allocating memory repeatedly.
- The following code addresses some of these problems:

```

async Task ProcessLinesAsync(NetworkStream stream)
{
    byte[] buffer = ArrayPool<byte>.Shared.Rent(1024);
    var bytesBuffered = 0;
    var bytesConsumed = 0;

    while (true)
    {
        // Calculate the amount of bytes remaining in the buffer.
        var bytesRemaining = buffer.Length - bytesBuffered;

        if (bytesRemaining == 0)
        {
            // Double the buffer size and copy the previously buffered data into the new buffer.
            var newBuffer = ArrayPool<byte>.Shared.Rent(buffer.Length * 2);
            Buffer.BlockCopy(buffer, 0, newBuffer, 0, buffer.Length);
            // Return the old buffer to the pool.
            ArrayPool<byte>.Shared.Return(buffer);
            buffer = newBuffer;
            bytesRemaining = buffer.Length - bytesBuffered;
        }

        var bytesRead = await stream.ReadAsync(buffer, bytesBuffered, bytesRemaining);
        if (bytesRead == 0)
        {
            // EOF
            break;
        }

        // Keep track of the amount of buffered bytes.
        bytesBuffered += bytesRead;
        var linePosition = -1;

        do
        {
            // Look for a EOL in the buffered data.
            linePosition = Array.IndexOf(buffer, (byte)'\\n', bytesConsumed,
                bytesBuffered - bytesConsumed);

            if (linePosition >= 0)
            {
                // Calculate the length of the line based on the offset.
                var lineLength = linePosition - bytesConsumed;

                // Process the line.
                ProcessLine(buffer, bytesConsumed, lineLength);

                // Move the bytesConsumed to skip past the line consumed (including \\n).
                bytesConsumed += lineLength + 1;
            }
        }
        while (linePosition >= 0);
    }
}

```

The previous code is complex and doesn't address all the problems identified. High-performance networking usually means writing complex code to maximize performance. `System.IO.Pipelines` was designed to make writing this type of code easier.

If you would like to see code comments translated to languages other than English, let us know in [this GitHub discussion issue](#).

Pipe

The [Pipe](#) class can be used to create a [PipeWriter/PipeReader](#) pair. All data written into the [PipeWriter](#) is available in the [PipeReader](#):

```
var pipe = new Pipe();
PipeReader reader = pipe.Reader;
PipeWriter writer = pipe.Writer;
```

Pipe basic usage

```
async Task ProcessLinesAsync(Socket socket)
{
    var pipe = new Pipe();
    Task writing = FillPipeAsync(socket, pipe.Writer);
    Task reading = ReadPipeAsync(pipe.Reader);

    await Task.WhenAll(reading, writing);
}

async Task FillPipeAsync(Socket socket, PipeWriter writer)
{
    const int minimumBufferSize = 512;

    while (true)
    {
        // Allocate at least 512 bytes from the PipeWriter.
        Memory<byte> memory = writer.GetMemory(minimumBufferSize);
        try
        {
            int bytesRead = await socket.ReceiveAsync(memory, SocketFlags.None);
            if (bytesRead == 0)
            {
                break;
            }
            // Tell the PipeWriter how much was read from the Socket.
            writer.Advance(bytesRead);
        }
        catch (Exception ex)
        {
            LogError(ex);
            break;
        }

        // Make the data available to the PipeReader.
        FlushResult result = await writer.FlushAsync();

        if (result.IsCompleted)
        {
            break;
        }
    }

    // By completing PipeWriter, tell the PipeReader that there's no more data coming.
    await writer.CompleteAsync();
}

async Task ReadPipeAsync(PipeReader reader)
{
    while (true)
    {
        ReadResult result = await reader.ReadAsync();
        ReadOnlySequence<byte> buffer = result.Buffer;

        while (TryReadLine(ref buffer, out ReadOnlySequence<byte> line))
        {
            // Process the line.
        }
    }
}
```

```

        ProcessLine(line);
    }

    // Tell the PipeReader how much of the buffer has been consumed.
    reader.AdvanceTo(buffer.Start, buffer.End);

    // Stop reading if there's no more data coming.
    if (result.IsCompleted)
    {
        break;
    }
}

// Mark the PipeReader as complete.
await reader.CompleteAsync();
}

bool TryReadLine(ref ReadOnlySequence<byte> buffer, out ReadOnlySequence<byte> line)
{
    // Look for a EOL in the buffer.
    SequencePosition? position = buffer.PositionOf((byte)'\n');

    if (position == null)
    {
        line = default;
        return false;
    }

    // Skip the line + the \n.
    line = buffer.Slice(0, position.Value);
    buffer = buffer.Slice(buffer.GetPosition(1, position.Value));
    return true;
}
}

```

There are two loops:

- `FillPipeAsync` reads from the `Socket` and writes to the `PipeWriter`.
- `ReadPipeAsync` reads from the `PipeReader` and parses incoming lines.

There are no explicit buffers allocated. All buffer management is delegated to the `PipeReader` and `PipeWriter` implementations. Delegating buffer management makes it easier for consuming code to focus solely on the business logic.

In the first loop:

- `PipeWriter.GetMemory(Int32)` is called to get memory from the underlying writer.
- `PipeWriter.Advance(Int32)` is called to tell the `PipeWriter` how much data was written to the buffer.
- `PipeWriter.FlushAsync` is called to make the data available to the `PipeReader`.

In the second loop, the `PipeReader` consumes the buffers written by `PipeWriter`. The buffers come from the socket. The call to `PipeReader.ReadAsync`:

- Returns a `ReadResult` that contains two important pieces of information:
 - The data that was read in the form of `ReadOnlySequence<byte>`.
 - A boolean `IsCompleted` that indicates if the end of data (EOF) has been reached.

After finding the end of line (EOL) delimiter and parsing the line:

- The logic processes the buffer to skip what's already processed.
- `PipeReader.AdvanceTo` is called to tell the `PipeReader` how much data has been consumed and examined.

The reader and writer loops end by calling `Complete`. `Complete` lets the underlying Pipe release the memory it

allocated.

Backpressure and flow control

Ideally, reading and parsing work together:

- The reading thread consumes data from the network and puts it in buffers.
- The parsing thread is responsible for constructing the appropriate data structures.

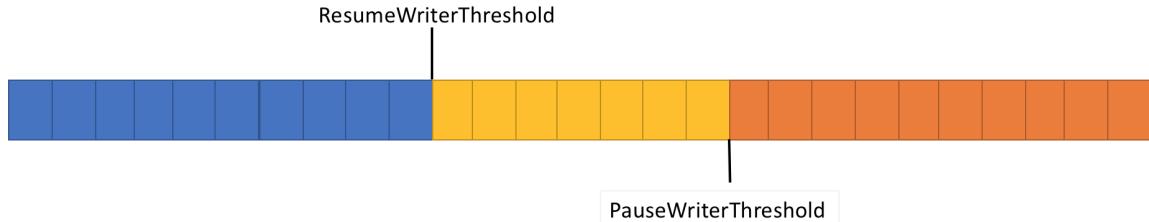
Typically, parsing takes more time than just copying blocks of data from the network:

- The reading thread gets ahead of the parsing thread.
- The reading thread has to either slow down or allocate more memory to store the data for the parsing thread.

For optimal performance, there's a balance between frequent pauses and allocating more memory.

To solve the preceding problem, the `Pipe` has two settings to control the flow of data:

- `PauseWriterThreshold`: Determines how much data should be buffered before calls to `FlushAsync` pause.
- `ResumeWriterThreshold`: Determines how much data the reader has to observe before calls to `PipeWriter.FlushAsync` resume.



`PipeWriter.FlushAsync`:

- Returns an incomplete `ValueTask<FlushResult>` when the amount of data in the `Pipe` crosses `PauseWriterThreshold`.
- Completes `valueTask<FlushResult>` when it becomes lower than `ResumeWriterThreshold`.

Two values are used to prevent rapid cycling, which can occur if one value is used.

Examples

```
// The Pipe will start returning incomplete tasks from FlushAsync until
// the reader examines at least 5 bytes.
var options = new PipeOptions(pauseWriterThreshold: 10, resumeWriterThreshold: 5);
var pipe = new Pipe(options);
```

PipeScheduler

Typically when using `async` and `await`, asynchronous code resumes on either a `TaskScheduler` or the current `SynchronizationContext`.

When doing I/O, it's important to have fine-grained control over where the I/O is performed. This control allows taking advantage of CPU caches effectively. Efficient caching is critical for high-performance apps like web servers. `PipeScheduler` provides control over where asynchronous callbacks run. By default:

- The current `SynchronizationContext` is used.
- If there's no `SynchronizationContext`, it uses the thread pool to run callbacks.

```

public static void Main(string[] args)
{
    var writeScheduler = new SingleThreadPipeScheduler();
    var readScheduler = new SingleThreadPipeScheduler();

    // Tell the Pipe what schedulers to use and disable the SynchronizationContext.
    var options = new PipeOptions(readerScheduler: readScheduler,
                                  writerScheduler: writeScheduler,
                                  useSynchronizationContext: false);
    var pipe = new Pipe(options);
}

// This is a sample scheduler that async callbacks on a single dedicated thread.
public class SingleThreadPipeScheduler : PipeScheduler
{
    private readonly BlockingCollection<(Action<object> Action, object State)> _queue =
        new BlockingCollection<(Action<object> Action, object State)>();
    private readonly Thread _thread;

    public SingleThreadPipeScheduler()
    {
        _thread = new Thread(DoWork);
        _thread.Start();
    }

    private void DoWork()
    {
        foreach (var item in _queue.GetConsumingEnumerable())
        {
            item.Action(item.State);
        }
    }

    public override void Schedule(Action<object?> action, object? state)
    {
        if (state is not null)
        {
            _queue.Add((action, state));
        }
        // else log the fact that _queue.Add was not called.
    }
}

```

[PipeScheduler.ThreadPool](#) is the [PipeScheduler](#) implementation that queues callbacks to the thread pool.

[PipeScheduler.ThreadPool](#) is the default and generally the best choice. [PipeScheduler.Inline](#) can cause unintended consequences such as deadlocks.

Pipe reset

It's frequently efficient to reuse the [Pipe](#) object. To reset the pipe, call [PipeReader.Reset](#) when both the [PipeReader](#) and [PipeWriter](#) are complete.

PipeReader

[PipeReader](#) manages memory on the caller's behalf. **Always** call [PipeReader.AdvanceTo](#) after calling [PipeReader.ReadAsync](#). This lets the [PipeReader](#) know when the caller is done with the memory so that it can be tracked. The [ReadOnlySequence<byte>](#) returned from [PipeReader.ReadAsync](#) is only valid until the call the [PipeReader.AdvanceTo](#). It's illegal to use [ReadOnlySequence<byte>](#) after calling [PipeReader.AdvanceTo](#).

[PipeReader.AdvanceTo](#) takes two [SequencePosition](#) arguments:

- The first argument determines how much memory was consumed.
- The second argument determines how much of the buffer was observed.

Marking data as consumed means that the pipe can return the memory to the underlying buffer pool. Marking data as observed controls what the next call to `PipeReader.ReadAsync` does. Marking everything as observed means that the next call to `PipeReader.ReadAsync` won't return until there's more data written to the pipe. Any other value will make the next call to `PipeReader.ReadAsync` return immediately with the observed *and* unobserved data, but not data that has already been consumed.

Read streaming data scenarios

There are a couple of typical patterns that emerge when trying to read streaming data:

- Given a stream of data, parse a single message.
- Given a stream of data, parse all available messages.

The following examples use the `TryParseLines` method for parsing messages from a `ReadOnlySequence<byte>`.

`TryParseLines` parses a single message and updates the input buffer to trim the parsed message from the buffer. `TryParseLines` isn't part of .NET, it's a user written method used in the following sections.

```
bool TryParseLines(ref ReadOnlySequence<byte> buffer, out Message message);
```

Read a single message

The following code reads a single message from a `PipeReader` and returns it to the caller.

```

async ValueTask<Message?> ReadSingleMessageAsync(PipeReader reader,
    CancellationToken cancellationToken = default)
{
    while (true)
    {
        ReadResult result = await reader.ReadAsync(cancellationToken);
        ReadOnlySequence<byte> buffer = result.Buffer;

        // In the event that no message is parsed successfully, mark consumed
        // as nothing and examined as the entire buffer.
        SequencePosition consumed = buffer.Start;
        SequencePosition examined = buffer.End;

        try
        {
            if (TryParseLines(ref buffer, out Message message))
            {
                // A single message was successfully parsed so mark the start of the
                // parsed buffer as consumed. TryParseLines trims the buffer to
                // point to the data after the message was parsed.
                consumed = buffer.Start;

                // Examined is marked the same as consumed here, so the next call
                // to ReadSingleMessageAsync will process the next message if there's
                // one.
                examined = consumed;

                return message;
            }

            // There's no more data to be processed.
            if (result.IsCompleted)
            {
                if (buffer.Length > 0)
                {
                    // The message is incomplete and there's no more data to process.
                    throw new InvalidDataException("Incomplete message.");
                }

                break;
            }
        }
        finally
        {
            reader.AdvanceTo(consumed, examined);
        }
    }

    return null;
}

```

The preceding code:

- Parses a single message.
- Updates the consumed `SequencePosition` and examined `SequencePosition` to point to the start of the trimmed input buffer.

The two `SequencePosition` arguments are updated because `TryParseLines` removes the parsed message from the input buffer. Generally, when parsing a single message from the buffer, the examined position should be one of the following:

- The end of the message.
- The end of the received buffer if no message was found.

The single message case has the most potential for errors. Passing the wrong values to `examined` can result in an out of memory exception or an infinite loop. For more information, see the [PipeReader common problems](#) section in this article.

Reading multiple messages

The following code reads all messages from a `PipeReader` and calls `ProcessMessageAsync` on each.

```
async Task ProcessMessagesAsync(PipeReader reader, CancellationToken cancellationToken = default)
{
    try
    {
        while (true)
        {
            ReadResult result = await reader.ReadAsync(cancellationToken);
            ReadOnlySequence<byte> buffer = result.Buffer;

            try
            {
                // Process all messages from the buffer, modifying the input buffer on each
                // iteration.
                while (TryParseLines(ref buffer, out Message message))
                {
                    await ProcessMessageAsync(message);
                }

                // There's no more data to be processed.
                if (result.IsCompleted)
                {
                    if (buffer.Length > 0)
                    {
                        // The message is incomplete and there's no more data to process.
                        throw new InvalidDataException("Incomplete message.");
                    }
                    break;
                }
            }
            finally
            {
                // Since all messages in the buffer are being processed, you can use the
                // remaining buffer's Start and End position to determine consumed and examined.
                reader.AdvanceTo(buffer.Start, buffer.End);
            }
        }
    }
    finally
    {
        await reader.CompleteAsync();
    }
}
```

Cancellation

`PipeReader.ReadAsync`:

- Supports passing a `CancellationToken`.
- Throws an `OperationCanceledException` if the `CancellationToken` is canceled while there's a read pending.
- Supports a way to cancel the current read operation via `PipeReader.CancelPendingRead`, which avoids raising an exception. Calling `PipeReader.CancelPendingRead` causes the current or next call to `PipeReader.ReadAsync` to return a `ReadResult` with `IsCanceled` set to `true`. This can be useful for halting the existing read loop in a non-destructive and non-exceptional way.

```

private PipeReader reader;

public MyConnection(PipeReader reader)
{
    this.reader = reader;
}

public void Abort()
{
    // Cancel the pending read so the process loop ends without an exception.
    reader.CancelPendingRead();
}

public async Task ProcessMessagesAsync()
{
    try
    {
        while (true)
        {
            ReadResult result = await reader.ReadAsync();
            ReadOnlySequence<byte> buffer = result.Buffer;

            try
            {
                if (result.IsCanceled)
                {
                    // The read was canceled. You can quit without reading the existing data.
                    break;
                }

                // Process all messages from the buffer, modifying the input buffer on each
                // iteration.
                while (TryParseLines(ref buffer, out Message message))
                {
                    await ProcessMessageAsync(message);
                }
            }

            // There's no more data to be processed.
            if (result.IsCompleted)
            {
                break;
            }
        }
        finally
        {
            // Since all messages in the buffer are being processed, you can use the
            // remaining buffer's Start and End position to determine consumed and examined.
            reader.AdvanceTo(buffer.Start, buffer.End);
        }
    }
    finally
    {
        await reader.CompleteAsync();
    }
}

```

PipeReader common problems

- Passing the wrong values to `consumed` or `examined` may result in reading already read data.
- Passing `buffer.End` as examined may result in:
 - Stalled data
 - Possibly an eventual Out of Memory (OOM) exception if data isn't consumed. For example,

`PipeReader.AdvanceTo(position, buffer.End)` when processing a single message at a time from the buffer.

- Passing the wrong values to `consumed` or `examined` may result in an infinite loop. For example, `PipeReader.AdvanceTo(buffer.Start)` if `buffer.Start` hasn't changed will cause the next call to `PipeReader.ReadAsync` to return immediately before new data arrives.
- Passing the wrong values to `consumed` or `examined` may result in infinite buffering (eventual OOM).
- Using the `ReadOnlySequence<byte>` after calling `PipeReader.AdvanceTo` may result in memory corruption (use after free).
- Failing to call `PipeReader.Complete/CompleteAsync` may result in a memory leak.
- Checking `ReadResult.IsCompleted` and exiting the reading logic before processing the buffer results in data loss. The loop exit condition should be based on `ReadResult.Buffer.IsEmpty` and `ReadResult.IsCompleted`. Doing this incorrectly could result in an infinite loop.

Problematic code

✗ Data loss

The `ReadResult` can return the final segment of data when `IsCompleted` is set to `true`. Not reading that data before exiting the read loop will result in data loss.

WARNING

Do NOT use the following code. Using this sample will result in data loss, hangs, security issues and should NOT be copied. The following sample is provided to explain [PipeReader Common problems](#).

```
Environment.FailFast("This code is terrible, don't use it!");
while (true)
{
    ReadResult result = await reader.ReadAsync(cancellationToken);
    ReadOnlySequence<byte> dataLossBuffer = result.Buffer;

    if (result.IsCompleted)
        break;

    Process(ref dataLossBuffer, out Message message);

    reader.AdvanceTo(dataLossBuffer.Start, dataLossBuffer.End);
}
```

WARNING

Do NOT use the preceding code. Using this sample will result in data loss, hangs, security issues and should NOT be copied. The preceding sample is provided to explain [PipeReader Common problems](#).

✗ Infinite loop

The following logic may result in an infinite loop if the `Result.IsCompleted` is `true` but there's never a complete message in the buffer.

WARNING

Do NOT use the following code. Using this sample will result in data loss, hangs, security issues and should NOT be copied. The following sample is provided to explain [PipeReader Common problems](#).

```
Environment.FailFast("This code is terrible, don't use it!");
while (true)
{
    ReadResult result = await reader.ReadAsync(cancellationToken);
    ReadOnlySequence<byte> infiniteLoopBuffer = result.Buffer;
    if (result.IsCompleted && infiniteLoopBuffer.IsEmpty)
        break;

    Process(ref infiniteLoopBuffer, out Message message);

    reader.AdvanceTo(infiniteLoopBuffer.Start, infiniteLoopBuffer.End);
}
```

WARNING

Do NOT use the preceding code. Using this sample will result in data loss, hangs, security issues and should NOT be copied. The preceding sample is provided to explain [PipeReader Common problems](#).

Here's another piece of code with the same problem. It's checking for a non-empty buffer before checking

`ReadResult.IsCompleted`. Because it's in an `else if`, it will loop forever if there's never a complete message in the buffer.

WARNING

Do NOT use the following code. Using this sample will result in data loss, hangs, security issues and should NOT be copied. The following sample is provided to explain [PipeReader Common problems](#).

```
Environment.FailFast("This code is terrible, don't use it!");
while (true)
{
    ReadResult result = await reader.ReadAsync(cancellationToken);
    ReadOnlySequence<byte> infiniteLoopBuffer = result.Buffer;

    if (!infiniteLoopBuffer.IsEmpty)
        Process(ref infiniteLoopBuffer, out Message message);

    else if (result.IsCompleted)
        break;

    reader.AdvanceTo(infiniteLoopBuffer.Start, infiniteLoopBuffer.End);
}
```

WARNING

Do NOT use the preceding code. Using this sample will result in data loss, hangs, security issues and should NOT be copied. The preceding sample is provided to explain [PipeReader Common problems](#).

✗ Unresponsive application

Unconditionally calling `PipeReader.AdvanceTo` with `buffer.End` in the `examined` position may result in the

application becoming unresponsive when parsing a single message. The next call to `PipeReader.AdvanceTo` won't return until:

- There's more data written to the pipe.
- And the new data wasn't previously examined.

WARNING

Do NOT use the following code. Using this sample will result in data loss, hangs, security issues and should NOT be copied. The following sample is provided to explain [PipeReader Common problems](#).

```
Environment.FailFast("This code is terrible, don't use it!");
while (true)
{
    ReadResult result = await reader.ReadAsync(cancellationToken);
    ReadOnlySequence<byte> hangBuffer = result.Buffer;

    Process(ref hangBuffer, out Message message);

    if (result.IsCompleted)
        break;

    reader.AdvanceTo(hangBuffer.Start, hangBuffer.End);

    if (message != null)
        return message;
}
```

WARNING

Do NOT use the preceding code. Using this sample will result in data loss, hangs, security issues and should NOT be copied. The preceding sample is provided to explain [PipeReader Common problems](#).

✖ Out of Memory (OOM)

With the following conditions, the following code keeps buffering until an [OutOfMemoryException](#) occurs:

- There's no maximum message size.
- The data returned from the `PipeReader` doesn't make a complete message. For example, it doesn't make a complete message because the other side is writing a large message (For example, a 4-GB message).

WARNING

Do NOT use the following code. Using this sample will result in data loss, hangs, security issues and should NOT be copied. The following sample is provided to explain [PipeReader Common problems](#).

```
Environment.FailFast("This code is terrible, don't use it!");
while (true)
{
    ReadResult result = await reader.ReadAsync(cancellationToken);
    ReadOnlySequence<byte> thisCouldOutOfMemory = result.Buffer;

    Process(ref thisCouldOutOfMemory, out Message message);

    if (result.IsCompleted)
        break;

    reader.AdvanceTo(thisCouldOutOfMemory.Start, thisCouldOutOfMemory.End);

    if (message != null)
        return message;
}
```

WARNING

Do NOT use the preceding code. Using this sample will result in data loss, hangs, security issues and should NOT be copied. The preceding sample is provided to explain [PipeReader Common problems](#).

✗ Memory Corruption

When writing helpers that read the buffer, any returned payload should be copied before calling `Advance`. The following example will return memory that the `Pipe` has discarded and may reuse it for the next operation (read/write).

WARNING

Do NOT use the following code. Using this sample will result in data loss, hangs, security issues and should NOT be copied. The following sample is provided to explain [PipeReader Common problems](#).

```
public class Message
{
    public ReadOnlySequence<byte> CorruptedPayload { get; set; }
}
```

```

Environment.FailFast("This code is terrible, don't use it!");
Message message = null;

while (true)
{
    ReadResult result = await reader.ReadAsync(cancellationToken);
    ReadOnlySequence<byte> buffer = result.Buffer;

    ReadHeader(ref buffer, out int length);

    if (length <= buffer.Length)
    {
        message = new Message
        {
            // Slice the payload from the existing buffer
            CorruptedPayload = buffer.Slice(0, length)
        };

        buffer = buffer.Slice(length);
    }

    if (result.IsCompleted)
        break;

    reader.AdvanceTo(buffer.Start, buffer.End);

    if (message != null)
    {
        // This code is broken since reader.AdvanceTo() was called with a position *after* the buffer
        // was captured.
        break;
    }
}

return message;
}

```

WARNING

Do NOT use the preceding code. Using this sample will result in data loss, hangs, security issues and should NOT be copied. The preceding sample is provided to explain [PipeReader Common problems](#).

PipeWriter

The [PipeWriter](#) manages buffers for writing on the caller's behalf. `PipeWriter` implements `IBufferWriter<byte>`. `IBufferWriter<byte>` makes it possible to get access to buffers to perform writes without extra buffer copies.

```

async Task WriteHelloAsync(PipeWriter writer, CancellationToken cancellationToken = default)
{
    // Request at least 5 bytes from the PipeWriter.
    Memory<byte> memory = writer.GetMemory(5);

    // Write directly into the buffer.
    int written = Encoding.ASCII.GetBytes("Hello".AsSpan(), memory.Span);

    // Tell the writer how many bytes were written.
    writer.Advance(written);

    await writer.FlushAsync(cancellationToken);
}

```

The previous code:

- Requests a buffer of at least 5 bytes from the `PipeWriter` using `GetMemory`.
- Writes bytes for the ASCII string `"Hello"` to the returned `Memory<byte>`.
- Calls `Advance` to indicate how many bytes were written to the buffer.
- Flushes the `PipeWriter`, which sends the bytes to the underlying device.

The previous method of writing uses the buffers provided by the `PipeWriter`. It could also have used `PipeWriter.WriteAsync`, which:

- Copies the existing buffer to the `PipeWriter`.
- Calls `GetSpan`, `Advance` as appropriate and calls `FlushAsync`.

```
async Task WriteHelloAsync(PipeWriter writer, CancellationToken cancellationToken = default)
{
    byte[] helloBytes = Encoding.ASCII.GetBytes("Hello");

    // Write helloBytes to the writer, there's no need to call Advance here
    // (Write does that).
    await writer.WriteAsync(helloBytes, cancellationToken);
}
```

Cancellation

`FlushAsync` supports passing a `CancellationToken`. Passing a `CancellationToken` results in an `OperationCanceledException` if the token is canceled while there's a flush pending. `PipeWriter.FlushAsync` supports a way to cancel the current flush operation via `PipeWriter.CancelPendingFlush` without raising an exception. Calling `PipeWriter.CancelPendingFlush` causes the current or next call to `PipeWriter.FlushAsync` or `PipeWriter.WriteAsync` to return a `FlushResult` with `IsCanceled` set to `true`. This can be useful for halting the yielding flush in a non-destructive and non-exceptional way.

PipeWriter common problems

- `GetSpan` and `GetMemory` return a buffer with at least the requested amount of memory. Don't assume exact buffer sizes.
- There's no guarantee that successive calls will return the same buffer or the same-sized buffer.
- A new buffer must be requested after calling `Advance` to continue writing more data. The previously acquired buffer can't be written to.
- Calling `GetMemory` or `GetSpan` while there's an incomplete call to `FlushAsync` isn't safe.
- Calling `Complete` or `CompleteAsync` while there's unflushed data can result in memory corruption.

Tips for using PipeReader and PipeWriter

The following tips will help you use the `System.IO.Pipelines` classes successfully:

- Always complete the `PipeReader` and `PipeWriter`, including an exception where applicable.
- Always call `PipeReader.AdvanceTo` after calling `PipeReader.ReadAsync`.
- Periodically `await PipeWriter.FlushAsync` while writing, and always check `FlushResult.IsCompleted`. Abort writing if `IsCompleted` is `true`, as that indicates the reader is completed and no longer cares about what is written.
- Do call `PipeWriter.FlushAsync` after writing something that you want the `PipeReader` to have access to.
- Do not call `FlushAsync` if the reader can't start until `FlushAsync` finishes, as that may cause a deadlock.
- Ensure that only one context "owns" a `PipeReader` or `PipeWriter` or accesses them. These types are not thread-safe.

- Never access a `ReadResult.Buffer` after calling `AdvanceTo` or completing the `PipeReader`.

IDuplexPipe

The `IDuplexPipe` is a contract for types that support both reading and writing. For example, a network connection would be represented by an `IDuplexPipe`.

Unlike `Pipe`, which contains a `PipeReader` and a `PipeWriter`, `IDuplexPipe` represents a single side of a full duplex connection. That means what is written to the `PipeWriter` will not be read from the `PipeReader`.

Streams

When reading or writing stream data, you typically read data using a de-serializer and write data using a serializer. Most of these read and write stream APIs have a `Stream` parameter. To make it easier to integrate with these existing APIs, `PipeReader` and `PipeWriter` expose an `AsStream` method. `AsStream` returns a `Stream` implementation around the `PipeReader` or `PipeWriter`.

Stream example

`PipeReader` and `PipeWriter` instances can be created using the static `Create` methods given a `Stream` object and optional corresponding creation options.

The `StreamPipeReaderOptions` allow for control over the creation of the `PipeReader` instance with the following parameters:

- `StreamPipeReaderOptions.BufferSize` is the minimum buffer size in bytes used when renting memory from the pool, and defaults to `4096`.
- `StreamPipeReaderOptions.LeaveOpen` flag determines whether or not the underlying stream is left open after the `PipeReader` completes, and defaults to `false`.
- `StreamPipeReaderOptions.MinimumReadSize` represents the threshold of remaining bytes in the buffer before a new buffer is allocated, and defaults to `1024`.
- `StreamPipeReaderOptions.Pool` is the `MemoryPool<byte>` used when allocating memory, and defaults to `null`
- .

The `StreamPipeWriterOptions` allow for control over the creation of the `PipeWriter` instance with the following parameters:

- `StreamPipeWriterOptions.LeaveOpen` flag determines whether or not the underlying stream is left open after the `PipeWriter` completes, and defaults to `false`.
- `StreamPipeWriterOptions.MinimumBufferSize` represents the minimum buffer size to use when renting memory from the `Pool`, and defaults to `4096`.
- `StreamPipeWriterOptions.Pool` is the `MemoryPool<byte>` used when allocating memory, and defaults to `null`.

IMPORTANT

When creating `PipeReader` and `PipeWriter` instances using the `Create` methods, you need to consider the `Stream` object lifetime. If you need access to the stream after the reader or writer is done with it, you'll need to set the `LeaveOpen` flag to `true` on the creation options. Otherwise, the stream will be closed.

The following code demonstrates the creation of `PipeReader` and `PipeWriter` instances using the `Create` methods from a stream.

```
using System.Buffers;
using System.IO.Pipelines;
using System.Text;
```

```
class Program
{
    static async Task Main()
    {
        using var stream = File.OpenRead("lorem-ipsum.txt");

        var reader = PipeReader.Create(stream);
        var writer = PipeWriter.Create(
            Console.OpenStandardOutput(),
            new StreamPipeWriterOptions(leaveOpen: true));

        WriteUserCancellationPrompt();

        var processMessagesTask = ProcessMessagesAsync(reader, writer);
        var userCanceled = false;
        var cancelProcessingTask = Task.Run(() =>
        {
            while (char.ToUpperInvariant(Console.ReadKey().KeyChar) != 'C')
            {
                WriteUserCancellationPrompt();
            }

            userCanceled = true;

            // No exceptions thrown
            reader.CancelPendingRead();
            writer.CancelPendingFlush();
        });
    }

    await Task.WhenAny(cancelProcessingTask, processMessagesTask);

    Console.WriteLine(
        $"{Environment.NewLine}Processing {(userCanceled ? "cancelled" : "completed")}{Environment.NewLine}");
}

static void WriteUserCancellationPrompt() =>
    Console.WriteLine("Press 'C' to cancel processing...{0}{1}", Environment.NewLine, Environment.NewLine);

static async Task ProcessMessagesAsync(
    PipeReader reader,
    PipeWriter writer)
{
    try
    {
        while (true)
        {
            ReadResult readResult = await reader.ReadAsync();
            ReadOnlySequence<byte> buffer = readResult.Buffer;

            try
            {
                if (readResult.IsCanceled)
                {
                    break;
                }

                if (TryParseLines(ref buffer, out string message))
                {
                    FlushResult flushResult =
                        await WriteMessagesAsync(writer, message);

                    if (flushResult.IsCanceled || flushResult.IsCompleted)
                    {
                        break;
                    }
                }
            }

            if (readResult.IsCompleted)

```

```

        {
            if (!buffer.IsEmpty)
            {
                throw new InvalidDataException("Incomplete message.");
            }
            break;
        }
    }
    finally
    {
        reader.AdvanceTo(buffer.Start, buffer.End);
    }
}
}
catch (Exception ex)
{
    Console.Error.WriteLine(ex);
}
finally
{
    await reader.CompleteAsync();
    await writer.CompleteAsync();
}
}
}

static bool TryParseLines(
    ref ReadOnlySequence<byte> buffer,
    out string message)
{
    SequencePosition? position;
    StringBuilder outputMessage = new();

    while(true)
    {
        position = buffer.PositionOf((byte)'\n');

        if (!position.HasValue)
            break;

        outputMessage.Append(Encoding.ASCII.GetString(buffer.Slice(buffer.Start, position.Value)))
                    .AppendLine();

        buffer = buffer.Slice(buffer.GetPosition(1, position.Value));
    };

    message = outputMessage.ToString();
    return message.Length != 0;
}

static ValueTask<FlushResult> WriteMessagesAsync(
    PipeWriter writer,
    string message) =>
    writer.WriteAsync(Encoding.ASCII.GetBytes(message));
}
}

```

The application uses a [StreamReader](#) to read the *lorem-ipsum.txt* file as a stream, and it must end with a blank line. The [FileStream](#) is passed to [PipeReader.Create](#), which instantiates a [PipeReader](#) object. The console application then passes its standard output stream to [PipeWriter.Create](#) using [Console.OpenStandardOutput\(\)](#). The example supports [cancellation](#).

Work with Buffers in .NET

9/20/2022 • 9 minutes to read • [Edit Online](#)

This article provides an overview of types that help read data that runs across multiple buffers. They're primarily used to support [PipeReader](#) objects.

IBufferWriter<T>

`System.Buffers.IBufferWriter<T>` is a contract for synchronous buffered writing. At the lowest level, the interface:

- Is basic and not difficult to use.
- Allows access to a `Memory<T>` or `Span<T>`. The `Memory<T>` or `Span<T>` can be written to and you can determine how many `T` items were written.

```
void WriteHello(IBufferWriter<byte> writer)
{
    // Request at least 5 bytes.
    Span<byte> span = writer.GetSpan(5);
    ReadOnlySpan<char> helloSpan = "Hello".AsSpan();
    int written = Encoding.ASCII.GetBytes(helloSpan, span);

    // Tell the writer how many bytes were written.
    writer.Advance(written);
}
```

The preceding method:

- Requests a buffer of at least 5 bytes from the `IBufferWriter<byte>` using `GetSpan(5)`.
- Writes bytes for the ASCII string "Hello" to the returned `Span<byte>`.
- Calls `IBufferWriter<T>` to indicate how many bytes were written to the buffer.

This method of writing uses the `Memory<T>` / `Span<T>` buffer provided by the `IBufferWriter<T>`. Alternatively, the `Write` extension method can be used to copy an existing buffer to the `IBufferWriter<T>`. `Write` does the work of calling `GetSpan` / `Advance` as appropriate, so there's no need to call `Advance` after writing:

```
void WriteHello(IBufferWriter<byte> writer)
{
    byte[] helloBytes = Encoding.ASCII.GetBytes("Hello");

    // Write helloBytes to the writer. There's no need to call Advance here
    // since Write calls Advance.
    writer.Write(helloBytes);
}
```

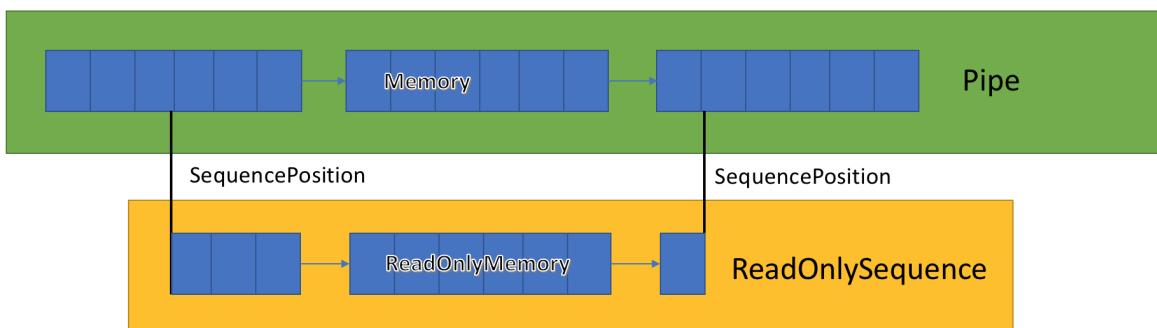
`ArrayBufferWriter<T>` is an implementation of `IBufferWriter<T>` whose backing store is a single contiguous array.

IBufferWriter common problems

- `GetSpan` and `GetMemory` return a buffer with at least the requested amount of memory. Don't assume exact buffer sizes.
- There's no guarantee that successive calls will return the same buffer or the same-sized buffer.
- A new buffer must be requested after calling `Advance` to continue writing more data. A previously acquired

buffer cannot be written to after `Advance` has been called.

ReadOnlySequence<T>



`ReadOnlySequence<T>` is a struct that can represent a contiguous or noncontiguous sequence of `T`. It can be constructed from:

1. A `T[]`
2. A `ReadOnlyMemory<T>`
3. A pair of linked list node `ReadOnlySequenceSegment<T>` and index to represent the start and end position of the sequence.

The third representation is the most interesting one as it has performance implications on various operations on the `ReadOnlySequence<T>`:

REPRESENTATION	OPERATION	COMPLEXITY
<code>T[] / ReadOnlyMemory<T></code>	<code>Length</code>	<code>O(1)</code>
<code>T[] / ReadOnlyMemory<T></code>	<code>GetPosition(long)</code>	<code>O(1)</code>
<code>T[] / ReadOnlyMemory<T></code>	<code>Slice(int, int)</code>	<code>O(1)</code>
<code>T[] / ReadOnlyMemory<T></code>	<code>Slice(SequencePosition, SequencePosition)</code>	<code>O(1)</code>
<code>ReadOnlySequenceSegment<T></code>	<code>Length</code>	<code>O(1)</code>
<code>ReadOnlySequenceSegment<T></code>	<code>GetPosition(long)</code>	<code>O(number of segments)</code>
<code>ReadOnlySequenceSegment<T></code>	<code>Slice(int, int)</code>	<code>O(number of segments)</code>
<code>ReadOnlySequenceSegment<T></code>	<code>Slice(SequencePosition, SequencePosition)</code>	<code>O(1)</code>

Because of this mixed representation, the `ReadOnlySequence<T>` exposes indexes as `SequencePosition` instead of an integer. A `SequencePosition`:

- Is an opaque value that represents an index into the `ReadOnlySequence<T>` where it originated.
- Consists of two parts, an integer and an object. What these two values represent are tied to the implementation of `ReadOnlySequence<T>`.

Access data

The `ReadOnlySequence<T>` exposes data as an enumerable of `ReadOnlyMemory<T>`. Enumerating each of the segments can be done using a basic foreach:

```
long FindIndexOf(in ReadOnlySequence<byte> buffer, byte data)
{
    long position = 0;

    foreach (ReadOnlyMemory<byte> segment in buffer)
    {
        ReadOnlySpan<byte> span = segment.Span;
        var index = span.IndexOf(data);
        if (index != -1)
        {
            return position + index;
        }

        position += span.Length;
    }

    return -1;
}
```

The preceding method searches each segment for a specific byte. If you need to keep track of each segment's `SequencePosition`, `ReadOnlySequence<T>.TryGet` is more appropriate. The next sample changes the preceding code to return a `SequencePosition` instead of an integer. Returning a `SequencePosition` has the benefit of allowing the caller to avoid a second scan to get the data at a specific index.

```
SequencePosition? FindIndexOf(in ReadOnlySequence<byte> buffer, byte data)
{
    SequencePosition position = buffer.Start;

    while (buffer.TryGet(ref position, out ReadOnlyMemory<byte> segment))
    {
        ReadOnlySpan<byte> span = segment.Span;
        var index = span.IndexOf(data);
        if (index != -1)
        {
            return buffer.GetPosition(position, index);
        }
    }
    return null;
}
```

The combination of `SequencePosition` and `TryGet` acts like an enumerator. The position field is modified at the start of each iteration to be start of each segment within the `ReadOnlySequence<T>`.

The preceding method exists as an extension method on `ReadOnlySequence<T>`. `PositionOf` can be used to simplify the preceding code:

```
SequencePosition? FindIndexOf(in ReadOnlySequence<byte> buffer, byte data) => buffer.PositionOf(data);
```

Process a `ReadOnlySequence<T>`

Processing a `ReadOnlySequence<T>` can be challenging since data may be split across multiple segments within the sequence. For the best performance, split code into two paths:

- A fast path that deals with the single segment case.
- A slow path that deals with the data split across segments.

There are a few approaches that can be used to process data in multi-segmented sequences:

- Use the `SequenceReader<T>`.
- Parse data segment by segment, keeping track of the `SequencePosition` and index within the segment parsed. This avoids unnecessary allocations but may be inefficient, especially for small buffers.
- Copy the `ReadOnlySequence<T>` to a contiguous array and treat it like a single buffer:
 - If the size of the `ReadOnlySequence<T>` is small, it may be reasonable to copy the data into a stack-allocated buffer using the `stackalloc` operator.
 - Copy the `ReadOnlySequence<T>` into a pooled array using `ArrayPool<T>.Shared`.
 - Use `ReadOnlySequence<T>.ToArray()`. This isn't recommended in hot paths as it allocates a new `T[]` on the heap.

The following examples demonstrate some common cases for processing `ReadOnlySequence<byte>`:

Process binary data

The following example parses a 4-byte big-endian integer length from the start of the `ReadOnlySequence<byte>`.

```
bool TryParseHeaderLength(ref ReadOnlySequence<byte> buffer, out int length)
{
    // If there's not enough space, the length can't be obtained.
    if (buffer.Length < 4)
    {
        length = 0;
        return false;
    }

    // Grab the first 4 bytes of the buffer.
    var lengthSlice = buffer.Slice(buffer.Start, 4);
    if (lengthSlice.IsSingleSegment)
    {
        // Fast path since it's a single segment.
        length = BinaryPrimitives.ReadInt32BigEndian(lengthSlice.First.Span);
    }
    else
    {
        // There are 4 bytes split across multiple segments. Since it's so small, it
        // can be copied to a stack allocated buffer. This avoids a heap allocation.
        Span<byte> stackBuffer = stackalloc byte[4];
        lengthSlice.CopyTo(stackBuffer);
        length = BinaryPrimitives.ReadInt32BigEndian(stackBuffer);
    }

    // Move the buffer 4 bytes ahead.
    buffer = buffer.Slice(lengthSlice.End);

    return true;
}
```

If you would like to see code comments translated to languages other than English, let us know in [this GitHub discussion issue](#).

Process text data

The following example:

- Finds the first newline (`\r\n`) in the `ReadOnlySequence<byte>` and returns it via the out 'line' parameter.
- Trims that line, excluding the `\r\n` from the input buffer.

```

static bool TryParseLine(ref ReadOnlySequence<byte> buffer, out ReadOnlySequence<byte> line)
{
    SequencePosition position = buffer.Start;
    SequencePosition previous = position;
    var index = -1;
    line = default;

    while (buffer.TryGet(ref position, out ReadOnlyMemory<byte> segment))
    {
        ReadOnlySpan<byte> span = segment.Span;

        // Look for \r in the current segment.
        index = span.IndexOf((byte)'\r');

        if (index != -1)
        {
            // Check next segment for \n.
            if (index + 1 >= span.Length)
            {
                var next = position;
                if (!buffer.TryGet(ref next, out ReadOnlyMemory<byte> nextSegment))
                {
                    // You're at the end of the sequence.
                    return false;
                }
                else if (nextSegment.Span[0] == (byte)'\n')
                {
                    // A match was found.
                    break;
                }
            }
            // Check the current segment of \n.
            else if (span[index + 1] == (byte)'\n')
            {
                // It was found.
                break;
            }
        }

        previous = position;
    }

    if (index != -1)
    {
        // Get the position just before the \r\n.
        var delimiter = buffer.GetPosition(index, previous);

        // Slice the line (excluding \r\n).
        line = buffer.Slice(buffer.Start, delimiter);

        // Slice the buffer to get the remaining data after the line.
        buffer = buffer.Slice(buffer.GetPosition(2, delimiter));
        return true;
    }

    return false;
}

```

Empty segments

It's valid to store empty segments inside of a `ReadOnlySequence<T>`. Empty segments may occur while enumerating segments explicitly:

```

static void EmptySegments()
{
    // This logic creates a ReadOnlySequence<byte> with 4 segments,
    // two of which are empty.
    var first = new BufferSegment(new byte[0]);
    var last = first.Append(new byte[] { 97 })
        .Append(new byte[0]).Append(new byte[] { 98 });

    // Construct the ReadOnlySequence<byte> from the linked list segments.
    var data = new ReadOnlySequence<byte>(first, 0, last, 1);

    // Slice using numbers.
    var sequence1 = data.Slice(0, 2);

    // Slice using SequencePosition pointing at the empty segment.
    var sequence2 = data.Slice(data.Start, 2);

    Console.WriteLine($"sequence1.Length={sequence1.Length}"); // sequence1.Length=2
    Console.WriteLine($"sequence2.Length={sequence2.Length}"); // sequence2.Length=2

    // sequence1.FirstSpan.Length=1
    Console.WriteLine($"sequence1.FirstSpan.Length={sequence1.FirstSpan.Length}");

    // Slicing using SequencePosition will Slice the ReadOnlySequence<byte> directly
    // on the empty segment!
    // sequence2.FirstSpan.Length=0
    Console.WriteLine($"sequence2.FirstSpan.Length={sequence2.FirstSpan.Length}");

    // The following code prints 0, 1, 0, 1.
    SequencePosition position = data.Start;
    while (data.TryGet(ref position, out ReadOnlyMemory<byte> memory))
    {
        Console.WriteLine(memory.Length);
    }
}

class BufferSegment : ReadOnlySequenceSegment<byte>
{
    public BufferSegment(Memory<byte> memory)
    {
        Memory = memory;
    }

    public BufferSegment Append(Memory<byte> memory)
    {
        var segment = new BufferSegment(memory)
        {
            RunningIndex = RunningIndex + Memory.Length
        };
        Next = segment;
        return segment;
    }
}

```

The preceding code creates a `ReadOnlySequence<byte>` with empty segments and shows how those empty segments affect the various APIs:

- `ReadOnlySequence<T>.Slice` with a `SequencePosition` pointing to an empty segment preserves that segment.
- `ReadOnlySequence<T>.Slice` with an int skips over the empty segments.
- Enumerating the `ReadOnlySequence<T>` enumerates the empty segments.

Potential problems with `ReadOnlySequence<T>` and `SequencePosition`

There are several unusual outcomes when dealing with a `ReadOnlySequence<T>` / `SequencePosition` vs. a normal `ReadOnlySpan<T>` / `ReadOnlyMemory<T>` / `T[]` / `int`:

- `SequencePosition` is a position marker for a specific `ReadOnlySequence<T>`, not an absolute position. Because it's relative to a specific `ReadOnlySequence<T>`, it doesn't have meaning if used outside of the `ReadOnlySequence<T>` where it originated.
- Arithmetic can't be performed on `SequencePosition` without the `ReadOnlySequence<T>`. That means doing basic things like `position++` is written `position = ReadOnlySequence<T>.GetPosition(1, position)`.
- `GetPosition(long)` does **not** support negative indexes. That means it's impossible to get the second to last character without walking all segments.
- Two `SequencePosition` can't be compared, making it difficult to:
 - Know if one position is greater than or less than another position.
 - Write some parsing algorithms.
- `ReadOnlySequence<T>` is bigger than an object reference and should be passed by `in` or `ref` where possible. Passing `ReadOnlySequence<T>` by `in` or `ref` reduces copies of the `struct`.
- Empty segments:
 - Are valid within a `ReadOnlySequence<T>`.
 - Can appear when iterating using the `ReadOnlySequence<T>.TryGet` method.
 - Can appear slicing the sequence using the `ReadOnlySequence<T>.Slice()` method with `SequencePosition` objects.

SequenceReader<T>

SequenceReader<T>:

- Is a new type that was introduced in .NET Core 3.0 to simplify the processing of a `ReadOnlySequence<T>`.
- Unifies the differences between a single segment `ReadOnlySequence<T>` and multi-segment `ReadOnlySequence<T>`.
- Provides helpers for reading binary and text data (`byte` and `char`) that may or may not be split across segments.

There are built-in methods for dealing with processing both binary and delimited data. The following section demonstrates what those same methods look like with the `SequenceReader<T>`:

Access data

`SequenceReader<T>` has methods for enumerating data inside of the `ReadOnlySequence<T>` directly. The following code is an example of processing a `ReadOnlySequence<byte>` a `byte` at a time:

```
while (reader.TryRead(out byte b))
{
    Process(b);
}
```

The `currentSpan` exposes the current segment's `Span`, which is similar to what was done in the method manually.

Use position

The following code is an example implementation of `FindIndexOf` using the `SequenceReader<T>`:

```

SequencePosition? FindIndexOf(in ReadOnlySequence<byte> buffer, byte data)
{
    var reader = new SequenceReader<byte>(buffer);

    while (!reader.End)
    {
        // Search for the byte in the current span.
        var index = reader.CurrentSpan.IndexOf(data);
        if (index != -1)
        {
            // It was found, so advance to the position.
            reader.Advance(index);

            return reader.Position;
        }
        // Skip the current segment since there's nothing in it.
        reader.Advance(reader.CurrentSpan.Length);
    }

    return null;
}

```

Process binary data

The following example parses a 4-byte big-endian integer length from the start of the `ReadOnlySequence<byte>`.

```

bool TryParseHeaderLength(ref ReadOnlySequence<byte> buffer, out int length)
{
    var reader = new SequenceReader<byte>(buffer);
    return reader.TryReadBigEndian(out length);
}

```

Process text data

```

static ReadOnlySpan<byte> NewLine => new byte[] { (byte)'\\r', (byte)'\\n' };

static bool TryParseLine(ref ReadOnlySequence<byte> buffer,
                        out ReadOnlySequence<byte> line)
{
    var reader = new SequenceReader<byte>(buffer);

    if (reader.TryReadTo(out line, NewLine))
    {
        buffer = buffer.Slice(reader.Position);

        return true;
    }

    line = default;
    return false;
}

```

SequenceReader<T> common problems

- Because `SequenceReader<T>` is a mutable struct, it should always be passed by [reference](#).
- `SequenceReader<T>` is a [ref struct](#) so it can only be used in synchronous methods and can't be stored in fields. For more information, see [Write safe and efficient C# code](#).
- `SequenceReader<T>` is optimized for use as a forward-only reader. `Rewind` is intended for small backups that can't be addressed utilizing other `Read`, `Peek`, and `IsNext` APIs.

Memory-mapped files

9/20/2022 • 10 minutes to read • [Edit Online](#)

A memory-mapped file contains the contents of a file in virtual memory. This mapping between a file and memory space enables an application, including multiple processes, to modify the file by reading and writing directly to the memory. You can use managed code to access memory-mapped files in the same way that native Windows functions access memory-mapped files, as described in [Managing Memory-Mapped Files](#).

There are two types of memory-mapped files:

- Persisted memory-mapped files

Persisted files are memory-mapped files that are associated with a source file on a disk. When the last process has finished working with the file, the data is saved to the source file on the disk. These memory-mapped files are suitable for working with extremely large source files.

- Non-persisted memory-mapped files

Non-persisted files are memory-mapped files that are not associated with a file on a disk. When the last process has finished working with the file, the data is lost and the file is reclaimed by garbage collection. These files are suitable for creating shared memory for inter-process communications (IPC).

Processes, Views, and Managing Memory

Memory-mapped files can be shared across multiple processes. Processes can map to the same memory-mapped file by using a common name that is assigned by the process that created the file.

To work with a memory-mapped file, you must create a view of the entire memory-mapped file or a part of it. You can also create multiple views to the same part of the memory-mapped file, thereby creating concurrent memory. For two views to remain concurrent, they have to be created from the same memory-mapped file.

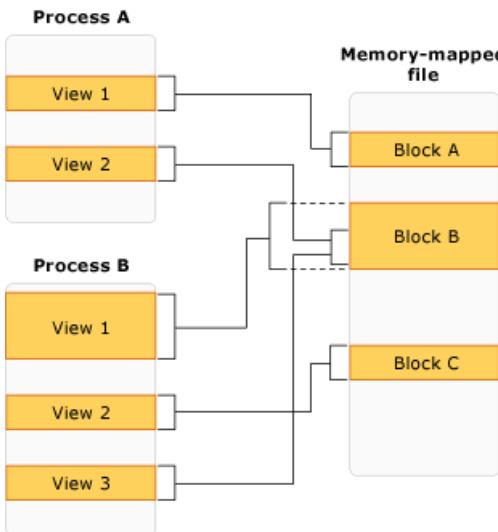
Multiple views may also be necessary if the file is greater than the size of the application's logical memory space available for memory mapping (2 GB on a 32-bit computer).

There are two types of views: stream access view and random access view. Use stream access views for sequential access to a file; this is recommended for non-persisted files and IPC. Random access views are preferred for working with persisted files.

Memory-mapped files are accessed through the operating system's memory manager, so the file is automatically partitioned into a number of pages and accessed as needed. You do not have to handle the memory management yourself.

The following illustration shows how multiple processes can have multiple and overlapping views to the same memory-mapped file at the same time.

The following image shows multiple and overlapped views to a memory-mapped file:



Programming with Memory-Mapped Files

The following table provides a guide for using memory-mapped file objects and their members.

TASK	METHODS OR PROPERTIES TO USE
To obtain a MemoryMappedFile object that represents a persisted memory-mapped file from a file on disk.	MemoryMappedFile.CreateFromFile method.
To obtain a MemoryMappedFile object that represents a non-persisted memory-mapped file (not associated with a file on disk).	MemoryMappedFile.CreateNew method. - or - MemoryMappedFile.CreateOrOpen method.
To obtain a MemoryMappedFile object of an existing memory-mapped file (either persisted or non-persisted).	MemoryMappedFile.OpenExisting method.
To obtain a UnmanagedMemoryStream object for a sequentially accessed view to the memory-mapped file.	MemoryMappedFile.CreateViewStream method.
To obtain a UnmanagedMemoryAccessor object for a random access view to a memory-mapped file.	MemoryMappedFile.CreateViewAccessor method.
To obtain a SafeMemoryMappedViewHandle object to use with unmanaged code.	MemoryMappedFile.SafeMemoryMappedFileHandle property. - or - MemoryMappedViewAccessor.SafeMemoryMappedViewHandle property. - or - MemoryMapViewStream.SafeMemoryMappedViewHandle property.

TASK	METHODS OR PROPERTIES TO USE
To delay allocating memory until a view is created (non-persisted files only). (To determine the current system page size, use the Environment.SystemPageSize property.)	CreateNew method with the MemoryMappedFileOptions.DelayAllocatePages value. - or - CreateOrOpen methods that have a MemoryMappedFileOptions enumeration as a parameter.

Security

You can apply access rights when you create a memory-mapped file, by using the following methods that take a [MemoryMappedFileAccess](#) enumeration as a parameter:

- [MemoryMappedFile.CreateFromFile](#)
- [MemoryMappedFile.CreateNew](#)
- [MemoryMappedFile.CreateOrOpen](#)

You can specify access rights for opening an existing memory-mapped file by using the [OpenExisting](#) methods that take an [MemoryMappedFileRights](#) as a parameter.

In addition, you can include a [MemoryMappedFileSecurity](#) object that contains predefined access rules.

To apply new or changed access rules to a memory-mapped file, use the [SetAccessControl](#) method. To retrieve access or audit rules from an existing file, use the [GetAccessControl](#) method.

Examples

Persisted Memory-Mapped Files

The [CreateFromFile](#) methods create a memory-mapped file from an existing file on disk.

The following example creates a memory-mapped view of a part of an extremely large file and manipulates a portion of it.

```

using System;
using System.IO;
using System.IO.MemoryMappedFiles;
using System.Runtime.InteropServices;

class Program
{
    static void Main(string[] args)
    {
        long offset = 0x10000000; // 256 megabytes
        long length = 0x20000000; // 512 megabytes

        // Create the memory-mapped file.
        using (var mmf = MemoryMappedFile.CreateFromFile(@"c:\ExtremelyLargeImage.data",
FileMode.Open,"ImgA"))
        {
            // Create a random access view, from the 256th megabyte (the offset)
            // to the 768th megabyte (the offset plus length).
            using (var accessor = mmf.CreateViewAccessor(offset, length))
            {
                int colorSize = Marshal.SizeOf(typeof(MyColor));
                MyColor color;

                // Make changes to the view.
                for (long i = 0; i < length; i += colorSize)
                {
                    accessor.Read(i, out color);
                    color.Brighten(10);
                    accessor.Write(i, ref color);
                }
            }
        }
    }

    public struct MyColor
    {
        public short Red;
        public short Green;
        public short Blue;
        public short Alpha;

        // Make the view brighter.
        public void Brighten(short value)
        {
            Red = (short)Math.Min(short.MaxValue, (int)Red + value);
            Green = (short)Math.Min(short.MaxValue, (int)Green + value);
            Blue = (short)Math.Min(short.MaxValue, (int)Blue + value);
            Alpha = (short)Math.Min(short.MaxValue, (int)Alpha + value);
        }
    }
}

```

```

Imports System.IO
Imports System.IO.MemoryMappedFiles
Imports System.Runtime.InteropServices

Class Program

    Sub Main()
        Dim offset As Long = &H10000000 ' 256 megabytes
        Dim length As Long = &H20000000 ' 512 megabytes

        ' Create the memory-mapped file.
        Using mmf = MemoryMappedFile.CreateFromFile("c:\ExtremelyLargeImage.data", FileMode.Open, "ImgA")
            ' Create a random access view, from the 256th megabyte (the offset)
            ' to the 768th megabyte (the offset plus length).
            Using accessor = mmf.CreateViewAccessor(offset, length)
                Dim colorSize As Integer = Marshal.SizeOf(GetType(MyColor))
                Dim color As MyColor
                Dim i As Long = 0

                ' Make changes to the view.
                Do While (i < length)
                    accessor.Read(i, color)
                    color.Brighten(10)
                    accessor.Write(i, color)
                    i += colorSize
                Loop
            End Using
        End Using
    End Sub
End Class

Public Structure MyColor
    Public Red As Short
    Public Green As Short
    Public Blue As Short
    Public Alpha As Short

    ' Make the view brighter.
    Public Sub Brighten(ByVal value As Short)
        Red = CType(Math.Min(Short.MaxValue, (CType(Red, Integer) + value)), Short)
        Green = CType(Math.Min(Short.MaxValue, (CType(Green, Integer) + value)), Short)
        Blue = CType(Math.Min(Short.MaxValue, (CType(Blue, Integer) + value)), Short)
        Alpha = CType(Math.Min(Short.MaxValue, (CType(Alpha, Integer) + value)), Short)
    End Sub
End Structure

```

If you would like to see code comments translated to languages other than English, let us know in [this GitHub discussion issue](#).

The following example opens the same memory-mapped file for another process.

```
using System;
using System.IO.MemoryMappedFiles;
using System.Runtime.InteropServices;

class Program
{
    static void Main(string[] args)
    {
        // Assumes another process has created the memory-mapped file.
        using (var mmf = MemoryMappedFile.OpenExisting("ImgA"))
        {
            using (var accessor = mmf.CreateViewAccessor(4000000, 2000000))
            {
                int colorSize = Marshal.SizeOf(typeof(MyColor));
                MyColor color;

                // Make changes to the view.
                for (long i = 0; i < 1500000; i += colorSize)
                {
                    accessor.Read(i, out color);
                    color.Brighten(20);
                    accessor.Write(i, ref color);
                }
            }
        }
    }

    public struct MyColor
    {
        public short Red;
        public short Green;
        public short Blue;
        public short Alpha;

        // Make the view brighter.
        public void Brighten(short value)
        {
            Red = (short)Math.Min(short.MaxValue, (int)Red + value);
            Green = (short)Math.Min(short.MaxValue, (int)Green + value);
            Blue = (short)Math.Min(short.MaxValue, (int)Blue + value);
            Alpha = (short)Math.Min(short.MaxValue, (int)Alpha + value);
        }
    }
}
```

```

Imports System.IO.MemoryMappedFiles
Imports System.Runtime.InteropServices

Class Program
    Public Shared Sub Main(ByVal args As String())
        ' Assumes another process has created the memory-mapped file.
        Using mmf = MemoryMappedFile.OpenExisting("ImgA")
            Using accessor = mmf.CreateViewAccessor(4000000, 2000000)
                Dim colorSize As Integer = Marshal.SizeOf(GetType(MyColor))
                Dim color As MyColor

                ' Make changes to the view.
                Dim i As Long = 0
                While i < 1500000
                    accessor.Read(i, color)
                    color.Brighten(30)
                    accessor.Write(i, color)
                    i += colorSize
                End While
            End Using
        End Using
    End Sub
End Class

Public Structure MyColor
    Public Red As Short
    Public Green As Short
    Public Blue As Short
    Public Alpha As Short

    ' Make the view brighter.
    Public Sub Brighten(ByVal value As Short)
        Red = CShort(Math.Min(Short.MaxValue, CInt(Red) + value))
        Green = CShort(Math.Min(Short.MaxValue, CInt(Green) + value))
        Blue = CShort(Math.Min(Short.MaxValue, CInt(Blue) + value))
        Alpha = CShort(Math.Min(Short.MaxValue, CInt(Alpha) + value))
    End Sub
End Structure

```

Non-Persisted Memory-Mapped Files

The [CreateNew](#) and [CreateOrOpen](#) methods create a memory-mapped file that is not mapped to an existing file on disk.

The following example consists of three separate processes (console applications) that write Boolean values to a memory-mapped file. The following sequence of actions occur:

1. **Process A** creates the memory-mapped file and writes a value to it.
2. **Process B** opens the memory-mapped file and writes a value to it.
3. **Process C** opens the memory-mapped file and writes a value to it.
4. **Process A** reads and displays the values from the memory-mapped file.
5. After **Process A** is finished with the memory-mapped file, the file is immediately reclaimed by garbage collection.

To run this example, do the following:

1. Compile the applications and open three Command Prompt windows.
2. In the first Command Prompt window, run **Process A**.
3. In the second Command Prompt window, run **Process B**.

4. Return to **Process A** and press ENTER.
5. In the third Command Prompt window, run **Process C**.
6. Return to **Process A** and press ENTER.

The output of **Process A** is as follows:

```
Start Process B and press ENTER to continue.
Start Process C and press ENTER to continue.
Process A says: True
Process B says: False
Process C says: True
```

Process A

```
using System;
using System.IO;
using System.IO.MemoryMappedFiles;
using System.Threading;

class Program
{
    // Process A:
    static void Main(string[] args)
    {
        using (MemoryMappedFile mmf = MemoryMappedFile.CreateNew("testmap", 10000))
        {
            bool mutexCreated;
            Mutex mutex = new Mutex(true, "testmapmutex", out mutexCreated);
            using (MemoryMappedViewStream stream = mmf.CreateViewStream())
            {
                BinaryWriter writer = new BinaryWriter(stream);
                writer.Write(1);
            }
            mutex.ReleaseMutex();

            Console.WriteLine("Start Process B and press ENTER to continue.");
            Console.ReadLine();

            Console.WriteLine("Start Process C and press ENTER to continue.");
            Console.ReadLine();

            mutex.WaitOne();
            using (MemoryMappedViewStream stream = mmf.CreateViewStream())
            {
                BinaryReader reader = new BinaryReader(stream);
                Console.WriteLine("Process A says: {0}", reader.ReadBoolean());
                Console.WriteLine("Process B says: {0}", reader.ReadBoolean());
                Console.WriteLine("Process C says: {0}", reader.ReadBoolean());
            }
            mutex.ReleaseMutex();
        }
    }
}
```

```

Imports System.IO
Imports System.IO.MemoryMappedFiles
Imports System.Threading

Module Module1

    ' Process A:
    Sub Main()
        Using mmf As MemoryMappedFile = MemoryMappedFile.CreateNew("testmap", 10000)
            Dim mutexCreated As Boolean
            Dim mTex As Mutex = New Mutex(True, "testmapmutex", mutexCreated)
            Using Stream As MemoryMappedViewStream = mmf.CreateViewStream()
                Dim writer As BinaryWriter = New BinaryWriter(Stream)
                writer.Write(1)
            End Using
            mTex.ReleaseMutex()
            Console.WriteLine("Start Process B and press ENTER to continue.")
            Console.ReadLine()

            Console.WriteLine("Start Process C and press ENTER to continue.")
            Console.ReadLine()

            mTex.WaitOne()
            Using Stream As MemoryMappedViewStream = mmf.CreateViewStream()
                Dim reader As BinaryReader = New BinaryReader(Stream)
                Console.WriteLine("Process A says: {0}", reader.ReadBoolean())
                Console.WriteLine("Process B says: {0}", reader.ReadBoolean())
                Console.WriteLine("Process C says: {0}", reader.ReadBoolean())
            End Using
            mTex.ReleaseMutex()
        End Using
    End Sub

End Module

```

Process B

```

using System;
using System.IO;
using System.IO.MemoryMappedFiles;
using System.Threading;

class Program
{
    // Process B:
    static void Main(string[] args)
    {
        try
        {
            using (MemoryMappedFile mmf = MemoryMappedFile.OpenExisting("testmap"))
            {

                Mutex mutex = Mutex.OpenExisting("testmapmutex");
                mutex.WaitOne();

                using (MemoryMappedViewStream stream = mmf.CreateViewStream(1, 0))
                {
                    BinaryWriter writer = new BinaryWriter(stream);
                    writer.Write(0);
                }
                mutex.ReleaseMutex();
            }
        }
        catch (FileNotFoundException)
        {
            Console.WriteLine("Memory-mapped file does not exist. Run Process A first.");
        }
    }
}

```

```

Imports System.IO
Imports System.IO.MemoryMappedFiles
Imports System.Threading

Module Module1
    ' Process B:
    Sub Main()
        Try
            Using mmf As MemoryMappedFile = MemoryMappedFile.OpenExisting("testmap")
                Dim mTex As Mutex = Mutex.OpenExisting("testmapmutex")
                mTex.WaitOne()
                Using Stream As MemoryMappedViewStream = mmf.CreateViewStream(1, 0)
                    Dim writer As BinaryWriter = New BinaryWriter(Stream)
                    writer.Write(0)
                End Using
                mTex.ReleaseMutex()
            End Using
        Catch noFile As FileNotFoundException
            Console.WriteLine("Memory-mapped file does not exist. Run Process A first." & vbCrLf & noFile.Message)
        End Try
    End Sub
End Module

```

Process C

```

using System;
using System.IO;
using System.IO.MemoryMappedFiles;
using System.Threading;

class Program
{
    // Process C:
    static void Main(string[] args)
    {
        try
        {
            using (MemoryMappedFile mmf = MemoryMappedFile.OpenExisting("testmap"))
            {

                Mutex mutex = Mutex.OpenExisting("testmapmutex");
                mutex.WaitOne();

                using (MemoryMappedViewStream stream = mmf.CreateViewStream(2, 0))
                {
                    BinaryWriter writer = new BinaryWriter(stream);
                    writer.Write(1);
                }
                mutex.ReleaseMutex();
            }
        }
        catch (FileNotFoundException)
        {
            Console.WriteLine("Memory-mapped file does not exist. Run Process A first, then B.");
        }
    }
}

```

```

Imports System.IO
Imports System.IO.MemoryMappedFiles
Imports System.Threading

Module Module1
    ' Process C:
    Sub Main()
        Try
            Using mmf As MemoryMappedFile = MemoryMappedFile.OpenExisting("testmap")
                Dim mTex As Mutex = Mutex.OpenExisting("testmapmutex")
                mTex.WaitOne()
                Using Stream As MemoryMappedViewStream = mmf.CreateViewStream(2, 0)
                    Dim writer As BinaryWriter = New BinaryWriter(Stream)
                    writer.Write(1)
                End Using
                mTex.ReleaseMutex()
            End Using
        Catch noFile As FileNotFoundException
            Console.WriteLine("Memory-mapped file does not exist. Run Process A first, then B." & vbCrLf & noFile.Message)
        End Try
    End Sub
End Module

```

See also

- [File and Stream I/O](#)

Console apps in .NET

9/20/2022 • 2 minutes to read • [Edit Online](#)

.NET applications can use the [System.Console](#) class to read characters from and write characters to the console. Data from the console is read from the standard input stream, data to the console is written to the standard output stream, and error data to the console is written to the standard error output stream. These streams are automatically associated with the console when the application starts and are presented as the [In](#), [Out](#), and [Error](#) properties, respectively.

The value of the [Console.In](#) property is a [System.IO.TextReader](#) object, whereas the values of the [Console.Out](#) and [Console.Error](#) properties are [System.IO.TextWriter](#) objects. You can associate these properties with streams that do not represent the console, making it possible for you to point the stream to a different location for input or output. For example, you can redirect the output to a file by setting the [Console.Out](#) property to a [System.IO.StreamWriter](#), which encapsulates a [System.IO.FileStream](#) by means of the [Console.SetOut](#) method. The [Console.In](#) and [Console.Out](#) properties do not need to refer to the same stream.

NOTE

For more information about building console applications, including examples in C#, Visual Basic, and C++, see the documentation for the [Console](#) class.

If the console does not exist, for example, in a Windows Forms application, output written to the standard output stream will not be visible, because there is no console to write the information to. Writing information to an inaccessible console does not cause an exception to be raised. (You can always change the application type to [Console Application](#), for example, in the project property pages in Visual Studio).

The [System.Console](#) class has methods that can read individual characters or entire lines from the console. Other methods convert data and format strings, and then write the formatted strings to the console. For more information on formatting strings, see [Formatting types](#).

TIP

Console applications lack a message pump that starts by default. Therefore, console calls to Microsoft Win32 timers might fail.

See also

- [System.Console](#)
- [Formatting Types](#)

Dependency injection in .NET

9/20/2022 • 14 minutes to read • [Edit Online](#)

.NET supports the dependency injection (DI) software design pattern, which is a technique for achieving [Inversion of Control \(IoC\)](#) between classes and their dependencies. Dependency injection in .NET is a built-in part of the framework, along with configuration, logging, and the options pattern.

A *dependency* is an object that another object depends on. Examine the following `MessageWriter` class with a `Write` method that other classes depend on:

```
public class MessageWriter
{
    public void Write(string message)
    {
        Console.WriteLine($"MessageWriter.Write(message: \"{message}\")");
    }
}
```

A class can create an instance of the `MessageWriter` class to make use of its `Write` method. In the following example, the `MessageWriter` class is a dependency of the `Worker` class:

```
public class Worker : BackgroundService
{
    private readonly MessageWriter _messageWriter = new MessageWriter();

    protected override async Task ExecuteAsync(CancellationToken stoppingToken)
    {
        while (!stoppingToken.IsCancellationRequested)
        {
            _messageWriter.Write($"Worker running at: {DateTimeOffset.Now}");
            await Task.Delay(1000, stoppingToken);
        }
    }
}
```

The class creates and directly depends on the `MessageWriter` class. Hard-coded dependencies, such as in the previous example, are problematic and should be avoided for the following reasons:

- To replace `MessageWriter` with a different implementation, the `Worker` class must be modified.
- If `MessageWriter` has dependencies, they must also be configured by the `Worker` class. In a large project with multiple classes depending on `MessageWriter`, the configuration code becomes scattered across the app.
- This implementation is difficult to unit test. The app should use a mock or stub `MessageWriter` class, which isn't possible with this approach.

Dependency injection addresses these problems through:

- The use of an interface or base class to abstract the dependency implementation.
- Registration of the dependency in a service container. .NET provides a built-in service container, `IServiceProvider`. Services are typically registered at the app's start-up and appended to an `IServiceCollection`. Once all services are added, you use `BuildServiceProvider` to create the service container.
- *Injection* of the service into the constructor of the class where it's used. The framework takes on the responsibility of creating an instance of the dependency and disposing of it when it's no longer needed.

As an example, the `IMessageWriter` interface defines the `Write` method:

```
namespace DependencyInjection.Example;

public interface IMessageWriter
{
    void Write(string message);
}
```

This interface is implemented by a concrete type, `MessageWriter`:

```
namespace DependencyInjection.Example;

public class MessageWriter : IMessageWriter
{
    public void Write(string message)
    {
        Console.WriteLine($"MessageWriter.Write(message: \"{message}\")");
    }
}
```

The sample code registers the `IMessageWriter` service with the concrete type `MessageWriter`. The [AddScoped](#) method registers the service with a scoped lifetime, the lifetime of a single request. [Service lifetimes](#) are described later in this article.

```
using DependencyInjection.Example;

var builder = Host.CreateDefaultBuilder(args);

builder.ConfigureServices(
    services =>
        services.AddHostedService<Worker>()
            .AddScoped<IMessageWriter, MessageWriter>());

var host = builder.Build();

host.Run();
```

In the preceding code, the sample app:

- Creates a host builder instance.
- Configures the services by registering:
 - The `Worker` as a hosted service. For more information, see [Worker Services in .NET](#).
 - The `IMessageWriter` interface as a scoped service with a corresponding implementation of the `MessageWriter` class.
- Builds the host and runs it.

The host contains the dependency injection service provider. It also contains all the other relevant services required to automatically instantiate the `Worker` and provide the corresponding `IMessageWriter` implementation as an argument.

```

namespace DependencyInjection.Example;

public class Worker : BackgroundService
{
    private readonly IMessageWriter _messageWriter;

    public Worker(IMessageWriter messageWriter) =>
        _messageWriter = messageWriter;

    protected override async Task ExecuteAsync(CancellationToken stoppingToken)
    {
        while (!stoppingToken.IsCancellationRequested)
        {
            _messageWriter.Write($"Worker running at: {DateTimeOffset.Now}");
            await Task.Delay(1000, stoppingToken);
        }
    }
}

```

By using the DI pattern, the worker service:

- Doesn't use the concrete type `MessageWriter`, only the `IMessageWriter` interface that implements it. That makes it easy to change the implementation that the worker service uses without modifying the worker service.
- Doesn't create an instance of `MessageWriter`. The instance is created by the DI container.

The implementation of the `IMessageWriter` interface can be improved by using the built-in logging API:

```

namespace DependencyInjection.Example;

public class LoggingMessageWriter : IMessageWriter
{
    private readonly ILogger<LoggingMessageWriter> _logger;

    public LoggingMessageWriter(ILogger<LoggingMessageWriter> logger) =>
        _logger = logger;

    public void Write(string message) =>
        _logger.LogInformation(message);
}

```

The updated `ConfigureServices` method registers the new `IMessageWriter` implementation:

```

static IHostBuilder CreateHostBuilder(string[] args) =>
    Host.CreateDefaultBuilder(args)
        .ConfigureServices((_, services) =>
            services.AddHostedService<Worker>()
                .AddScoped<IMessageWriter, LoggingMessageWriter>();

```

`LoggingMessageWriter` depends on `ILogger<TCategoriesName>`, which it requests in the constructor.

`ILogger<TCategoriesName>` is a [framework-provided service](#).

It's not unusual to use dependency injection in a chained fashion. Each requested dependency in turn requests its own dependencies. The container resolves the dependencies in the graph and returns the fully resolved service. The collective set of dependencies that must be resolved is typically referred to as a *dependency tree*, *dependency graph*, or *object graph*.

The container resolves `ILogger<TCategoriesName>` by taking advantage of [\(generic\) open types](#), eliminating the need to register every [\(generic\) constructed type](#).

With dependency injection terminology, a service:

- Is typically an object that provides a service to other objects, such as the `IMessageWriter` service.
- Is not related to a web service, although the service may use a web service.

The framework provides a robust logging system. The `IMessageWriter` implementations shown in the preceding examples were written to demonstrate basic DI, not to implement logging. Most apps shouldn't need to write loggers. The following code demonstrates using the default logging, which only requires the `Worker` to be registered in `ConfigureServices` as a hosted service [AddHostedService](#):

```
public class Worker : BackgroundService
{
    private readonly ILogger<Worker> _logger;

    public Worker(ILogger<Worker> logger) =>
        _logger = logger;

    protected override async Task ExecuteAsync(CancellationToken stoppingToken)
    {
        while (!stoppingToken.IsCancellationRequested)
        {
            _logger.LogInformation("Worker running at: {time}", DateTimeOffset.Now);
            await Task.Delay(1000, stoppingToken);
        }
    }
}
```

Using the preceding code, there is no need to update `ConfigureServices`, because logging is provided by the framework.

Multiple constructor discovery rules

When a type defines more than one constructor, the service provider has logic for determining which constructor to use. The constructor with the most parameters where the types are DI-resolvable is selected. Consider the following C# example service:

```
public class ExampleService
{
    public ExampleService()
    {}

    public ExampleService(ILogger<ExampleService> logger)
    {
        // omitted for brevity
    }

    public ExampleService(FooService fooService, BarService barService)
    {
        // omitted for brevity
    }
}
```

In the preceding code, assume that logging has been added and is resolvable from the service provider but the `FooService` and `BarService` types are not. The constructor with the `ILogger<ExampleService>` parameter is used to resolve the `ExampleService` instance. Even though there's a constructor that defines more parameters, the `FooService` and `BarService` types are not DI-resolvable.

If there's ambiguity when discovering constructors, an exception is thrown. Consider the following C# example

service:

```
public class ExampleService
{
    public ExampleService()
    {
    }

    public ExampleService(ILocator<ExampleService> logger)
    {
        // omitted for brevity
    }

    public ExampleService(IOptions<ExampleOptions> options)
    {
        // omitted for brevity
    }
}
```

WARNING

The `ExampleService` code with ambiguous DI-resolvable type parameters would throw an exception. Do **not** do this—it's intended to show what is meant by "ambiguous DI-resolvable types".

In the preceding example, there are three constructors. The first constructor is parameterless and requires no services from the service provider. Assume that both logging and options have been added to the DI container and are DI-resolvable services. When the DI container attempts to resolve the `ExampleService` type, it will throw an exception, as the two constructors are ambiguous.

You can avoid ambiguity by defining a constructor that accepts both DI-resolvable types instead:

```
public class ExampleService
{
    public ExampleService()
    {
    }

    public ExampleService(
        ILocator<ExampleService> logger,
        IOptions<ExampleOptions> options)
    {
        // omitted for brevity
    }
}
```

Register groups of services with extension methods

Microsoft Extensions uses a convention for registering a group of related services. The convention is to use a single `Add{GROUP_NAME}` extension method to register all of the services required by a framework feature. For example, the `AddOptions` extension method registers all of the services required for using options.

Framework-provided services

The `ConfigureServices` method registers services that the app uses, including platform features. Initially, the `IServiceCollection` provided to `ConfigureServices` has services defined by the framework depending on [how the host was configured](#). For apps based on the .NET templates, the framework registers hundreds of services.

The following table lists a small sample of these framework-registered services:

SERVICE TYPE	LIFETIME
<code>Microsoft.Extensions.DependencyInjection.IServiceScopeFactory</code>	Singleton
<code>IHostApplicationLifetime</code>	Singleton
<code>Microsoft.Extensions.Logging.ILogger<TCategoryName></code>	Singleton
<code>Microsoft.Extensions.Logging.ILoggerFactory</code>	Singleton
<code>Microsoft.Extensions.ObjectPool.ObjectPoolProvider</code>	Singleton
<code>Microsoft.Extensions.Options.IConfigureOptions<TOptions></code>	Transient
<code>Microsoft.Extensions.Options.IOptions<TOptions></code>	Singleton
<code>System.Diagnostics.DiagnosticListener</code>	Singleton
<code>System.Diagnostics.DiagnosticSource</code>	Singleton

Service lifetimes

Services can be registered with one of the following lifetimes:

- Transient
- Scoped
- Singleton

The following sections describe each of the preceding lifetimes. Choose an appropriate lifetime for each registered service.

Transient

Transient lifetime services are created each time they're requested from the service container. This lifetime works best for lightweight, stateless services. Register transient services with [AddTransient](#).

In apps that process requests, transient services are disposed at the end of the request.

Scoped

For web applications, a scoped lifetime indicates that services are created once per client request (connection). Register scoped services with [AddScoped](#).

In apps that process requests, scoped services are disposed at the end of the request.

When using Entity Framework Core, the [AddDbContext](#) extension method registers `DbContext` types with a scoped lifetime by default.

NOTE

Do *not* resolve a scoped service from a singleton and be careful not to do so indirectly, for example, through a transient service. It may cause the service to have incorrect state when processing subsequent requests. It's fine to:

- Resolve a singleton service from a scoped or transient service.
- Resolve a scoped service from another scoped or transient service.

By default, in the development environment, resolving a service from another service with a longer lifetime throws an exception. For more information, see [Scope validation](#).

Singleton

Singleton lifetime services are created either:

- The first time they're requested.
- By the developer, when providing an implementation instance directly to the container. This approach is rarely needed.

Every subsequent request of the service implementation from the dependency injection container uses the same instance. If the app requires singleton behavior, allow the service container to manage the service's lifetime. Don't implement the singleton design pattern and provide code to dispose of the singleton. Services should never be disposed by code that resolved the service from the container. If a type or factory is registered as a singleton, the container disposes the singleton automatically.

Register singleton services with [AddSingleton](#). Singleton services must be thread safe and are often used in stateless services.

In apps that process requests, singleton services are disposed when the [ServiceProvider](#) is disposed on application shutdown. Because memory is not released until the app is shut down, consider memory use with a singleton service.

Service registration methods

The framework provides service registration extension methods that are useful in specific scenarios:

METHOD	AUTOMATIC OBJECT DISPOSAL	MULTIPLE IMPLEMENTATIONS	PASS ARGS
<pre>Add{LIFETIME} <{SERVICE}> {IMPLEMENTATION}>()</pre> Example: <pre>services.AddSingleton<IMyDep, MyDep>();</pre>	Yes	Yes	No
<pre>Add{LIFETIME} <{SERVICE}>(sp => new {IMPLEMENTATION})</pre> Examples: <pre>services.AddSingleton<IMyDep> (sp => new MyDep()); services.AddSingleton<IMyDep> (sp => new MyDep(99));</pre>	Yes	Yes	Yes
<pre>Add{LIFETIME} <{IMPLEMENTATION}>()</pre> Example: <pre>services.AddSingleton<MyDep> ();</pre>	Yes	No	No

METHOD	AUTOMATIC OBJECT DISPOSAL	MULTIPLE IMPLEMENTATIONS	PASS ARGS
AddSingleton<{SERVICE}>(new {IMPLEMENTATION})	No	Yes	Yes
Examples: <pre>services.AddSingleton<IMyDep> (new MyDep()); services.AddSingleton<IMyDep> (new MyDep(99));</pre>			
AddSingleton(new {IMPLEMENTATION})	No	No	Yes
Examples: <pre>services.AddSingleton(new MyDep()); services.AddSingleton(new MyDep(99));</pre>			

For more information on type disposal, see the [Disposal of services](#) section.

Registering a service with only an implementation type is equivalent to registering that service with the same implementation and service type. This is why multiple implementations of a service cannot be registered using the methods that don't take an explicit service type. These methods can register multiple *instances* of a service, but they will all have the same *implementation* type.

Any of the above service registration methods can be used to register multiple service instances of the same service type. In the following example, `AddSingleton` is called twice with `IMessageWriter` as the service type. The second call to `AddSingleton` overrides the previous one when resolved as `IMessageWriter` and adds to the previous one when multiple services are resolved via `IEnumerable<IMessageWriter>`. Services appear in the order they were registered when resolved via `IEnumerable<{SERVICE}>`.

```
using ConsoleDI.IEnumerableExample;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;

namespace ConsoleDI.Example;

class Program
{
    static Task Main(string[] args)
    {
        using IHost host = CreateHostBuilder(args).Build();

        _ = host.Services.GetService<ExampleService>();

        return host.RunAsync();
    }

    static IHostBuilder CreateHostBuilder(string[] args) =>
        Host.CreateDefaultBuilder(args)
            .ConfigureServices(_>, services) =>
                services.AddSingleton<IMessageWriter, ConsoleMessageWriter>()
                    .AddSingleton<IMessageWriter, LoggingMessageWriter>()
                    .AddSingleton<ExampleService>();
    }
}
```

The preceding sample source code registers two implementations of the `IMessageWriter`.

```
using System.Diagnostics;

namespace ConsoleDI.IEnumerableExample;

public class ExampleService
{
    public ExampleService(
        IMessageWriter messageWriter,
        IEnumerable<IMessageWriter> messageWriters)
    {
        Trace.Assert(messageWriter is LoggingMessageWriter);

        var dependencyArray = messageWriters.ToArray();
        Trace.Assert(dependencyArray[0] is ConsoleMessageWriter);
        Trace.Assert(dependencyArray[1] is LoggingMessageWriter);
    }
}
```

The `ExampleService` defines two constructor parameters; a single `IMessageWriter`, and an `IEnumerable<IMessageWriter>`. The single `IMessageWriter` is the last implementation to have been registered, whereas the `IEnumerable<IMessageWriter>` represents all registered implementations.

The framework also provides `TryAdd{LIFETIME}` extension methods, which register the service only if there isn't already an implementation registered.

In the following example, the call to `AddSingleton` registers `ConsoleMessageWriter` as an implementation for `IMessageWriter`. The call to `TryAddSingleton` has no effect because `IMessageWriter` already has a registered implementation:

```
services.AddSingleton<IMessageWriter, ConsoleMessageWriter>();
services.TryAddSingleton<IMessageWriter, LoggingMessageWriter>();
```

The `TryAddSingleton` has no effect, as it was already added and the "try" will fail. The `ExampleService` would assert the following:

```
public class ExampleService
{
    public ExampleService(
        IMessageWriter messageWriter,
        IEnumerable<IMessageWriter> messageWriters)
    {
        Trace.Assert(messageWriter is ConsoleMessageWriter);
        Trace.Assert(messageWriters.Single() is ConsoleMessageWriter);
    }
}
```

For more information, see:

- [TryAdd](#)
- [TryAddTransient](#)
- [TryAddScoped](#)
- [TryAddSingleton](#)

The `TryAddEnumerable(ServiceDescriptor)` methods register the service only if there isn't already an implementation of the same type. Multiple services are resolved via `IEnumerable<{SERVICE}>`. When registering services, add an instance if one of the same types hasn't already been added. Library authors use

`TryAddEnumerable` to avoid registering multiple copies of an implementation in the container.

In the following example, the first call to `TryAddEnumerable` registers `MessageWriter` as an implementation for `IMessageWriter1`. The second call registers `MessageWriter` for `IMessageWriter2`. The third call has no effect because `IMessageWriter1` already has a registered implementation of `MessageWriter`:

```
public interface IMessageWriter1 { }
public interface IMessageWriter2 { }

public class MessageWriter : IMessageWriter1, IMessageWriter2
{
}

services.TryAddEnumerable(ServiceDescriptor.Singleton<IMessageWriter1, MessageWriter>());
services.TryAddEnumerable(ServiceDescriptor.Singleton<IMessageWriter2, MessageWriter>());
services.TryAddEnumerable(ServiceDescriptor.Singleton<IMessageWriter1, MessageWriter>());
```

Service registration is generally order-independent except when registering multiple implementations of the same type.

`IServiceCollection` is a collection of `ServiceDescriptor` objects. The following example shows how to register a service by creating and adding a `ServiceDescriptor`:

```
string secretKey = Configuration["SecretKey"];
var descriptor = new ServiceDescriptor(
    typeof(IMessageWriter),
    _ => new DefaultMessageWriter(secretKey),
    ServiceLifetime.Transient);

services.Add(descriptor);
```

The built-in `Add{LIFETIME}` methods use the same approach. For example, see the [AddScoped source code](#).

Constructor injection behavior

Services can be resolved by using:

- [IServiceProvider](#)
- [ActivatorUtilities](#):
 - Creates objects that aren't registered in the container.
 - Used with some framework features.

Constructors can accept arguments that aren't provided by dependency injection, but the arguments must assign default values.

When services are resolved by `IServiceProvider` or `ActivatorUtilities`, constructor injection requires a *public* constructor.

When services are resolved by `ActivatorUtilities`, constructor injection requires that only one applicable constructor exists. Constructor overloads are supported, but only one overload can exist whose arguments can all be fulfilled by dependency injection.

Scope validation

When the app runs in the `Development` environment and calls `CreateDefaultBuilder` to build the host, the default service provider performs checks to verify that:

- Scoped services aren't resolved from the root service provider.

- Scoped services aren't injected into singletons.

The root service provider is created when [BuildServiceProvider](#) is called. The root service provider's lifetime corresponds to the app's lifetime when the provider starts with the app and is disposed when the app shuts down.

Scoped services are disposed by the container that created them. If a scoped service is created in the root container, the service's lifetime is effectively promoted to singleton because it's only disposed by the root container when the app shuts down. Validating service scopes catches these situations when

[BuildServiceProvider](#) is called.

Scope scenarios

The [IServiceScopeFactory](#) is always registered as a singleton, but the [IServiceProvider](#) can vary based on the lifetime of the containing class. For example, if you resolve services from a scope, and any of those services take an [IServiceProvider](#), it'll be a scoped instance.

To achieve scoping services within implementations of [IHostedService](#), such as the [BackgroundService](#), do *not* inject the service dependencies via constructor injection. Instead, inject [IServiceScopeFactory](#), create a scope, then resolve dependencies from the scope to use the appropriate service lifetime.

```

namespace WorkerScope.Example;

public class Worker : BackgroundService
{
    private readonly ILogger<Worker> _logger;
    private readonly IServiceScopeFactory _serviceScopeFactory;

    public Worker(ILogger<Worker> logger, IServiceScopeFactory serviceScopeFactory) =>
        (_logger, _serviceScopeFactory) = (logger, serviceScopeFactory);

    protected override async Task ExecuteAsync(CancellationToken stoppingToken)
    {
        while (!stoppingToken.IsCancellationRequested)
        {
            using (IServiceScope scope = _serviceScopeFactory.CreateScope())
            {
                try
                {
                    _logger.LogInformation(
                        "Starting scoped work, provider hash: {hash}.",
                        scope.ServiceProvider.GetHashCode());

                    var store = scope.ServiceProvider.GetRequiredService<IObjectStore>();
                    var next = await store.GetNextAsync();
                    _logger.LogInformation("{next}", next);

                    var processor = scope.ServiceProvider.GetRequiredService<IObjectProcessor>();
                    await processor.ProcessAsync(next);
                    _logger.LogInformation("Processing {name}.", next.Name);

                    var relay = scope.ServiceProvider.GetRequiredService<IObjectRelay>();
                    await relay.RelayAsync(next);
                    _logger.LogInformation("Processed results have been relayed.");

                    var marked = await store.MarkAsync(next);
                    _logger.LogInformation("Marked as processed: {next}", marked);
                }
                finally
                {
                    _logger.LogInformation(
                        "Finished scoped work, provider hash: {hash}.{nl}",
                        scope.ServiceProvider.GetHashCode(), Environment.NewLine);
                }
            }
        }
    }
}

```

In the preceding code, while the app is running, the background service:

- Depends on the [IServiceScopeFactory](#).
- Creates an [IServiceScope](#) for resolving additional services.
- Resolves scoped services for consumption.
- Works on processing objects and then relaying them, and finally marks them as processed.

From the sample source code, you can see how implementations of [IHostedService](#) can benefit from scoped service lifetimes.

See also

- [Use dependency injection in .NET](#)
- [Dependency injection guidelines](#)

- Dependency injection in ASP.NET Core
- NDC Conference Patterns for DI app development
- Explicit dependencies principle
- Inversion of control containers and the dependency injection pattern (Martin Fowler)
- DI bugs should be created in the github.com/dotnet/extensions repo

Tutorial: Use dependency injection in .NET

9/20/2022 • 3 minutes to read • [Edit Online](#)

This tutorial shows how to use [dependency injection \(DI\) in .NET](#). With *Microsoft Extensions*, DI is a first-class citizen where services are added and configured in an [IServiceCollection](#). The [IHost](#) interface exposes the [IServiceProvider](#) instance, which acts as a container of all the registered services.

In this tutorial, you learn how to:

- Create a .NET console app that uses dependency injection
- Build and configure a [Generic Host](#)
- Write several interfaces and corresponding implementations
- Use service lifetime and scoping for DI

Prerequisites

- [.NET Core 3.1 SDK](#) or later.
- Familiarity with creating new .NET applications and installing NuGet packages.

Create a new console application

Using either the [dotnet new](#) command or an IDE new project wizard, create a new .NET console application named `ConsoleDI.Example`. Add the [Microsoft.Extensions.Hosting](#) NuGet package to the project.

Add interfaces

Add the following interfaces to the project root directory:

`IOperation.cs`

```
namespace ConsoleDI.Example;

public interface IOperation
{
    string OperationId { get; }
}
```

The `IOperation` interface defines a single `OperationId` property.

`ITransientOperation.cs`

```
namespace ConsoleDI.Example;

public interface ITransientOperation : IOperation
{
}
```

`IScopedOperation.cs`

```
namespace ConsoleDI.Example;

public interface IScopedOperation : IOperation
{
}
```

ISingletonOperation.cs

```
namespace ConsoleDI.Example;

public interface ISingletonOperation : IOperation
{
}
```

All of the subinterfaces of `IOperation` name their intended service lifetime. For example, "Transient" or "Singleton".

Add default implementation

Add the following default implementation for the various operations:

DefaultOperation.cs

```
using static System.Guid;

namespace ConsoleDI.Example;

public class DefaultOperation :
    ITransientOperation,
    IScopedOperation,
    ISingletonOperation
{
    public string OperationId { get; } = NewGuid().ToString()[^4..];
}
```

The `DefaultOperation` implements all of the named marker interfaces and initializes the `OperationId` property to the last four characters of a new globally unique identifier (GUID).

Add service that requires DI

Add the following operation logger object, which acts as a service to the console app:

OperationLogger.cs

```

namespace ConsoleDI.Example;

public class OperationLogger
{
    private readonly ITransientOperation _transientOperation;
    private readonly IScopedOperation _scopedOperation;
    private readonly ISingletonOperation _singletonOperation;

    public OperationLogger(
        ITransientOperation transientOperation,
        IScopedOperation scopedOperation,
        ISingletonOperation singletonOperation) =>
        (_transientOperation, _scopedOperation, _singletonOperation) =
            (transientOperation, scopedOperation, singletonOperation);

    public void LogOperations(string scope)
    {
        LogOperation(_transientOperation, scope, "Always different");
        LogOperation(_scopedOperation, scope, "Changes only with scope");
        LogOperation(_singletonOperation, scope, "Always the same");
    }

    private static void LogOperation<T>(T operation, string scope, string message)
        where T : IOperation =>
        Console.WriteLine(
            $"{scope}: {typeof(T).Name,-19} [ {operation.OperationId}...{message,-23} ]");
}

```

The `operationLogger` defines a constructor that requires each of the aforementioned marker interfaces, that is, `ITransientOperation`, `IScopedOperation`, and `ISingletonOperation`. The object exposes a single method that allows the consumer to log the operations with a given `scope` parameter. When invoked, the `LogOperations` method logs each operation's unique identifier with the scope string and message.

Register services for DI

Update `Program.cs` with the following code:

```

using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using ConsoleDI.Example;

using IHost host = Host.CreateDefaultBuilder(args)
    .ConfigureServices(_ , services) =>
    services.AddTransient<ITransientOperation, DefaultOperation>()
        .AddScoped<IScopedOperation, DefaultOperation>()
        .AddSingleton<ISingletonOperation, DefaultOperation>()
        .AddTransient<OperationLogger>()
    .Build();

ExemplifyScoping(host.Services, "Scope 1");
ExemplifyScoping(host.Services, "Scope 2");

await host.RunAsync();

static void ExemplifyScoping(IServiceProvider services, string scope)
{
    using IServiceScope serviceScope = services.CreateScope();
    IServiceProvider provider = serviceScope.ServiceProvider;

    OperationLogger logger = provider.GetRequiredService<OperationLogger>();
    logger.LogOperations($"{scope}-Call 1 .GetRequiredService<OperationLogger>()");

    Console.WriteLine("...");

    logger = provider.GetRequiredService<OperationLogger>();
    logger.LogOperations($"{scope}-Call 2 .GetRequiredService<OperationLogger>()");

    Console.WriteLine();
}

```

Each `services.Add{LIFETIME}<{SERVICE}>` extension method adds (and potentially configures) services. We recommended that apps follow this convention. Place extension methods in the [Microsoft.Extensions.DependencyInjection](#) namespace to encapsulate groups of service registrations. Including the namespace portion `Microsoft.Extensions.DependencyInjection` for DI extension methods also:

- Allows them to be displayed in [IntelliSense](#) without adding additional `using` blocks.
- Prevents excessive `using` statements in the `Program` or `Startup` classes where these extension methods are typically called.

The app:

- Creates an `IHostBuilder` instance with the [default binder settings](#).
- Configures services and adds them with their corresponding service lifetime.
- Calls `Build()` and assigns an instance of `IHost`.
- Calls `ExemplifyScoping`, passing in the `IHostServices`.

Conclusion

The app displays output similar to the following example:

```
Scope 1-Call 1 .GetRequiredService<OperationLogger>(): ITransientOperation [ 80f4...Always different
]
Scope 1-Call 1 .GetRequiredService<OperationLogger>(): IScopedOperation      [ c878...Changes only with scope
]
Scope 1-Call 1 .GetRequiredService<OperationLogger>(): ISingletonOperation [ 1586...Always the same
]
...
Scope 1-Call 2 .GetRequiredService<OperationLogger>(): ITransientOperation [ f3c0...Always different
]
Scope 1-Call 2 .GetRequiredService<OperationLogger>(): IScopedOperation      [ c878...Changes only with scope
]
Scope 1-Call 2 .GetRequiredService<OperationLogger>(): ISingletonOperation [ 1586...Always the same
]

Scope 2-Call 1 .GetRequiredService<OperationLogger>(): ITransientOperation [ f9af...Always different
]
Scope 2-Call 1 .GetRequiredService<OperationLogger>(): IScopedOperation      [ 2bd0...Changes only with scope
]
Scope 2-Call 1 .GetRequiredService<OperationLogger>(): ISingletonOperation [ 1586...Always the same
]
...
Scope 2-Call 2 .GetRequiredService<OperationLogger>(): ITransientOperation [ fa65...Always different
]
Scope 2-Call 2 .GetRequiredService<OperationLogger>(): IScopedOperation      [ 2bd0...Changes only with scope
]
Scope 2-Call 2 .GetRequiredService<OperationLogger>(): ISingletonOperation [ 1586...Always the same
]
```

From the app output, you can see that:

- Transient operations are always different, a new instance is created with every retrieval of the service.
- Scoped operations change only with a new scope, but are the same instance within a scope.
- Singleton operations are always the same, a new instance is only created once.

See also

- [Dependency injection guidelines](#)
- [Dependency injection in ASP.NET Core](#)

Dependency injection guidelines

9/20/2022 • 9 minutes to read • [Edit Online](#)

This article provides general guidelines and best practices for implementing dependency injection in .NET applications.

Design services for dependency injection

When designing services for dependency injection:

- Avoid stateful, static classes and members. Avoid creating global state by designing apps to use singleton services instead.
- Avoid direct instantiation of dependent classes within services. Direct instantiation couples the code to a particular implementation.
- Make services small, well-factored, and easily tested.

If a class has many injected dependencies, it might be a sign that the class has too many responsibilities and violates the [Single Responsibility Principle \(SRP\)](#). Attempt to refactor the class by moving some of its responsibilities into new classes.

Disposal of services

The container is responsible for cleanup of types it creates, and calls [Dispose](#) on [IDisposable](#) instances. Services resolved from the container should never be disposed by the developer. If a type or factory is registered as a singleton, the container disposes the singleton automatically.

In the following example, the services are created by the service container and disposed automatically:

```
namespace ConsoleDisposable.Example;

public sealed class TransientDisposable : IDisposable
{
    public void Dispose() => Console.WriteLine($"{nameof(TransientDisposable)}.Dispose()");
}
```

The preceding disposable is intended to have a transient lifetime.

```
namespace ConsoleDisposable.Example;

public sealed class ScopedDisposable : IDisposable
{
    public void Dispose() => Console.WriteLine($"{nameof(ScopedDisposable)}.Dispose()");
}
```

The preceding disposable is intended to have a scoped lifetime.

```
namespace ConsoleDisposable.Example;

public sealed class SingletonDisposable : IDisposable
{
    public void Dispose() => Console.WriteLine($"{nameof(SingletonDisposable)}.Dispose()");
}
```

The preceding disposable is intended to have a singleton lifetime.

```
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;

namespace ConsoleDisposable.Example;

class Program
{
    static async Task Main(string[] args)
    {
        using IHost host = CreateHostBuilder(args).Build();

        ExemplifyDisposableScoping(host.Services, "Scope 1");
        Console.WriteLine();

        ExemplifyDisposableScoping(host.Services, "Scope 2");
        Console.WriteLine();

        await host.RunAsync();
    }

    static IHostBuilder CreateHostBuilder(string[] args) =>
        Host.CreateDefaultBuilder(args)
            .ConfigureServices(_ => services)
            .AddTransient<TransientDisposable>()
            .AddScoped<ScopedDisposable>()
            .AddSingleton<SingletonDisposable>();

    static void ExemplifyDisposableScoping(IServiceProvider services, string scope)
    {
        Console.WriteLine($"{scope}...");

        using IServiceScope serviceScope = services.CreateScope();
        IServiceProvider provider = serviceScope.ServiceProvider;

        _ = provider.GetRequiredService<TransientDisposable>();
        _ = provider.GetRequiredService<ScopedDisposable>();
        _ = provider.GetRequiredService<SingletonDisposable>();
    }
}
```

The debug console shows the following sample output after running:

```
Scope 1...
ScopedDisposable.Dispose()
TransientDisposable.Dispose()

Scope 2...
ScopedDisposable.Dispose()
TransientDisposable.Dispose()

info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Production
info: Microsoft.Hosting.Lifetime[0]
      Content root path: .\configuration\console-di-disposable\bin\Debug\net5.0
info: Microsoft.Hosting.Lifetime[0]
      Application is shutting down...
SingletonDisposable.Dispose()
```

Services not created by the service container

Consider the following code:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddSingleton(new ExampleService());

    // ...
}
```

In the preceding code:

- The `ExampleService` instance is **not** created by the service container.
- The framework does **not** dispose of the services automatically.
- The developer is responsible for disposing the services.

IDisposable guidance for transient and shared instances

Transient, limited lifetime

Scenario

The app requires an `IDisposable` instance with a transient lifetime for either of the following scenarios:

- The instance is resolved in the root scope (root container).
- The instance should be disposed before the scope ends.

Solution

Use the factory pattern to create an instance outside of the parent scope. In this situation, the app would generally have a `Create` method that calls the final type's constructor directly. If the final type has other dependencies, the factory can:

- Receive an `IServiceProvider` in its constructor.
- Use `ActivatorUtilities.CreateInstance` to instantiate the instance outside of the container, while using the container for its dependencies.

Shared instance, limited lifetime

Scenario

The app requires a shared `IDisposable` instance across multiple services, but the `IDisposable` instance should have a limited lifetime.

Solution

Register the instance with a scoped lifetime. Use `IServiceScopeFactory.CreateScope` to create a new `IServiceScope`. Use the scope's `IServiceProvider` to get required services. Dispose the scope when it's no longer needed.

General `IDisposable` guidelines

- Don't register `IDisposable` instances with a transient lifetime. Use the factory pattern instead.
- Don't resolve `IDisposable` instances with a transient or scoped lifetime in the root scope. The only exception to this is if the app creates/recreates and disposes `IServiceProvider`, but this isn't an ideal pattern.
- Receiving an `IDisposable` dependency via DI doesn't require that the receiver implement `IDisposable` itself. The receiver of the `IDisposable` dependency shouldn't call `Dispose` on that dependency.
- Use scopes to control the lifetimes of services. Scopes aren't hierarchical, and there's no special connection among scopes.

For more information on resource cleanup, see [Implement a `Dispose` method](#), or [Implement a `DisposeAsync` method](#). Additionally, consider the [Disposable transient services captured by container scenario](#) as it relates to resource cleanup.

Default service container replacement

The built-in service container is designed to serve the needs of the framework and most consumer apps. We recommend using the built-in container unless you need a specific feature that it doesn't support, such as:

- Property injection
- Injection based on name
- Child containers
- Custom lifetime management
- `Func<T>` support for lazy initialization
- Convention-based registration

The following third-party containers can be used with ASP.NET Core apps:

- [Autofac](#)
- [Dryloc](#)
- [Grace](#)
- [LightInject](#)
- [Lamar](#)
- [Stashbox](#)
- [Unity](#)
- [Simple Injector](#)

Thread safety

Create thread-safe singleton services. If a singleton service has a dependency on a transient service, the transient service may also require thread safety depending on how it's used by the singleton.

The factory method of a singleton service, such as the second argument to `AddSingleton<TService>(IServiceCollection, Func<IServiceProvider, TService>)`, doesn't need to be thread-safe. Like a type (`static`) constructor, it's guaranteed to be called only once by a single thread.

Recommendations

- `async/await` and `Task` based service resolution isn't supported. Because C# doesn't support asynchronous constructors, use asynchronous methods after synchronously resolving the service.
- Avoid storing data and configuration directly in the service container. For example, a user's shopping cart shouldn't typically be added to the service container. Configuration should use the options pattern. Similarly, avoid "data holder" objects that only exist to allow access to another object. It's better to request the actual item via DI.
- Avoid static access to services. For example, avoid capturing `IApplicationBuilder.ApplicationServices` as a static field or property for use elsewhere.
- Keep [DI factories](#) fast and synchronous.
- Avoid using the [service locator pattern](#). For example, don't invoke `GetService` to obtain a service instance when you can use DI instead.
- Another service locator variation to avoid is injecting a factory that resolves dependencies at run time. Both of these practices mix [Inversion of Control](#) strategies.
- Avoid calls to `BuildServiceProvider` in `ConfigureServices`. Calling `BuildServiceProvider` typically happens when the developer wants to resolve a service in `ConfigureServices`.
- [Disposable transient services are captured](#) by the container for disposal. This can turn into a memory leak if resolved from the top-level container.
- Enable scope validation to make sure the app doesn't have singletons that capture scoped services. For more

information, see [Scope validation](#).

Like all sets of recommendations, you may encounter situations where ignoring a recommendation is required. Exceptions are rare, mostly special cases within the framework itself.

DI is an *alternative* to static/global object access patterns. You may not be able to realize the benefits of DI if you mix it with static object access.

Example anti-patterns

In addition to the guidelines in this article, there are several anti-patterns *you should avoid*. Some of these anti-patterns are learnings from developing the runtimes themselves.

WARNING

These are example anti-patterns, *do not copy the code, do not use these patterns, and avoid these patterns at all costs.*

Disposable transient services captured by container

When you register *Transient* services that implement [IDisposable](#), by default the DI container will hold onto these references, and not [Dispose\(\)](#) of them until the container is disposed when application stops if they were resolved from the container, or until the scope is disposed if they were resolved from a scope. This can turn into a memory leak if resolved from container level.

```
static void TransientDisposablesWithoutDispose()
{
    var services = new ServiceCollection();
    services.AddTransient<ExampleDisposable>();
    ServiceProvider serviceProvider = services.BuildServiceProvider();

    for (int i = 0; i < 1000; ++ i)
    {
        _ = serviceProvider.GetRequiredService<ExampleDisposable>();
    }

    // serviceProvider.Dispose();
}
```

In the preceding anti-pattern, 1,000 `ExampleDisposable` objects are instantiated and rooted. They will not be disposed of until the `serviceProvider` instance is disposed.

For more information on debugging memory leaks, see [Debug a memory leak in .NET](#).

Async DI factories can cause deadlocks

The term "DI factories" refers to the overload methods that exist when calling `Add{LIFETIME}`. There are overloads accepting a `Func<IServiceProvider, T>` where `T` is the service being registered, and the parameter is named `implementationFactory`. The `implementationFactory` can be provided as a lambda expression, local function, or method. If the factory is asynchronous, and you use `Task<TResult>.Result`, this will cause a deadlock.

```

static void DeadLockWithAsyncFactory()
{
    var services = new ServiceCollection();
    services.AddSingleton<Foo>(implementationFactory: provider =>
    {
        Bar bar = GetBarAsync(provider).Result;
        return new Foo(bar);
    });

    services.AddSingleton<Bar>();

    using ServiceProvider serviceProvider = services.BuildServiceProvider();
    _ = serviceProvider.GetRequiredService<Foo>();
}

```

In the preceding code, the `implementationFactory` is given a lambda expression where the body calls `Task<TResult>.Result` on a `Task<Bar>` returning method. This *causes a deadlock*. The `GetBarAsync` method simply emulates an asynchronous work operation with `Task.Delay`, and then calls `GetRequiredService<T>(IServiceProvider)`.

```

static async Task<Bar> GetBarAsync(IServiceProvider serviceProvider)
{
    // Emulate asynchronous work operation
    await Task.Delay(1000);

    return serviceProvider.GetRequiredService<Bar>();
}

```

For more information on asynchronous guidance, see [Asynchronous programming: Important info and advice](#). For more information debugging deadlocks, see [Debug a deadlock in .NET](#).

When you're running this anti-pattern and the deadlock occurs, you can view the two threads waiting from Visual Studio's Parallel Stacks window. For more information, see [View threads and tasks in the Parallel Stacks window](#).

Captive dependency

The term "[captive dependency](#)" was coined by [Mark Seemann](#), and refers to the misconfiguration of service lifetimes, where a longer-lived service holds a shorter-lived service captive.

```

static void CaptiveDependency()
{
    var services = new ServiceCollection();
    services.AddSingleton<Foo>();
    services.AddScoped<Bar>();

    using ServiceProvider serviceProvider = services.BuildServiceProvider();
    // Enable scope validation
    // using ServiceProvider serviceProvider = services.BuildServiceProvider(validateScopes: true);

    _ = serviceProvider.GetRequiredService<Foo>();
}

```

In the preceding code, `Foo` is registered as a singleton and `Bar` is scoped - which on the surface seems valid. However, consider the implementation of `Foo`.

```
namespace DependencyInjection.AntiPatterns
{
    public class Foo
    {
        public Foo(Bar bar)
        {
        }
    }
}
```

The `Foo` object requires a `Bar` object, and since `Foo` is a singleton, and `Bar` is scoped - this is a misconfiguration. As is, `Foo` would only be instantiated once, and it would hold onto `Bar` for its lifetime, which is longer than the intended scoped lifetime of `Bar`. You should consider validating scopes, by passing `validateScopes: true` to the `BuildServiceProvider(IServiceCollection, Boolean)`. When you validate the scopes, you'd get an `InvalidOperationException` with a message similar to "Cannot consume scoped service 'Bar' from singleton 'Foo'.".

For more information, see [Scope validation](#).

Scoped service as singleton

When using scoped services, if you're not creating a scope or within an existing scope - the service becomes a singleton.

```
static void ScopedServiceBecomesSingleton()
{
    var services = new ServiceCollection();
    services.AddScoped<Bar>();

    using ServiceProvider serviceProvider = services.BuildServiceProvider(validateScopes: true);
    using (IServiceScope scope = serviceProvider.CreateScope())
    {
        // Correctly scoped resolution
        Bar correct = scope.ServiceProvider.GetRequiredService<Bar>();
    }

    // Not within a scope, becomes a singleton
    Bar avoid = serviceProvider.GetRequiredService<Bar>();
}
```

In the preceding code, `Bar` is retrieved within an `IServiceScope`, which is correct. The anti-pattern is the retrieval of `Bar` outside of the scope, and the variable is named `avoid` to show which example retrieval is incorrect.

See also

- [Dependency injection in .NET](#)
- [Tutorial: Use dependency injection in .NET](#)

Configuration in .NET

9/20/2022 • 7 minutes to read • [Edit Online](#)

Configuration in .NET is performed using one or more [configuration providers](#). Configuration providers read configuration data from key-value pairs using a variety of configuration sources:

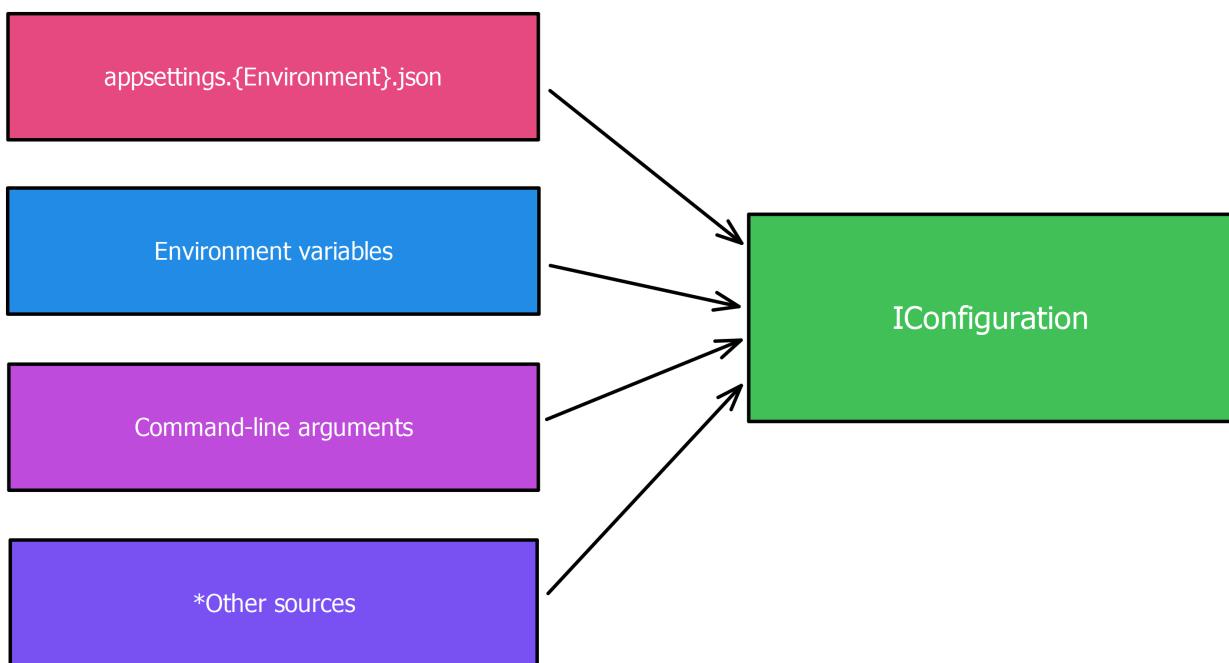
- Settings files, such as `appsettings.json`
- Environment variables
- [Azure Key Vault](#)
- [Azure App Configuration](#)
- Command-line arguments
- Custom providers, installed or created
- Directory files
- In-memory .NET objects
- Third-party providers

NOTE

For information about configuring the .NET runtime itself, see [.NET Runtime configuration settings](#).

Concepts and abstractions

Given one or more configuration sources, the [IConfiguration](#) type provides a unified view of the configuration data. Configuration is read-only, and the configuration pattern is not designed to be programmatically writable. The [IConfiguration](#) interface is a single representation of all the configuration sources, as shown in the following diagram:



Configure console apps

.NET console applications created using the `dotnet new` command template or Visual Studio by default *do not*

expose configuration capabilities. To add configuration in a new .NET console application, [add a package reference to Microsoft.Extensions.Hosting](#). Modify the *Program.cs* file to match the following code:

```
using Microsoft.Extensions.Hosting;

using IHost host = Host.CreateDefaultBuilder(args).Build();

// Application code should start here.

await host.RunAsync();
```

The [Host.CreateDefaultBuilder\(String\[\]\)](#) method provides default configuration for the app in the following order, from highest to lowest priority:

1. Command-line arguments using the [Command-line configuration provider](#).
2. Environment variables using the [Environment Variables configuration provider](#).
3. [App secrets](#) when the app runs in the `Development` environment.
4. `appsettings.[Environment].json` using the [JSON configuration provider](#). For example, `appsettings.Production.json` and `appsettings.Development.json`.
5. `appsettings.json` using the [JSON configuration provider](#).
6. [ChainedConfigurationProvider](#) : Adds an existing `IConfiguration` as a source.

Adding a configuration provider overrides previous configuration values. For example, the [Command-line configuration provider](#) overrides all values from other providers because it's added last. If `SomeKey` is set in both `appsettings.json` and the environment, the environment value is used because it was added after `appsettings.json`.

Binding

One of the key advantages of using the .NET configuration abstractions is the ability to bind configuration values to instances of .NET objects. For example, the JSON configuration provider can be used to map `appsettings.json` files to .NET objects and is used with [dependency injection](#). This enables the [options pattern](#), which uses classes to provide strongly typed access to groups of related settings. .NET configuration provides various abstractions. Consider the following interfaces:

- [IConfiguration](#): Represents a set of key/value application configuration properties.
- [IConfigurationRoot](#): Represents the root of an IConfiguration hierarchy.
- [IConfigurationSection](#): Represents a section of application configuration values.

These abstractions are agnostic to their underlying configuration provider ([IConfigurationProvider](#)). In other words, you can use an `IConfiguration` instance to access any configuration value from multiple providers.

The binder can use different approaches to process configuration values:

- Direct deserialization (using built-in converters) for primitive types.
- The [TypeConverter](#) for a complex type when the type has one.
- Reflection for a complex type that has properties.

NOTE

The binder has a few limitations:

- Properties are ignored if they have private setters or their type can't be converted.
- Properties without corresponding configuration keys are ignored.

Configuration values can contain hierarchical data. Hierarchical objects are represented with the use of the `:` delimiter in the configuration keys. To access a configuration value, use the `:` character to delimit a hierarchy. For example, consider the following configuration values:

```
{  
  "Parent": {  
    "FavoriteNumber": 7,  
    "Child": {  
      "Name": "Example",  
      "GrandChild": {  
        "Age": 3  
      }  
    }  
  }  
}
```

The following table represents example keys and their corresponding values for the preceding example JSON:

KEY	VALUE
"Parent:FavoriteNumber"	7
"Parent:Child:Name"	"Example"
"Parent:Child:GrandChild:Age"	3

Basic example

To access configuration values in their basic form, without the assistance of the *generic host* approach, use the [ConfigurationBuilder](#) type directly.

TIP

The [System.Configuration.ConfigurationBuilder](#) type is different to the [Microsoft.Extensions.Configuration.ConfigurationBuilder](#) type. All of this content is specific to the [Microsoft.Extensions.*](#) NuGet packages and namespaces.

Consider the following C# project:

```

<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>net6.0</TargetFramework>
    <Nullable>enable</Nullable>
    <ImplicitUsings>true</ImplicitUsings>
  </PropertyGroup>

  <ItemGroup>
    <Content Include="appsettings.json">
      <CopyToOutputDirectory>Always</CopyToOutputDirectory>
    </Content>
  </ItemGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.Extensions.Configuration.Binder" Version="6.0.0" />
    <PackageReference Include="Microsoft.Extensions.Configuration.Json" Version="6.0.0" />
    <PackageReference Include="Microsoft.Extensions.Configuration.EnvironmentVariables" Version="6.0.0" />
  </ItemGroup>

</Project>

```

The preceding project file references several configuration NuGet packages:

- [Microsoft.Extensions.Configuration.Binder](#): Functionality to bind an object to data in configuration providers for `Microsoft.Extensions.Configuration`.
- [Microsoft.Extensions.Configuration.Json](#): JSON configuration provider implementation for `Microsoft.Extensions.Configuration`.
- [Microsoft.Extensions.Configuration.EnvironmentVariables](#): Environment variables configuration provider implementation for `Microsoft.Extensions.Configuration`.

Consider an example *appsettings.json* file:

```
{
  "Settings": {
    "KeyOne": 1,
    "KeyTwo": true,
    "KeyThree": {
      "Message": "Oh, that's nice...",
      "SupportedVersions": {
        "v1": "1.0.0",
        "v3": "3.0.7"
      }
    },
    "IPAddressRange": [
      "46.36.198.121",
      "46.36.198.122",
      "46.36.198.123",
      "46.36.198.124",
      "46.36.198.125"
    ]
  }
}
```

Now, given this JSON file, here's an example consumption pattern using the configuration builder directly:

```

using Microsoft.Extensions.Configuration;

// Build a config object, using env vars and JSON providers.
IConfiguration config = new ConfigurationBuilder()
    .AddJsonFile("appsettings.json")
    .AddEnvironmentVariables()
    .Build();

// Get values from the config given their key and their target type.
Settings settings = config.GetRequiredSection("Settings").Get<Settings>();

// Write the values to the console.
Console.WriteLine($"KeyOne = {settings.KeyOne}");
Console.WriteLine($"KeyTwo = {settings.KeyTwo}");
Console.WriteLine($"KeyThree:Message = {settings.KeyThree.Message}");

// Application code which might rely on the config could start here.

// This will output the following:
//   KeyOne = 1
//   KeyTwo = True
//   KeyThree:Message = Oh, that's nice...

```

The preceding C# code:

- Instantiates a [ConfigurationBuilder](#).
- Adds the `"appsettings.json"` file to be recognized by the JSON configuration provider.
- Adds environment variables as being recognized by the Environment Variable configuration provider.
- Gets the required `"Settings"` section and the corresponding `settings` instance by using the `config` instance.

The `Settings` object is shaped as follows:

```

public sealed class Settings
{
    public int KeyOne { get; set; }
    public bool KeyTwo { get; set; }
    public NestedSettings KeyThree { get; set; } = null!;
}

```

```

public sealed class NestedSettings
{
    public string Message { get; set; } = null!;
}

```

Basic example with hosting

To access the `IConfiguration` value, you can rely again on the [Microsoft.Extensions.Hosting](#) NuGet package.

Create a new console application, and paste the following project file contents into it:

```
<Project Sdk="Microsoft.NET.Sdk">

<PropertyGroup>
  <OutputType>Exe</OutputType>
  <TargetFramework>net6.0</TargetFramework>
  <Nullable>enable</Nullable>
  <ImplicitUsings>true</ImplicitUsings>
</PropertyGroup>

<ItemGroup>
  <Content Include="appsettings.json">
    <CopyToOutputDirectory>Always</CopyToOutputDirectory>
  </Content>
</ItemGroup>

<ItemGroup>
  <PackageReference Include="Microsoft.Extensions.Hosting" Version="6.0.0" />
</ItemGroup>

</Project>
```

The preceding project file defines:

- That the application is an executable.
- That an *appsettings.json* file is to be copied to the output directory when the project is compiled.
- That the `Microsoft.Extensions.Hosting` NuGet package reference is added.

Add the *appsettings.json* file at the root of the project with the following contents:

```
{
  "KeyOne": 1,
  "KeyTwo": true,
  "KeyThree": {
    "Message": "Thanks for checking this out!"
  }
}
```

Replace the contents of the *Program.cs* file with the following C# code:

```

using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;

using IHost host = Host.CreateDefaultBuilder(args).Build();

// Ask the service provider for the configuration abstraction.
 IConfiguration config = host.Services.GetRequiredService<IConfiguration>();

// Get values from the config given their key and their target type.
int keyOneValue = config.GetValue<int>("KeyOne");
bool keyTwoValue = config.GetValue<bool>("KeyTwo");
string keyThreeNestedValue = config.GetValue<string>("KeyThree:Message");

// Write the values to the console.
Console.WriteLine($"KeyOne = {keyOneValue}");
Console.WriteLine($"KeyTwo = {keyTwoValue}");
Console.WriteLine($"KeyThree:Message = {keyThreeNestedValue}");

// Application code which might rely on the config could start here.

await host.RunAsync();

// This will output the following:
//   KeyOne = 1
//   KeyTwo = True
//   KeyThree:Message = Thanks for checking this out!

```

When you run this application, the `Host.CreateDefaultBuilder` defines the behavior to discover the JSON configuration and expose it through the `IConfiguration` instance. From the `host` instance, you can ask the service provider for the `IConfiguration` instance and then ask it for values.

TIP

Using the raw `IConfiguration` instance in this way, while convenient, doesn't scale very well. When applications grow in complexity, and their corresponding configurations become more complex, we recommend that you use the [options pattern](#) as an alternative.

Basic example with hosting and using the indexer API

Consider the same `appsettings.json` file contents from the previous example:

```
{
  "SupportedVersions": {
    "v1": "1.0.0",
    "v3": "3.0.7"
  },
  "IPAddressRange": [
    "46.36.198.123",
    "46.36.198.124",
    "46.36.198.125"
  ]
}
```

Replace the contents of the `Program.cs` file with the following C# code:

```

using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;

using IHost host = Host.CreateDefaultBuilder(args).Build();

// Ask the service provider for the configuration abstraction.
IConfiguration config = host.Services.GetRequiredService<IConfiguration>();

// Get values from the config given their key and their target type.
string ipOne = config["IPAddressRange:0"];
string ipTwo = config["IPAddressRange:1"];
string ipThree = config["IPAddressRange:2"];
string versionOne = config["SupportedVersions:v1"];
string versionThree = config["SupportedVersions:v3"];

// Write the values to the console.
Console.WriteLine($"IPAddressRange:0 = {ipOne}");
Console.WriteLine($"IPAddressRange:1 = {ipTwo}");
Console.WriteLine($"IPAddressRange:2 = {ipThree}");
Console.WriteLine($"SupportedVersions:v1 = {versionOne}");
Console.WriteLine($"SupportedVersions:v3 = {versionThree}");

// Application code which might rely on the config could start here.

await host.RunAsync();

// This will output the following:
//      IPAddressRange:0 = 46.36.198.123
//      IPAddressRange:1 = 46.36.198.124
//      IPAddressRange:2 = 46.36.198.125
//      SupportedVersions:v1 = 1.0.0
//      SupportedVersions:v3 = 3.0.7

```

The values are accessed using the indexer API where each key is a string, and the value is a string. Configuration supports properties, objects, arrays, and dictionaries.

Configuration providers

The following table shows the configuration providers available to .NET Core apps.

PROVIDER	PROVIDES CONFIGURATION FROM
Azure App configuration provider	Azure App Configuration
Azure Key Vault configuration provider	Azure Key Vault
Command-line configuration provider	Command-line parameters
Custom configuration provider	Custom source
Environment Variables configuration provider	Environment variables
File configuration provider	JSON, XML, and INI files
Key-per-file configuration provider	Directory files
Memory configuration provider	In-memory collections

PROVIDER	PROVIDES CONFIGURATION FROM
App secrets (Secret Manager)	File in the user profile directory

TIP

The order in which configuration providers are added matters. When multiple configuration providers are used and more than one provided specifies the same key, the last one added is used.

For more information on various configuration providers, see [Configuration providers in .NET](#).

See also

- [Configuration providers in .NET](#)
- [Implement a custom configuration provider](#)
- Configuration bugs should be created in the github.com/dotnet/extensions repo
- [Configuration in ASP.NET Core](#)

Configuration providers in .NET

9/20/2022 • 10 minutes to read • [Edit Online](#)

Configuration in .NET is possible with configuration providers. There are several types of providers that rely on various configuration sources. This article details all of the different configuration providers and their corresponding sources.

- [File configuration provider](#)
- [Environment variable configuration provider](#)
- [Command-line configuration provider](#)
- [Key-per-file configuration provider](#)
- [Memory configuration provider](#)

File configuration provider

[FileConfigurationProvider](#) is the base class for loading configuration from the file system. The following configuration providers derive from `FileConfigurationProvider`:

- [JSON configuration provider](#)
- [XML configuration provider](#)
- [INI configuration provider](#)

Keys are case-insensitive. All of the file configuration providers throw the [FormatException](#) when duplicate keys are found in a single provider.

JSON configuration provider

The [JsonConfigurationProvider](#) class loads configuration from a JSON file. Install the [Microsoft.Extensions.Configuration.Json](#) NuGet package.

Overloads can specify:

- Whether the file is optional.
- Whether the configuration is reloaded if the file changes.

Consider the following code:

```

using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.Hosting;
using ConsoleJson.Example;

using IHost host = Host.CreateDefaultBuilder(args)
    .ConfigureAppConfiguration((hostingContext, configuration) =>
{
    configuration.Sources.Clear();

    IHostEnvironment env = hostingContext.HostingEnvironment;

    configuration
        .AddJsonFile("appsettings.json", optional: true, reloadOnChange: true)
        .AddJsonFile($"appsettings.{env.EnvironmentName}.json", true, true);

    IConfigurationRoot configurationRoot = configuration.Build();

    TransientFaultHandlingOptions options = new();
    configurationRoot.GetSection(nameof(TransientFaultHandlingOptions))
        .Bind(options);

    Console.WriteLine($"TransientFaultHandlingOptions.Enabled={options.Enabled}");
    Console.WriteLine($"TransientFaultHandlingOptions.AutoRetryDelay={options.AutoRetryDelay}");
})
.Build();

// Application code should start here.

await host.RunAsync();

```

The preceding code:

- Clears all existing configuration providers that were added by default in the [CreateDefaultBuilder\(String\[\]\)](#) method.
- Configures the JSON configuration provider to load the *appsettings.json* and *appsettings.`Environment`.json* files with the following options:
 - `optional: true` : The file is optional.
 - `reloadOnChange: true` : The file is reloaded when changes are saved.

IMPORTANT

When [adding configuration providers](#) with [IConfigurationBuilder.Add](#), the added configuration provider is added to the end of the [IConfigurationSource](#) list. When keys are found by multiple providers, the last provider to read the key overrides previous providers.

An example *appsettings.json* file with various configuration settings follows:

```
{
    "SecretKey": "Secret key value",
    "TransientFaultHandlingOptions": {
        "Enabled": true,
        "AutoRetryDelay": "00:00:07"
    },
    "Logging": {
        "LogLevel": {
            "Default": "Information",
            "Microsoft": "Warning",
            "Microsoft.Hosting.Lifetime": "Information"
        }
    }
}
```

From the [IConfigurationBuilder](#) instance, after configuration providers have been added, you can call [IConfigurationBuilder.Build\(\)](#) to get the [IConfigurationRoot](#) object. The configuration root represents the root of a configuration hierarchy. Sections from the configuration can be bound to instances of .NET objects and later provided as [IOptions<TOptions>](#) through dependency injection.

NOTE

The *Build Action* and *Copy to Output Directory* properties of the JSON file must be set to *Content* and *Copy if newer (or Copy always)*, respectively.

Consider the `TransientFaultHandlingOptions` class defined as follows:

```
namespace ConsoleJson.Example;

public class TransientFaultHandlingOptions
{
    public bool Enabled { get; set; }
    public TimeSpan AutoRetryDelay { get; set; }
}
```

The following code builds the configuration root, binds a section to the `TransientFaultHandlingOptions` class type, and prints the bound values to the console window:

```
IConfigurationRoot configurationRoot = configuration.Build();

TransientFaultHandlingOptions options = new();
configurationRoot.GetSection(nameof(TransientFaultHandlingOptions))
    .Bind(options);

Console.WriteLine($"TransientFaultHandlingOptions.Enabled={options.Enabled}");
Console.WriteLine($"TransientFaultHandlingOptions.AutoRetryDelay={options.AutoRetryDelay}");
```

The application writes the following sample output:

```
// Sample output:
//   TransientFaultHandlingOptions.Enabled=True
//   TransientFaultHandlingOptions.AutoRetryDelay=00:00:07
```

XML configuration provider

The [XmlConfigurationProvider](#) class loads configuration from an XML file at run time. Install the [Microsoft.Extensions.Configuration.Xml](#) NuGet package.

The following code demonstrates the configuration of XML files using the XML configuration provider.

```
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.Hosting;

using IHost host = Host.CreateDefaultBuilder(args)
    .ConfigureAppConfiguration((hostingContext, configuration) =>
{
    configuration.Sources.Clear();

    configuration
        .AddXmlFile("appsettings.xml", optional: true, reloadOnChange: true)
        .AddXmlFile("repeating-example.xml", optional: true, reloadOnChange: true);

    configuration.AddEnvironmentVariables();

    if (args is { Length: > 0 })
    {
        configuration.AddCommandLine(args);
    }
})
.Build();

// Application code should start here.

await host.RunAsync();
```

The preceding code:

- Clears all existing configuration providers that were added by default in the [CreateDefaultBuilder\(String\[\]\)](#) method.
- Configures the XML configuration provider to load the *appsettings.xml* and *repeating-example.xml* files with the following options:
 - `optional: true` : The file is optional.
 - `reloadOnChange: true` : The file is reloaded when changes are saved.
- Configures the environment variables configuration provider.
- Configures the command-line configuration provider if the given `args` contain arguments.

The XML settings are overridden by settings in the [Environment variables configuration provider](#) and the [Command-line configuration provider](#).

An example *appsettings.xml* file with various configuration settings follows:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <SecretKey>Secret key value</SecretKey>
    <TransientFaultHandlingOptions>
        <Enabled>true</Enabled>
        <AutoRetryDelay>00:00:07</AutoRetryDelay>
    </TransientFaultHandlingOptions>
    <Logging>
        <LogLevel>
            <Default>Information</Default>
            <Microsoft>Warning</Microsoft>
        </LogLevel>
    </Logging>
</configuration>
```

TIP

To use the `IConfiguration` type in WinForms apps, add a reference to the [Microsoft.Extensions.Configuration.Xml](#) NuGet package. Instantiate the `ConfigurationBuilder` and chain calls to `AddXmlFile` and `Build()`. For more information, see [.NET Docs Issue #29679](#).

In .NET 5 and earlier versions, add the `name` attribute to distinguish repeating elements that use the same element name. In .NET 6 and later versions, the XML configuration provider automatically indexes repeating elements. That means you don't have to specify the `name` attribute, except if you want the "0" index in the key and there's only one element.

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <section name="section0">
    <key name="key0">value 00</key>
    <key name="key1">value 01</key>
  </section>
  <section name="section1">
    <key name="key0">value 10</key>
    <key name="key1">value 11</key>
  </section>
</configuration>
```

The following code reads the previous configuration file and displays the keys and values:

```
IConfigurationRoot configurationRoot = configuration.Build();

string key00 = "section:section0:key:key0";
string key01 = "section:section0:key:key1";
string key10 = "section:section1:key:key0";
string key11 = "section:section1:key:key1";

string val00 = configurationRoot[key00];
string val01 = configurationRoot[key01];
string val10 = configurationRoot[key10];
string val11 = configurationRoot[key11];

Console.WriteLine($"{key00} = {val00}");
Console.WriteLine($"{key01} = {val01}");
Console.WriteLine($"{key10} = {val10}");
Console.WriteLine($"{key11} = {val11}");
```

The application would write the following sample output:

```
// Sample output:
//   section:section0:key:key0 = value 00
//   section:section0:key:key1 = value 01
//   section:section1:key:key0 = value 10
//   section:section1:key:key1 = value 11
```

Attributes can be used to supply values:

```

<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    <key attribute="value" />
    <section>
        <key attribute="value" />
    </section>
</configuration>

```

The previous configuration file loads the following keys with `value`:

- `key:attribute`
- `section:key:attribute`

INI configuration provider

The [IniConfigurationProvider](#) class loads configuration from an INI file at run time. Install the

[Microsoft.Extensions.Configuration.Ini](#) NuGet package.

The following code clears all the configuration providers and adds the `IniConfigurationProvider` with two INI files as the source:

```

using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.Hosting;

using IHost host = Host.CreateDefaultBuilder(args)
    .ConfigureAppConfiguration((hostingContext, configuration) =>
{
    configuration.Sources.Clear();

    IHostEnvironment env = hostingContext.HostingEnvironment;

    configuration
        .AddIniFile("appsettings.ini", optional: true, reloadOnChange: true)
        .AddIniFile($"appsettings.{env.EnvironmentName}.ini", true, true);
})
.Build();

// Application code should start here.

await host.RunAsync();

```

An example `appsettings.ini` file with various configuration settings follows:

```

SecretKey="Secret key value"

[TransientFaultHandlingOptions]
Enabled=True
AutoRetryDelay="00:00:07"

[Logging:LogLevel]
Default=Information
Microsoft=Warning

```

The following code displays the preceding configuration settings by writing them to the console window:

```

foreach ((string key, string value) in
    configuration.Build().AsEnumerable().Where(t => t.Value is not null))
{
    Console.WriteLine($"{key}={value}");
}

```

The application would write the following sample output:

```
// Sample output:  
//   TransientFaultHandlingOptions:Enabled=True  
//   TransientFaultHandlingOptions:AutoRetryDelay=00:00:07  
//   SecretKey=Secret key value  
//   Logging:LogLevel:Microsoft=Warning  
//   Logging:LogLevel:Default=Information
```

Environment variable configuration provider

Using the default configuration, the [EnvironmentVariablesConfigurationProvider](#) loads configuration from environment variable key-value pairs after reading `appsettings.json`, `appsettings.Environment.json`, and Secret manager. Therefore, key values read from the environment override values read from `appsettings.json`, `appsettings.Environment.json`, and Secret manager.

The `:` separator doesn't work with environment variable hierarchical keys on all platforms. The double underscore (`__`) is automatically replaced by a `:` and is supported by all platforms. For example, the `:` separator is not supported by [Bash](#), but `__` is.

The following `set` commands:

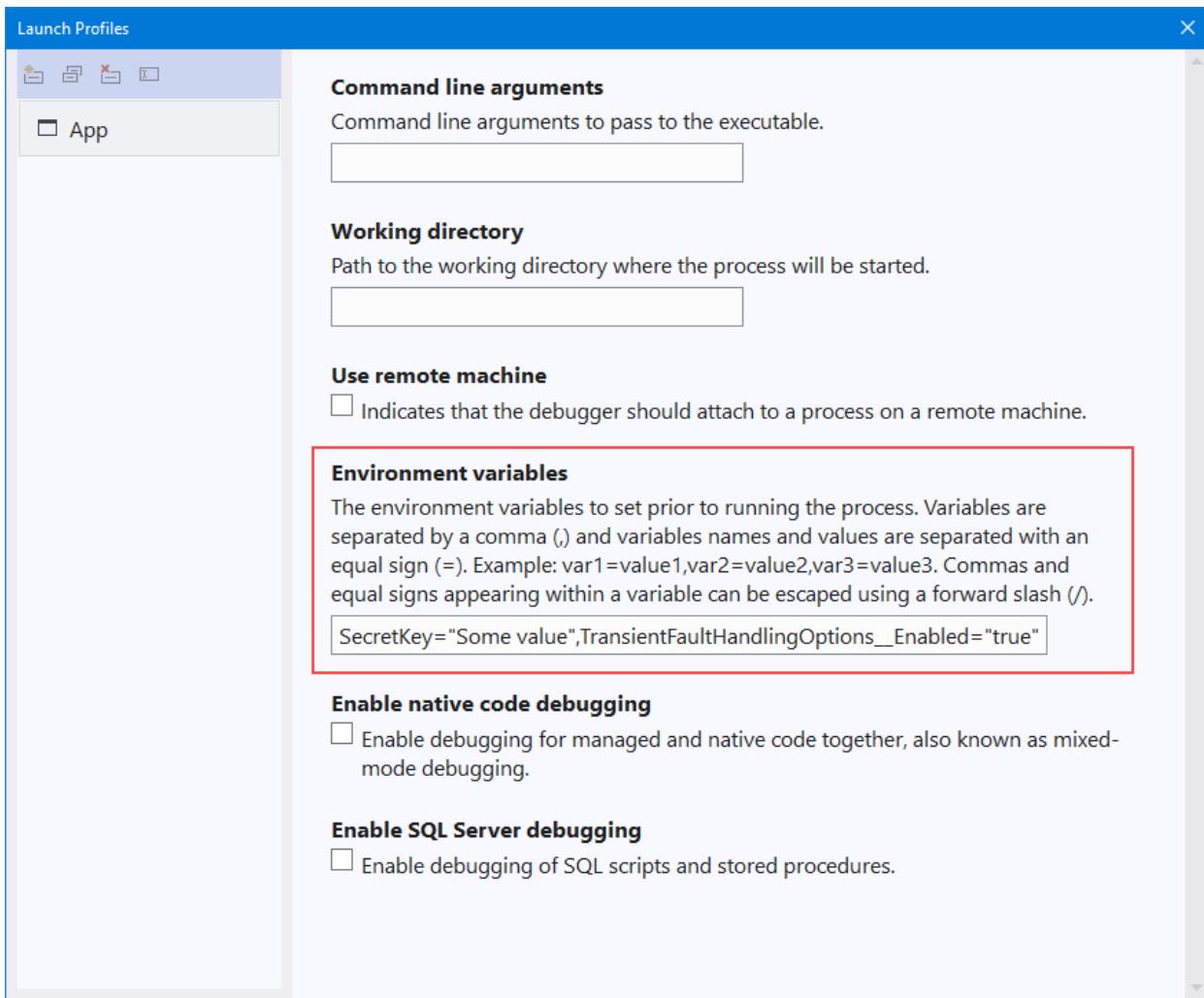
- Set the environment keys and values of the preceding example on Windows.
- Test the settings by changing them from their default values. The `dotnet run` command must be run in the project directory.

```
set SecretKey="Secret key from environment"  
set TransientFaultHandlingOptions__Enabled="true"  
set TransientFaultHandlingOptions__AutoRetryDelay="00:00:13"  
  
dotnet run
```

The preceding environment settings:

- Are only set in processes launched from the command window they were set in.
- Won't be read by web apps launched with Visual Studio.

With Visual Studio 2019 version 16.10 preview 4 and later, you can specify environment variables using the [Launch Profiles](#) dialog.



The following `setx` commands can be used to set the environment keys and values on Windows. Unlike `set`, `setx` settings are persisted. `/M` sets the variable in the system environment. If the `/M` switch isn't used, a user environment variable is set.

```
setx SecretKey "Secret key from setx environment" /M
setx TransientFaultHandlingOptions__Enabled "true" /M
setx TransientFaultHandlingOptions__AutoRetryDelay "00:00:05" /M

dotnet run
```

To test that the preceding commands override `appsettings.json` and `appsettings.Environment.json`:

- With Visual Studio: Exit and restart Visual Studio.
- With the CLI: Start a new command window and enter `dotnet run`.

Call `AddEnvironmentVariables` with a string to specify a prefix for environment variables:

```

using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.Hosting;

using IHost host = Host.CreateDefaultBuilder(args)
    .ConfigureAppConfiguration((hostingContext, configuration) =>
        configuration.AddEnvironmentVariables(
            prefix: "CustomPrefix_"))
    .Build();

// Application code should start here.

await host.RunAsync();

```

In the preceding code:

- `config.AddEnvironmentVariables(prefix: "CustomPrefix_")` is added after the default configuration providers. For an example of ordering the configuration providers, see [XML configuration provider](#).
- Environment variables set with the `CustomPrefix_` prefix override the default configuration providers. This includes environment variables without the prefix.

The prefix is stripped off when the configuration key-value pairs are read.

The following commands test the custom prefix:

```

set CustomPrefix_SecretKey="Secret key with CustomPrefix_ environment"
set CustomPrefix_TransientFaultHandlingOptions__Enabled=true
set CustomPrefix_TransientFaultHandlingOptions__AutoRetryDelay=00:00:21

dotnet run

```

The default configuration loads environment variables and command-line arguments prefixed with `DOTNET_`.

The `DOTNET_` prefix is used by .NET for [host](#) and [app configuration](#), but not for user configuration.

For more information on host and app configuration, see [.NET Generic Host](#).

On [Azure App Service](#), select **New application setting** on the **Settings > Configuration** page. Azure App Service application settings are:

- Encrypted at rest and transmitted over an encrypted channel.
- Exposed as environment variables.

Connection string prefixes

The Configuration API has special processing rules for four connection string environment variables. These connection strings are involved in configuring Azure connection strings for the app environment. Environment variables with the prefixes shown in the table are loaded into the app with the default configuration or when no prefix is supplied to `AddEnvironmentVariables`.

CONNECTION STRING PREFIX	PROVIDER
<code>CUSTOMCONNSTR_</code>	Custom provider
<code>MYSQLCONNSTR_</code>	MySQL
<code>SQLAZURECONNSTR_</code>	Azure SQL Database
<code>SQLCONNSTR_</code>	SQL Server

When an environment variable is discovered and loaded into configuration with any of the four prefixes shown in the table:

- The configuration key is created by removing the environment variable prefix and adding a configuration key section (`ConnectionStrings`).
- A new configuration key-value pair is created that represents the database connection provider (except for `CUSTOMCONNSTR_`, which has no stated provider).

ENVIRONMENT VARIABLE KEY	CONVERTED CONFIGURATION KEY	PROVIDER CONFIGURATION ENTRY
<code>CUSTOMCONNSTR_{KEY}</code>	<code>ConnectionStrings:{KEY}</code>	Configuration entry not created.
<code>MYSQLCONNSTR_{KEY}</code>	<code>ConnectionStrings:{KEY}</code>	Key: <code>ConnectionStrings:</code> <code>{KEY}_ProviderName</code> : Value: <code>MySql.Data.MySqlClient</code>
<code>SQLAZURECONNSTR_{KEY}</code>	<code>ConnectionStrings:{KEY}</code>	Key: <code>ConnectionStrings:</code> <code>{KEY}_ProviderName</code> : Value: <code>System.Data.SqlClient</code>
<code>SQLCONNSTR_{KEY}</code>	<code>ConnectionStrings:{KEY}</code>	Key: <code>ConnectionStrings:</code> <code>{KEY}_ProviderName</code> : Value: <code>System.Data.SqlClient</code>

Environment variables set in `launchSettings.json`

Environment variables set in `launchSettings.json` override those set in the system environment.

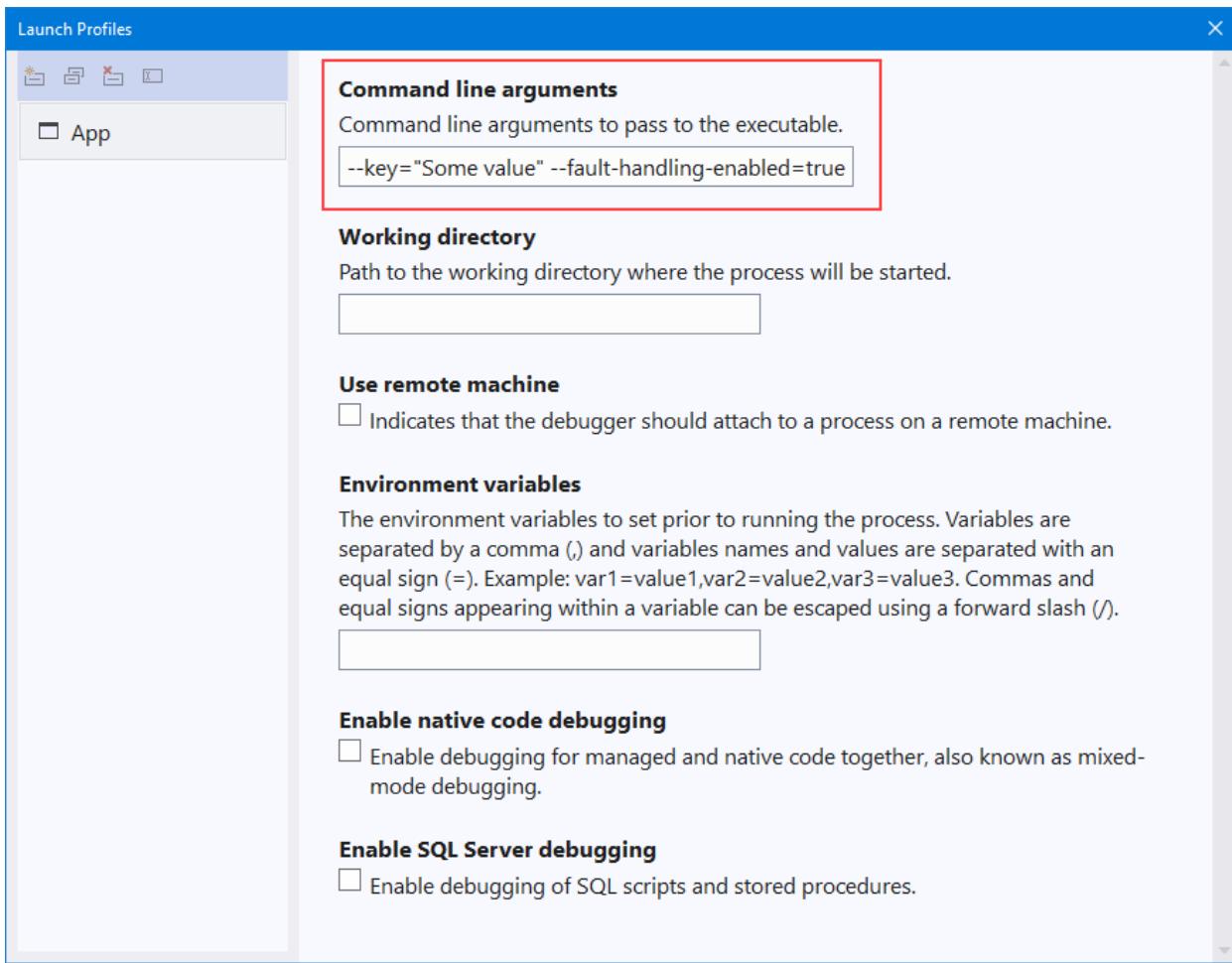
Command-line configuration provider

Using the default configuration, the [CommandLineConfigurationProvider](#) loads configuration from command-line argument key-value pairs after the following configuration sources:

- `appsettings.json` and `appsettings.Environment.json` files.
- App secrets (Secret Manager) in the `Development` environment.
- Environment variables.

By default, configuration values set on the command line override configuration values set with all the other configuration providers.

With Visual Studio 2019 version 16.10 preview 4 and later, you can specify command-line arguments using the [Launch Profiles](#) dialog.



Command-line arguments

The following command sets keys and values using `=`:

```
dotnet run SecretKey="Secret key from command line"
```

The following command sets keys and values using `/`:

```
dotnet run /SecretKey "Secret key set from forward slash"
```

The following command sets keys and values using `--`:

```
dotnet run --SecretKey "Secret key set from double hyphen"
```

The key value:

- Must follow `=`, or the key must have a prefix of `--` or `/` when the value follows a space.
- Isn't required if `=` is used. For example, `SomeKey=`.

Within the same command, don't mix command-line argument key-value pairs that use `=` with key-value pairs that use a space.

Key-per-file configuration provider

The [KeyPerFileConfigurationProvider](#) uses a directory's files as configuration key-value pairs. The key is the file name. The value is the file's contents. The Key-per-file configuration provider is used in Docker hosting scenarios.

To activate key-per-file configuration, call the [AddKeyPerFile](#) extension method on an instance of [ConfigurationBuilder](#). The `directoryPath` to the files must be an absolute path.

Overloads permit specifying:

- An `Action<KeyPerFileConfigurationSource>` delegate that configures the source.
- Whether the directory is optional and the path to the directory.

The double-underscore (`__`) is used as a configuration key delimiter in file names. For example, the file name `Logging__LogLevel__System` produces the configuration key `Logging:LogLevel:System`.

Call `ConfigureAppConfiguration` when building the host to specify the app's configuration:

```
.ConfigureAppConfiguration(_ , configuration) =>
{
    var path = Path.Combine(
        Directory.GetCurrentDirectory(), "path/to/files");

    configuration.AddKeyPerFile(directoryPath: path, optional: true);
}
```

Memory configuration provider

The [MemoryConfigurationProvider](#) uses an in-memory collection as configuration key-value pairs.

The following code adds a memory collection to the configuration system:

```
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.Hosting;

using IHost host = Host.CreateDefaultBuilder(args)
    .ConfigureAppConfiguration(_ , configuration) =>
    configuration.AddInMemoryCollection(
        new Dictionary<string, string>
    {
        ["SecretKey"] = "Dictionary MyKey Value",
        ["TransientFaultHandlingOptions:Enabled"] = bool.TrueString,
        ["TransientFaultHandlingOptions:AutoRetryDelay"] = "00:00:07",
        ["Logging:LogLevel:Default"] = "Warning"
    })
    .Build();

// Application code should start here.

await host.RunAsync();
```

In the preceding code, [MemoryConfigurationBuilderExtensions.AddInMemoryCollection\(IConfigurationBuilder, IEnumerable<KeyValuePair<String, String>>\)](#) adds the memory provider after the default configuration providers. For an example of ordering the configuration providers, see [XML configuration provider](#).

See also

- [Configuration in .NET](#)
- [.NET Generic Host](#)
- [Implement a custom configuration provider](#)

Implement a custom configuration provider in .NET

9/20/2022 • 3 minutes to read • [Edit Online](#)

There are many [configuration providers](#) available for common configuration sources such as JSON, XML, and INI files. You may need to implement a custom configuration provider when one of the available providers doesn't suit your application needs. In this article, you'll learn how to implement a custom configuration provider that relies on a database as its configuration source.

Custom configuration provider

The sample app demonstrates how to create a basic configuration provider that reads configuration key-value pairs from a database using [Entity Framework \(EF\) Core](#).

The provider has the following characteristics:

- The EF in-memory database is used for demonstration purposes.
 - To use a database that requires a connection string, get a connection string from an interim configuration.
- The provider reads a database table into configuration at startup. The provider doesn't query the database on a per-key basis.
- Reload-on-change isn't implemented, so updating the database after the app has started will not affect the app's configuration.

Define a `Settings` record type entity for storing configuration values in the database. For example, you could add a `Settings.cs` file in your `Models` folder:

```
namespace CustomProvider.Example.Models;

public record Settings(string Id, string Value);
```

For information on record types, see [Record types in C# 9](#).

Add an `EntityConfigurationContext` to store and access the configured values.

`Providers/EntityConfigurationContext.cs`:

```

using CustomProvider.Example.Models;
using Microsoft.EntityFrameworkCore;

namespace CustomProvider.Example.Providers;

public class EntityConfigurationContext : DbContext
{
    private readonly string _connectionString;

    public DbSet<Settings>? Settings { get; set; }

    public EntityConfigurationContext(string connectionString) =>
        _connectionString = connectionString;

    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        _ = _connectionString switch
        {
            { Length: > 0 } => optionsBuilder.UseSqlServer(_connectionString),
            _ => optionsBuilder.UseInMemoryDatabase("InMemoryDatabase")
        };
    }
}

```

By overriding [OnConfiguring\(DbContextOptionsBuilder\)](#) you can use the appropriate database connection. For example, if a connection string was provided you could connect to SQL Server, otherwise you could rely on an in-memory database.

Create a class that implements [IConfigurationSource](#).

Providers/EntityConfigurationSource.cs:

```

using Microsoft.Extensions.Configuration;

namespace CustomProvider.Example.Providers;

public class EntityConfigurationSource : IConfigurationSource
{
    private readonly string _connectionString;

    public EntityConfigurationSource(string connectionString) =>
        _connectionString = connectionString;

    public IConfigurationProvider Build(IConfigurationBuilder builder) =>
        new EntityConfigurationProvider(_connectionString);
}

```

Create the custom configuration provider by inheriting from [ConfigurationProvider](#). The configuration provider initializes the database when it's empty. Since configuration keys are case-insensitive, the dictionary used to initialize the database is created with the case-insensitive comparer ([StringComparer.OrdinalIgnoreCase](#)).

Providers/EntityConfigurationProvider.cs:

```

using CustomProvider.Example.Models;
using Microsoft.Extensions.Configuration;

namespace CustomProvider.Example.Providers;

public class EntityConfigurationProvider : ConfigurationProvider
{
    private readonly string _connectionString;

    public EntityConfigurationProvider(string connectionString) =>
        _connectionString = connectionString;

    public override void Load()
    {
        using var dbContext = new EntityConfigurationContext(_connectionString);

        dbContext.Database.EnsureCreated();

        Data = dbContext.Settings.Any()
            ? dbContext.Settings.ToDictionary(c => c.Id, c => c.Value, StringComparer.OrdinalIgnoreCase)
            : CreateAndSaveDefaultValues(dbContext);
    }

    static IDictionary<string, string> CreateAndSaveDefaultValues(
        EntityConfigurationContext context)
    {
        var settings = new Dictionary<string, string>(
            StringComparer.OrdinalIgnoreCase)
        {
            ["WidgetOptions:EndpointId"] = "b3da3c4c-9c4e-4411-bc4d-609e2dcc5c67",
            ["WidgetOptions:DisplayLabel"] = "Widgets Incorporated, LLC.",
            ["WidgetOptions:WidgetRoute"] = "api/widgets"
        };

        context.Settings.AddRange(
            settings.Select(kvp => new Settings(kvp.Key, kvp.Value))
                .ToArray());

        context.SaveChanges();

        return settings;
    }
}

```

An `AddEntityConfiguration` extension method permits adding the configuration source to a `IConfigurationBuilder` instance.

Extensions/ConfigurationBuilderExtensions.cs:

```

using CustomProvider.Example.Providers;

namespace Microsoft.Extensions.Configuration;

public static class ConfigurationBuilderExtensions
{
    public static IConfigurationBuilder AddEntityConfiguration(
        IConfigurationBuilder builder)
    {
        var tempConfig = builder.Build();
        var connectionString =
            tempConfig.GetConnectionString("WidgetConnectionString");

        return builder.Add(new EntityConfigurationSource(connectionString));
    }
}

```

IMPORTANT

The use of a temporary configuration source to acquire the connection string is important. The current `builder` has its configuration constructed temporarily by calling `IConfigurationBuilder.Build()`, and `GetConnectionString`. After obtaining the connection string, the `builder` adds the `EntityConfigurationSource` given the `connectionString`.

The following code shows how to use the custom `EntityConfigurationProvider` in *Program.cs*:

```

using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Options;
using CustomProvider.Example;

using IHost host = Host.CreateDefaultBuilder(args)
    .ConfigureAppConfiguration((_, configuration) =>
    {
        configuration.Sources.Clear();
        configuration.AddEntityConfiguration();
    })
    .ConfigureServices((context, services) =>
        services.Configure<WidgetOptions>(
            context.Configuration.GetSection("WidgetOptions")))
    .Build();

var options = host.Services.GetRequiredService<IOptions<WidgetOptions>>().Value;
Console.WriteLine($"DisplayLabel={options.DisplayLabel}");
Console.WriteLine($"EndpointId={options.EndpointId}");
Console.WriteLine($"WidgetRoute={options.WidgetRoute}");

await host.RunAsync();
// Sample output:
//   WidgetRoute=api/widgets
//   EndpointId=b3da3c4c-9c4e-4411-bc4d-609e2dcc5c67
//   DisplayLabel=Widgets Incorporated, LLC.

```

Consume provider

To consume the custom configuration provider, you can use the [options pattern](#). With the sample app in place, define an options object to represent the widget settings.

```

namespace CustomProvider.Example;

public class WidgetOptions
{
    public Guid EndpointId { get; set; }

    public string DisplayName { get; set; } = null!;

    public string WidgetRoute { get; set; } = null!;
}

```

A call to [ConfigureServices](#) configures the mapping of the options.

```

using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Options;
using CustomProvider.Example;

using IHost host = Host.CreateDefaultBuilder(args)
    .ConfigureAppConfiguration((_, configuration) =>
    {
        configuration.Sources.Clear();
        configuration.AddEntityConfiguration();
    })
    .ConfigureServices((context, services) =>
        services.Configure<WidgetOptions>(
            context.Configuration.GetSection("WidgetOptions")))
    .Build();

var options = host.Services.GetRequiredService<IOptions<WidgetOptions>>().Value;
Console.WriteLine($"DisplayName={options.DisplayName}");
Console.WriteLine($"EndpointId={options.EndpointId}");
Console.WriteLine($"WidgetRoute={options.WidgetRoute}");

await host.RunAsync();
// Sample output:
//   WidgetRoute=api/widgets
//   EndpointId=b3da3c4c-9c4e-4411-bc4d-609e2dcc5c67
//   DisplayName=Widgets Incorporated, LLC.

```

The preceding code configures the `WidgetOptions` object from the `"WidgetOptions"` section of the configuration. This enables the options pattern, exposing a dependency injection-ready `IOptions<WidgetOptions>` representation of the EF settings. The options are ultimately provided from the custom configuration provider.

See also

- [Configuration in .NET](#)
- [Configuration providers in .NET](#)
- [Options pattern in .NET](#)
- [Dependency injection in .NET](#)

Options pattern in .NET

9/20/2022 • 11 minutes to read • [Edit Online](#)

The options pattern uses classes to provide strongly-typed access to groups of related settings. When [configuration settings](#) are isolated by scenario into separate classes, the app adheres to two important software engineering principles:

- The [Interface Segregation Principle \(ISP\) or Encapsulation](#): Scenarios (classes) that depend on configuration settings depend only on the configuration settings that they use.
- [Separation of Concerns](#): Settings for different parts of the app aren't dependent or coupled with one another.

Options also provide a mechanism to validate configuration data. For more information, see the [Options validation](#) section.

Bind hierarchical configuration

The preferred way to read related configuration values is using the options pattern. The options pattern is possible through the `IOptions<TOptions>` interface, where the generic type parameter `TOptions` is constrained to a `class`. The `IOptions<TOptions>` can later be provided through dependency injection. For more information, see [Dependency injection in .NET](#).

For example, to read the highlighted configuration values from an `appsettings.json` file:

```
{  
    "SecretKey": "Secret key value",  
    "TransientFaultHandlingOptions": {  
        "Enabled": true,  
        "AutoRetryDelay": "00:00:07"  
    },  
    "Logging": {  
        "LogLevel": {  
            "Default": "Information",  
            "Microsoft": "Warning",  
            "Microsoft.Hosting.Lifetime": "Information"  
        }  
    }  
}
```

Create the following `TransientFaultHandlingOptions` class:

```
public class TransientFaultHandlingOptions  
{  
    public bool Enabled { get; set; }  
    public TimeSpan AutoRetryDelay { get; set; }  
}
```

When using the options pattern, an options class:

- Must be non-abstract with a public parameterless constructor
- Contain public read-write properties to bind (fields are *not* bound)

The following code is part of the `Program.cs` C# file and:

- Calls `ConfigurationBinder.Bind` to bind the `TransientFaultHandlingOptions` class to the

"TransientFaultHandlingOptions" section.

- Displays the configuration data.

```
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.Hosting;
using ConsoleJson.Example;

IHost host = Host.CreateDefaultBuilder(args)
    .ConfigureAppConfiguration((hostingContext, configuration) =>
{
    configuration.Sources.Clear();

    IHostEnvironment env = hostingContext.HostingEnvironment;

    configuration
        .AddJsonFile("appsettings.json", optional: true, reloadOnChange: true)
        .AddJsonFile($"appsettings.{env.EnvironmentName}.json", true, true);

    IConfigurationRoot configurationRoot = configuration.Build();

    TransientFaultHandlingOptions options = new();
    configurationRoot.GetSection(nameof(TransientFaultHandlingOptions))
        .Bind(options);

    Console.WriteLine($"TransientFaultHandlingOptions.Enabled={options.Enabled}");
    Console.WriteLine($"TransientFaultHandlingOptions.AutoRetryDelay={options.AutoRetryDelay}");
})
.Build();

// Application code should start here.

await host.RunAsync();
```

In the preceding code, changes to the JSON configuration file after the app has started are read.

`ConfigurationBinder.Get<T>` binds and returns the specified type. `ConfigurationBinder.Get<T>` maybe more convenient than using `ConfigurationBinder.Bind`. The following code shows how to use `ConfigurationBinder.Get<T>` with the `TransientFaultHandlingOptions` class:

```
IConfigurationRoot configurationRoot = configuration.Build();

var options =
    configurationRoot.GetSection(nameof(TransientFaultHandlingOptions))
        .Get<TransientFaultHandlingOptions>();

Console.WriteLine($"TransientFaultHandlingOptions.Enabled={options.Enabled}");
Console.WriteLine($"TransientFaultHandlingOptions.AutoRetryDelay={options.AutoRetryDelay}");
```

In the preceding code, changes to the JSON configuration file after the app has started are read.

IMPORTANT

The `ConfigurationBinder` class exposes several APIs, such as `.Bind(object instance)` and `.Get<T>()` that are *not* constrained to `class`. When using any of the `Options interfaces`, you must adhere to aforementioned `options class constraints`.

An alternative approach when using the options pattern is to bind the "TransientFaultHandlingOptions" section and add it to the `dependency injection service container`. In the following code, `TransientFaultHandlingOptions` is added to the service container with `Configure` and bound to configuration:

```
services.Configure<TransientFaultHandlingOptions>(
    configurationRoot.GetSection(
        key: nameof(TransientFaultHandlingOptions)));
```

To access both the `services` and the `configurationRoot` objects, you must use the [ConfigureServices](#) method — the `IConfiguration` is available as the [HostBuilderContext.Configuration](#) property.

```
Host.CreateDefaultBuilder(args)
    .ConfigureServices((context, services) =>
{
    var configurationRoot = context.Configuration;
    services.Configure<TransientFaultHandlingOptions>(
        configurationRoot.GetSection(nameof(TransientFaultHandlingOptions)));
});
```

TIP

The `key` parameter is the name of the configuration section to search for. It does *not* have to match the name of the type that represents it. For example, you could have a section named `"FaultHandling"` and it could be represented by the `TransientFaultHandlingOptions` class. In this instance, you'd pass `"FaultHandling"` to the `GetSection` function instead. The `nameof` operator is used as a convenience when the named section matches the type it corresponds to.

Using the preceding code, the following code reads the position options:

```
using Microsoft.Extensions.Options;

namespace ConsoleJson.Example;

public class ExampleService
{
    private readonly TransientFaultHandlingOptions _options;

    public ExampleService(IOptions<TransientFaultHandlingOptions> options) =>
        _options = options.Value;

    public void DisplayValues()
    {
        Console.WriteLine($"TransientFaultHandlingOptions.Enabled={_options.Enabled}");
        Console.WriteLine($"TransientFaultHandlingOptions.AutoRetryDelay={_options.AutoRetryDelay}");
    }
}
```

In the preceding code, changes to the JSON configuration file after the app has started are *not* read. To read changes after the app has started, use [IOptionsSnapshot](#).

Options interfaces

[IOptions<TOptions>](#):

- Does *not* support:
 - Reading of configuration data after the app has started.
 - [Named options](#)
- Is registered as a [Singleton](#) and can be injected into any [service lifetime](#).

[IOptionsSnapshot<TOptions>](#):

- Is useful in scenarios where options should be recomputed on every injection resolution, in [scoped](#) or

[transient lifetimes](#). For more information, see [Use IOptionsSnapshot to read updated data](#).

- Is registered as [Scoped](#) and therefore cannot be injected into a Singleton service.
- Supports [named options](#)

IOptionsMonitor<TOptions>:

- Is used to retrieve options and manage options notifications for [TOptions](#) instances.
- Is registered as a [Singleton](#) and can be injected into any [service lifetime](#).
- Supports:
 - Change notifications
 - [Named options](#)
 - [Reloadable configuration](#)
 - Selective options invalidation ([IOptionsMonitorCache<TOptions>](#))

[IOptionsFactory<TOptions>](#) is responsible for creating new options instances. It has a single [Create](#) method. The default implementation takes all registered [IConfigureOptions<TOptions>](#) and [IPostConfigureOptions<TOptions>](#) and runs all the configurations first, followed by the post-configuration. It distinguishes between [IConfigureNamedOptions<TOptions>](#) and [IConfigureOptions<TOptions>](#) and only calls the appropriate interface.

[IOptionsMonitorCache<TOptions>](#) is used by [IOptionsMonitor<TOptions>](#) to cache [TOptions](#) instances. The [IOptionsMonitorCache<TOptions>](#) invalidates options instances in the monitor so that the value is recomputed ([TryRemove](#)). Values can be manually introduced with [TryAdd](#). The [Clear](#) method is used when all named instances should be recreated on demand.

Options interfaces benefits

Using a generic wrapper type gives you the ability to decouple the lifetime of the option from the DI container. The [IOptions<TOptions>.Value](#) interface provides a layer of abstraction, including generic constraints, on your options type. This provides the following benefits:

- The evaluation of the [T](#) configuration instance is deferred to the accessing of [IOptions<TOptions>.Value](#), rather than when it is injected. This is important because you can consume the [T](#) option from various places and choose the lifetime semantics without changing anything about [T](#).
- When registering options of type [T](#), you do not need to explicitly register the [T](#) type. This is a convenience when you're [authoring a library](#) with simple defaults, and you don't want to force the caller to register options into the DI container with a specific lifetime.
- From the perspective of the API, it allows for constraints on the type [T](#) (in this case, [T](#) is constrained to a reference type).

Use IOptionsSnapshot to read updated data

When you use [IOptionsSnapshot<TOptions>](#), options are computed once per request when accessed and are cached for the lifetime of the request. Changes to the configuration are read after the app starts when using configuration providers that support reading updated configuration values.

The difference between [IOptionsMonitor](#) and [IOptionsSnapshot](#) is that:

- [IoptionsMonitor](#) is a [singleton service](#) that retrieves current option values at any time, which is especially useful in singleton dependencies.
- [IoptionsSnapshot](#) is a [scoped service](#) and provides a snapshot of the options at the time the [IoptionsSnapshot<T>](#) object is constructed. Options snapshots are designed for use with transient and scoped dependencies.

The following code uses [IOptionsSnapshot<TOptions>](#).

```

using Microsoft.Extensions.Options;

namespace ConsoleJson.Example;

public class ScopedService
{
    private readonly TransientFaultHandlingOptions _options;

    public ScopedService(IOptionsSnapshot<TransientFaultHandlingOptions> options) =>
        _options = options.Value;

    public void DisplayValues()
    {
        Console.WriteLine($"TransientFaultHandlingOptions.Enabled={_options.Enabled}");
        Console.WriteLine($"TransientFaultHandlingOptions.AutoRetryDelay={_options.AutoRetryDelay}");
    }
}

```

The following code registers a configuration instance which `TransientFaultHandlingOptions` binds against:

```

services.Configure<TransientFaultHandlingOptions>(
    configurationRoot.GetSection(
        nameof(TransientFaultHandlingOptions)));

```

In the preceding code, changes to the JSON configuration file after the app has started are read.

IOptionsMonitor

The following code registers a configuration instance which `TransientFaultHandlingOptions` binds against.

```

services.Configure<TransientFaultHandlingOptions>(
    configurationRoot.GetSection(
        nameof(TransientFaultHandlingOptions)));

```

The following example uses `IOptionsMonitor<TOptions>`:

```

using Microsoft.Extensions.Options;

namespace ConsoleJson.Example;

public class MonitorService
{
    private readonly IOptionsMonitor<TransientFaultHandlingOptions> _monitor;

    public MonitorService(IOptionsMonitor<TransientFaultHandlingOptions> monitor) =>
        _monitor = monitor;

    public void DisplayValues()
    {
        TransientFaultHandlingOptions options = _monitor.CurrentValue;

        Console.WriteLine($"TransientFaultHandlingOptions.Enabled={options.Enabled}");
        Console.WriteLine($"TransientFaultHandlingOptions.AutoRetryDelay={options.AutoRetryDelay}");
    }
}

```

In the preceding code, changes to the JSON configuration file after the app has started are read.

TIP

Some file systems, such as Docker containers and network shares, may not reliably send change notifications. When using the `IOptionsMonitor<TOptions>` interface in these environments, set the `DOTNET_USE_POLLING_FILE_WATCHER` environment variable to `1` or `true` to poll the file system for changes. The interval at which changes are polled is every four seconds and is not configurable.

For more information on Docker containers, see [Containerize a .NET app](#).

Named options support using `IConfigureNamedOptions`

Named options:

- Are useful when multiple configuration sections bind to the same properties.
- Are case-sensitive.

Consider the following `appsettings.json` file:

```
{  
  "Features": {  
    "Personalize": {  
      "Enabled": true,  
      "ApiKey": "aGEgaGEgeW91IHRob3VnaHQgdGhhCB3YXMgcmVhbGx5IHNvbWV0aGluZw=="  
    },  
    "WeatherStation": {  
      "Enabled": true,  
      "ApiKey": "QXJlIHlvdSBhdHR1bXB0aW5nIHRvIGhhY2sgdXM/"  
    }  
  }  
}
```

Rather than creating two classes to bind `Features:Personalize` and `Features:WeatherStation`, the following class is used for each section:

```
public class Features  
{  
  public const string Personalize = nameof(Personalize);  
  public const string WeatherStation = nameof(WeatherStation);  
  
  public bool Enabled { get; set; }  
  public string ApiKey { get; set; }  
}
```

The following code configures the named options:

```
ConfigureServices(services =>  
{  
  services.Configure<Features>(  
    Features.Personalize,  
    Configuration.GetSection("Features:Personalize"));  
  
  services.Configure<Features>(  
    Features.WeatherStation,  
    Configuration.GetSection("Features:WeatherStation"));  
});
```

The following code displays the named options:

```

public class Service
{
    private readonly Features _personalizeFeature;
    private readonly Features _weatherStationFeature;

    public Service(IOptionsSnapshot<Features> namedOptionsAccessor)
    {
        _personalizeFeature = namedOptionsAccessor.Get(Features.Personalize);
        _weatherStationFeature = namedOptionsAccessor.Get(Features.WeatherStation);
    }
}

```

All options are named instances. `IConfigureOptions<TOptions>` instances are treated as targeting the `Options.DefaultName` instance, which is `string.Empty`. `IConfigureNamedOptions<TOptions>` also implements `IConfigureOptions<TOptions>`. The default implementation of the `IOptionsFactory<TOptions>` has logic to use each appropriately. The `null` named option is used to target all of the named instances instead of a specific named instance. [ConfigureAll](#) and [PostConfigureAll](#) use this convention.

OptionsBuilder API

`OptionsBuilder<TOptions>` is used to configure `TOptions` instances. `OptionsBuilder` streamlines creating named options as it's only a single parameter to the initial `AddOptions<TOptions>(string optionsName)` call instead of appearing in all of the subsequent calls. Options validation and the `ConfigureOptions` overloads that accept service dependencies are only available via `OptionsBuilder`.

`OptionsBuilder` is used in the [Options validation](#) section.

Use DI services to configure options

Services can be accessed from dependency injection while configuring options in two ways:

- Pass a configuration delegate to `Configure` on `OptionsBuilder<TOptions>`. `OptionsBuilder<TOptions>` provides overloads of `Configure` that allow use of up to five services to configure options:

```

services.AddOptions<MyOptions>("optionalName")
    .Configure<ExampleService, ScopedService, MonitorService>(
        (options, es, ss, ms) =>
            options.Property = DoSomethingWith(es, ss, ms));

```

- Create a type that implements `IConfigureOptions<TOptions>` or `IConfigureNamedOptions<TOptions>` and register the type as a service.

We recommend passing a configuration delegate to `Configure`, since creating a service is more complex. Creating a type is equivalent to what the framework does when calling `Configure`. Calling `Configure` registers a transient generic `IConfigureNamedOptions<TOptions>`, which has a constructor that accepts the generic service types specified.

Options validation

Options validation enables option values to be validated.

Consider the following `appsettings.json` file:

```
{
    "MyCustomSettingsSection": {
        "SiteTitle": "Amazing docs from Awesome people!",
        "Scale": 10,
        "VerbosityLevel": 32
    }
}
```

The following class binds to the `"MyCustomSettingsSection"` configuration section and applies a couple of `DataAnnotations` rules:

```
using System.ComponentModel.DataAnnotations;

namespace ConsoleJson.Example;

public class SettingsOptions
{
    public const string ConfigurationSectionName = "MyCustomSettingsSection";

    [RegularExpression(@"^[\w\W'-\s]{1,40}$")]
    public string SiteTitle { get; set; } = null!;

    [Range(0, 1000,
        ErrorMessage = "Value for {0} must be between {1} and {2}.")]
    public int Scale { get; set; }

    public int VerbosityLevel { get; set; }
}
```

In the preceding `SettingsOptions` class, the `ConfigurationSectionName` property contains the name of the configuration section to bind to. In this scenario, the options object provides the name of its configuration section.

TIP

The configuration section name is independent of the configuration object that it's binding to. In other words, a configuration section named `"FooBarOptions"` can be bound to an options object named `ZedOptions`. Although it might be common to name them the same, it's *not* necessary and can actually cause name conflicts.

The following code:

- Calls `AddOptions` to get an `OptionsResolver<TOptions>` that binds to the `SettingsOptions` class.
- Calls `ValidateDataAnnotations` to enable validation using `DataAnnotations`.

```
services.AddOptions<SettingsOptions>()
    .Bind(Configuration.GetSection(SettingsOptions.ConfigurationSectionName))
    .ValidateDataAnnotations();
```

The `ValidateDataAnnotations` extension method is defined in the [Microsoft.Extensions.Options.DataAnnotations](#) NuGet package.

The following code displays the configuration values or the validation errors:

```

using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Options;

namespace ConsoleJson.Example;

public class ValidationService
{
    private readonly ILogger<ValidationService> _logger;
    private readonly IOptions<SettingsOptions> _config;

    public ValidationService(
        ILogger<ValidationService> logger,
        IOptions<SettingsOptions> config)
    {
        _config = config;
        _logger = logger;

        try
        {
            SettingsOptions options = _config.Value;
        }
        catch (OptionsValidationException ex)
        {
            foreach (string failure in ex.Failures)
            {
                _logger.LogError(failure);
            }
        }
    }
}

```

The following code applies a more complex validation rule using a delegate:

```

services.AddOptions<SettingsOptions>()
    .Bind(Configuration.GetSection(SettingsOptions.ConfigurationSectionName))
    .ValidateDataAnnotations()
    .Validate(config =>
    {
        if (config.Scale != 0)
        {
            return config.verbosityLevel > config.Scale;
        }

        return true;
    }, "VerbosityLevel must be > than Scale.");

```

IValidateOptions for complex validation

The following class implements [IValidateOptions<TOptions>](#):

```

using System.Text;
using System.Text.RegularExpressions;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.Options;

namespace ConsoleJson.Example;

class ValidateSettingsOptions : IValidateOptions<SettingsOptions>
{
    public SettingsOptions _settings { get; private set; }

    public ValidateSettingsOptions(IConfiguration config) =>
        _settings = config.GetSection(SettingsOptions.ConfigurationSectionName)
            .Get<SettingsOptions>();

    public ValidateOptionsResult Validate(string name, SettingsOptions options)
    {
        StringBuilder failure = new();
        var rx = new Regex(@"^a-zA-Z'-'s]{1,40}$");
        Match match = rx.Match(options.SiteTitle);
        if (string.IsNullOrEmpty(match.Value))
        {
            failure.AppendLine($"{options.SiteTitle} doesn't match RegEx");
        }

        if (options.Scale < 0 || options.Scale > 1000)
        {
            failure.AppendLine($"{options.Scale} isn't within Range 0 - 1000");
        }

        if (_settings.Scale is 0 && _settings.VerboseLevel <= _settings.Scale)
        {
            failure.AppendLine("VerboseLevel must be > than Scale.");
        }

        return failure.Length > 0
            ? ValidateOptionsResult.Fail(failure.ToString())
            : ValidateOptionsResult.Success;
    }
}

```

`IValidateOptions` enables moving the validation code into a class.

NOTE

This example code relies on the [Microsoft.Extensions.Configuration.Json](#) NuGet package.

Using the preceding code, validation is enabled in `ConfigureServices` with the following code:

```

services.Configure<SettingsOptions>(
    Configuration.GetSection(
        SettingsOptions.ConfigurationSectionName));
services.TryAddEnumerable(
    ServiceDescriptor.Singleton
        <IValidateOptions<SettingsOptions>, ValidateSettingsOptions>());

```

Options post-configuration

Set post-configuration with [IPostConfigureOptions<TOptions>](#). Post-configuration runs after all [IConfigureOptions<TOptions>](#) configuration occurs, and can be useful in scenarios when you need to override configuration:

```
services.PostConfigure<CustomOptions>(customOptions =>
{
    customOptions.Option1 = "post_configured_option1_value";
});
```

[PostConfigure](#) is available to post-configure named options:

```
services.PostConfigure<CustomOptions>("named_options_1", customOptions =>
{
    customOptions.Option1 = "post_configured_option1_value";
});
```

Use [PostConfigureAll](#) to post-configure all configuration instances:

```
services.PostConfigureAll<CustomOptions>(customOptions =>
{
    customOptions.Option1 = "post_configured_option1_value";
});
```

See also

- [Configuration in .NET](#)
- [Options pattern guidance for .NET library authors](#)

Options pattern guidance for .NET library authors

9/20/2022 • 5 minutes to read • [Edit Online](#)

With the help of dependency injection, registering your services and their corresponding configurations can make use of the *options pattern*. The options pattern enables consumers of your library (and your services) to require instances of [options interfaces](#) where `TOptions` is your options class. Consuming configuration options through strongly-typed objects helps to ensure consistent value representation, and removes the burden of manually parsing string values. There are many [configuration providers](#) for consumers of your library to use. With these providers, consumers can configure your library in many ways.

As a .NET library author, you'll learn general guidance on how to correctly expose the options pattern to consumers of your library. There are various ways to achieve the same thing, and several considerations to make.

Naming conventions

By convention, extension methods responsible for registering services are named `Add{Service}`, where `{Service}` is a meaningful and descriptive name. `Add{Service}` extension methods are commonplace in [ASP.NET Core](#).

- ✓ CONSIDER names that disambiguate your service from other offerings.
- ✗ DO NOT use names that are already part of the .NET ecosystem from official Microsoft packages.
- ✓ CONSIDER naming static classes that expose extension methods as `{Type}Extensions`, where `{Type}` is the type that you're extending.

Namespace guidance

Microsoft packages make use of the `Microsoft.Extensions.DependencyInjection` namespace to unify the registration of various service offerings.

- ✓ CONSIDER a namespace that clearly identifies your package offering.
- ✗ DO NOT use the `Microsoft.Extensions.DependencyInjection` namespace for non-official Microsoft packages.

Parameterless

If your service can work with minimal or no explicit configuration, consider a parameterless extension method.

```

using Microsoft.Extensions.DependencyInjection;

namespace ExampleLibrary.Extensions.DependencyInjection;

public static class ServiceCollectionExtensions
{
    public static IServiceCollection AddMyLibraryService(
        this IServiceCollection services)
    {
        services.AddOptions<LibraryOptions>()
            .Configure(options =>
        {
            // Specify default option values
        });

        // Register lib services here...
        // services.AddScoped<ILibraryService, DefaultLibraryService>();

        return services;
    }
}

```

In the preceding code, the `AddMyLibraryService`:

- Extends an instance of `IServiceCollection`
- Calls `OptionsServiceCollectionExtensions.AddOptions<TOptions>(IServiceCollection)` with the type parameter of `LibraryOptions`
- Chains a call to `Configure`, which specifies the default option values

IConfiguration parameter

When you author a library that exposes many options to consumers, you may want to consider requiring an `IConfiguration` parameter extension method. The expected `IConfiguration` instance should be scoped to a named section of the configuration by using the `IConfiguration.GetSection` function.

```

using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;

namespace ExampleLibrary.Extensions.DependencyInjection;

public static class ServiceCollectionExtensions
{
    public static IServiceCollection AddMyLibraryService(
        this IServiceCollection services,
        IConfiguration namedConfigurationSection)
    {
        // Default library options are overridden
        // by bound configuration values.
        services.Configure<LibraryOptions>(namedConfigurationSection);

        // Register lib services here...
        // services.AddScoped<ILibraryService, DefaultLibraryService>();

        return services;
    }
}

```

TIP

The `Configure<TOptions>(IServiceCollection, IConfiguration)` method is part of the `Microsoft.Extensions.Options.ConfigurationExtensions` NuGet package.

In the preceding code, the `AddMyLibraryService`:

- Extends an instance of `IServiceCollection`
- Defines an `IConfiguration` parameter `namedConfigurationSection`
- Calls `Configure<TOptions>(IServiceCollection, IConfiguration)` passing the generic type parameter of `LibraryOptions` and the `namedConfigurationSection` instance to configure

Consumers in this pattern provide the scoped `IConfiguration` instance of the named section:

```
using ExampleLibrary.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;

namespace Options.ConfigParam;

class Program
{
    static async Task Main(string[] args)
    {
        using IHost host = CreateHostBuilder(args).Build();

        // Application code should start here.

        await host.RunAsync();
    }

    static IHostBuilder CreateHostBuilder(string[] args) =>
        Host.CreateDefaultBuilder(args)
            .ConfigureServices((context, services) =>
            {
                services.AddMyLibraryService(
                    context.Configuration.GetSection("LibraryOptions"));
            });
    }
}
```

The call to `.AddMyLibraryService` is made in the `ConfigureServices` method. The same is true when using a `Startup` class, the addition of services being registered occurs in `ConfigureServices`.

As the library author, specifying default values is up to you.

NOTE

It is possible to bind configuration to an options instance. However, there is a risk of name collisions - which will cause errors. Additionally, when manually binding in this way, you limit the consumption of your options pattern to read-once. Changes to settings will not be re-bound, as such consumers will not be able to use the `IOptionsMonitor` interface.

```
services.AddOptions<LibraryOptions>()
    .Configure<IConfiguration>(
        (options, configuration) =>
            configuration.GetSection("LibraryOptions").Bind(options));
```

Action<TOptions> parameter

Consumers of your library may be interested in providing a lambda expression that yields an instance of your options class. In this scenario, you define an `Action<LibraryOptions>` parameter in your extension method.

```
using Microsoft.Extensions.DependencyInjection;

namespace ExampleLibrary.Extensions.DependencyInjection;

public static class ServiceCollectionExtensions
{
    public static IServiceCollection AddMyLibraryService(
        this IServiceCollection services,
        Action<LibraryOptions> configureOptions)
    {
        services.Configure(configureOptions);

        // Register lib services here...
        // services.AddScoped<ILibraryService, DefaultLibraryService>();

        return services;
    }
}
```

In the preceding code, the `AddMyLibraryService`:

- Extends an instance of `IServiceCollection`
- Defines an `Action<T>` parameter `configureOptions` where `T` is `LibraryOptions`
- Calls `Configure` given the `configureOptions` action

Consumers in this pattern provide a lambda expression (or a delegate that satisfies the `Action<LibraryOptions>` parameter):

```
using ExampleLibrary.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;

namespace Options.Action;

class Program
{
    static async Task Main(string[] args)
    {
        using IHost host = CreateHostBuilder(args).Build();

        // Application code should start here.

        await host.RunAsync();
    }

    static IHostBuilder CreateHostBuilder(string[] args) =>
        Host.CreateDefaultBuilder(args)
            .ConfigureServices(services =>
            {
                services.AddMyLibraryService(options =>
                {
                    // User defined option values
                    // options.SomePropertyValue = ...
                });
            });
}
```

Options instance parameter

Consumers of your library might prefer to provide an inlined options instance. In this scenario, you expose an extension method that takes an instance of your options object, `LibraryOptions`.

```
using Microsoft.Extensions.DependencyInjection;

namespace ExampleLibrary.Extensions.DependencyInjection;

public static class ServiceCollectionExtensions
{
    public static IServiceCollection AddMyLibraryService(
        this IServiceCollection services,
        LibraryOptions userOptions)
    {
        services.AddOptions<LibraryOptions>()
            .Configure(options =>
        {
            // Overwrite default option values
            // with the user provided options.
            // options.SomeValue = userOptions.SomeValue;
        });

        // Register lib services here...
        // services.AddScoped<ILibraryService, DefaultLibraryService>();

        return services;
    }
}
```

In the preceding code, the `AddMyLibraryService`:

- Extends an instance of `IServiceCollection`
- Calls `OptionsServiceCollectionExtensions.AddOptions<TOptions>(IServiceCollection)` with the type parameter of `LibraryOptions`
- Chains a call to `Configure`, which specifies default option values that can be overridden from the given `userOptions` instance

Consumers in this pattern provide an instance of the `LibraryOptions` class, defining desired property values inline:

```

using ExampleLibrary.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;

namespace Options.Object;

class Program
{
    static async Task Main(string[] args)
    {
        using IHost host = CreateHostBuilder(args).Build();

        // Application code should start here.

        await host.RunAsync();
    }

    static IHostBuilder CreateHostBuilder(string[] args) =>
        Host.CreateDefaultBuilder(args)
            .ConfigureServices(services =>
        {
            services.AddMyLibraryService(new LibraryOptions
            {
                // Specify option values
                // SomePropertyValue = ...
            });
        });
    }
}

```

Post configuration

After all configuration option values are bound or specified, post configuration functionality is available.

Exposing the same `Action<TOptions>` parameter detailed earlier, you could choose to call `PostConfigure`. Post configure runs after all `.Configure` calls.

```

using Microsoft.Extensions.DependencyInjection;

namespace ExampleLibrary.Extensions.DependencyInjection;

public static class ServiceCollectionExtensions
{
    public static IServiceCollection AddMyLibraryService(
        this IServiceCollection services,
        Action<LibraryOptions> configureOptions)
    {
        services.PostConfigure(configureOptions);

        // Register lib services here...
        // services.AddScoped<ILibraryService, DefaultLibraryService>();

        return services;
    }
}

```

In the preceding code, the `AddMyLibraryService`:

- Extends an instance of `IServiceCollection`
- Defines an `Action<T>` parameter `configureOptions` where `T` is `LibraryOptions`
- Calls `PostConfigure` given the `configureOptions` action

Consumers in this pattern provide a lambda expression (or a delegate that satisfies the `Action<LibraryOptions>` parameter), just as they would with the `Action<TOptions>` parameter in a non-post configuration scenario:

```
using ExampleLibrary.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;

namespace Options.PostConfig;

class Program
{
    static async Task Main(string[] args)
    {
        using IHost host = CreateHostBuilder(args).Build();

        // Application code should start here.

        await host.RunAsync();
    }

    static IHostBuilder CreateHostBuilder(string[] args) =>
        Host.CreateDefaultBuilder(args)
            .ConfigureServices(services =>
            {
                services.AddMyLibraryService(options =>
                {
                    // Specify option values
                    // options.SomePropertyValue = ...
                });
            });
    }
}
```

See also

- [Options pattern in .NET](#)
- [Dependency injection in .NET](#)
- [Dependency injection guidelines](#)

Logging in .NET

9/20/2022 • 17 minutes to read • [Edit Online](#)

.NET supports a logging API that works with a variety of built-in and third-party logging providers. This article shows how to use the logging API with built-in providers. Most of the code examples shown in this article apply to any .NET app that uses the [Generic Host](#). For apps that don't use the Generic Host, see [Non-host console app](#).

TIP

All of the logging example source code is available in the [Samples Browser](#) for download. For more information, see [Browse code samples: Logging in .NET](#).

IMPORTANT

Starting with .NET 6, logging services no longer register the `ILogger` type. When using a logger, specify the generic-type alternative `ILogger<TCategoryName>` or register the `ILogger` with [dependency injection \(DI\)](#).

Create logs

To create logs, use an `ILogger<TCategoryName>` object from [DI](#).

The following example:

- Creates a logger, `ILogger<Worker>`, which uses a log *category* of the fully qualified name of the type `Worker`. The log category is a string that is associated with each log.
- Calls `LogInformation` to log at the `Information` level. The Log *level* indicates the severity of the logged event.

```
public class Worker : BackgroundService
{
    private readonly ILogger<Worker> _logger;

    public Worker(ILogger<Worker> logger) =>
        _logger = logger;

    protected override async Task ExecuteAsync(CancellationToken stoppingToken)
    {
        while (!stoppingToken.IsCancellationRequested)
        {
            _logger.LogInformation("Worker running at: {time}", DateTimeOffset.UtcNow);
            await Task.Delay(1000, stoppingToken);
        }
    }
}
```

Levels and [categories](#) are explained in more detail later in this article.

Configure logging

Logging configuration is commonly provided by the `Logging` section of `appsettings.{Environment}.json` files. The following `appsettings.Development.json` file is generated by the .NET Worker service templates:

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  }
}
```

In the preceding JSON:

- The `"Default"`, `"Microsoft"`, and `"Microsoft.Hosting.Lifetime"` log level categories are specified.
- The `"Default"` value is applied to all categories that aren't otherwise specified, effectively making all default values for all categories `"Information"`. You can override this behavior by specifying a value for a category.
- The `"Microsoft"` category applies to all categories that start with `"Microsoft"`.
- The `"Microsoft"` category logs at a log level of `Warning` and higher.
- The `"Microsoft.Hosting.Lifetime"` category is more specific than the `"Microsoft"` category, so the `"Microsoft.Hosting.Lifetime"` category logs at log level `"Information"` and higher.
- A specific log provider is not specified, so `LogLevel` applies to all the enabled logging providers except for the [Windows EventLog](#).

The `Logging` property can have [LogLevel](#) and log provider properties. The `LogLevel` specifies the minimum [level](#) to log for selected categories. In the preceding JSON, `Information` and `Warning` log levels are specified. `LogLevel` indicates the severity of the log and ranges from 0 to 6:

`Trace` = 0, `Debug` = 1, `Information` = 2, `Warning` = 3, `Error` = 4, `Critical` = 5, and `None` = 6.

When a `LogLevel` is specified, logging is enabled for messages at the specified level and higher. In the preceding JSON, the `Default` category is logged for `Information` and higher. For example, `Information`, `Warning`, `Error`, and `critical` messages are logged. If no `LogLevel` is specified, logging defaults to the `Information` level. For more information, see [Log levels](#).

A provider property can specify a `LogLevel` property. `LogLevel` under a provider specifies levels to log for that provider, and overrides the non-provider log settings. Consider the following `appsettings.json` file:

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Error",
      "Microsoft": "Warning"
    },
    "Debug": {
      "LogLevel": {
        "Default": "Information",
        "Microsoft.Hosting": "Trace"
      }
    },
    "EventSource": {
      "LogLevel": {
        "Default": "Warning"
      }
    }
  }
}
```

Settings in `Logging.{ProviderName}.LogLevel` override settings in `Logging.LogLevel`. In the preceding JSON, the `Debug` provider's default log level is set to `Information`:

```
Logging:Debug:LogLevel:Default:Information
```

The preceding setting specifies the `Information` log level for every `Logging:Debug:` category except `Microsoft.Hosting`. When a specific category is listed, the specific category overrides the default category. In the preceding JSON, the `Logging:Debug:LogLevel` categories `"Microsoft.Hosting"` and `"Default"` override the settings in `Logging:LogLevel`.

The minimum log level can be specified for any of:

- Specific providers: For example, `Logging:EventSource:LogLevel:Default:Information`
- Specific categories: For example, `Logging:LogLevel:Microsoft:Warning`
- All providers and all categories: `Logging:LogLevel:Default:Warning`

Any logs below the minimum level are *not*:

- Passed to the provider.
- Logged or displayed.

To suppress all logs, specify `LogLevel.None`. `LogLevel.None` has a value of 6, which is higher than `LogLevel.Critical` (5).

If a provider supports [log scopes](#), `IncludeScopes` indicates whether they're enabled. For more information, see [log scopes](#)

The following `appsettings.json` file contains settings for all of the built-in providers:

```
{
    "Logging": {
        "LogLevel": {
            "Default": "Error",
            "Microsoft": "Warning",
            "Microsoft.Hosting.Lifetime": "Warning"
        },
        "Debug": {
            "LogLevel": {
                "Default": "Information"
            }
        },
        "Console": {
            "IncludeScopes": true,
            "LogLevel": {
                "Microsoft.Extensions.Hosting": "Warning",
                "Default": "Information"
            }
        },
        "EventSource": {
            "LogLevel": {
                "Microsoft": "Information"
            }
        },
        "EventLog": {
            "LogLevel": {
                "Microsoft": "Information"
            }
        },
        "AzureAppServicesFile": {
            "IncludeScopes": true,
            "LogLevel": {
                "Default": "Warning"
            }
        },
        "AzureAppServicesBlob": {
            "IncludeScopes": true,
            "LogLevel": {
                "Microsoft": "Information"
            }
        },
        "ApplicationInsights": {
            "LogLevel": {
                "Default": "Information"
            }
        }
    }
}
```

In the preceding sample:

- The categories and levels are not suggested values. The sample is provided to show all the default providers.
- Settings in `Logging.{ProviderName}.LogLevel` override settings in `Logging.LogLevel`. For example, the level in `Debug.LogLevel.Default` overrides the level in `LogLevel.Default`.
- Each provider's *alias* is used. Each provider defines an *alias* that can be used in configuration in place of the fully qualified type name. The built-in providers' aliases are:
 - Console
 - Debug
 - EventSource
 - EventLog
 - AzureAppServicesFile
 - AzureAppServicesBlob

- ApplicationInsights

Set log level by command line, environment variables, and other configuration

Log level can be set by any of the [configuration providers](#). For example, you can create a persisted environment variable named `Logging:LogLevel:Microsoft` with a value of `Information`.

- [Command Line](#)
- [PowerShell](#)
- [Bash](#)

Create and assign persisted environment variable, given the log level value.

```
:: Assigns the env var to the value  
setx "Logging__LogLevel__Microsoft" "Information" /M
```

In a *new* instance of the **Command Prompt**, read the environment variable.

```
:: Prints the env var value  
echo %Logging__LogLevel__Microsoft%
```

The preceding environment setting is persisted in the environment. To test the settings when using an app created with the .NET Worker service templates, use the `dotnet run` command in the project directory after the environment variable is assigned.

```
dotnet run
```

TIP

After setting an environment variable, restart your integrated development environment (IDE) to ensure that newly added environment variables are available.

On [Azure App Service](#), select **New application setting** on the **Settings > Configuration** page. Azure App Service application settings are:

- Encrypted at rest and transmitted over an encrypted channel.
- Exposed as environment variables.

For more information on setting .NET configuration values using environment variables, see [environment variables](#).

How filtering rules are applied

When an `ILogger<TCategoryName>` object is created, the `ILoggerFactory` object selects a single rule per provider to apply to that logger. All messages written by an `ILogger` instance are filtered based on the selected rules. The most specific rule for each provider and category pair is selected from the available rules.

The following algorithm is used for each provider when an `ILogger` is created for a given category:

- Select all rules that match the provider or its alias. If no match is found, select all rules with an empty provider.
- From the result of the preceding step, select rules with longest matching category prefix. If no match is found, select all rules that don't specify a category.
- If multiple rules are selected, take the last one.

- If no rules are selected, use `LoggingBuilderExtensions.SetMinimumLevel(ILoggingBuilder, LogLevel)` to specify the minimum logging level.

Log category

When an `ILogger` object is created, a *category* is specified. That category is included with each log message created by that instance of `ILogger`. The category string is arbitrary, but the convention is to use the class name. For example, in an application with a service defined like the following object, the category might be `"Example.DefaultService"`:

```
namespace Example
{
    public class DefaultService : IService
    {
        private readonly ILogger<DefaultService> _logger;

        public DefaultService(ILogger<DefaultService> logger) =>
            _logger = logger;

        // ...
    }
}
```

To explicitly specify the category, call `LoggerFactory.CreateLogger`:

```
namespace Example
{
    public class DefaultService : IService
    {
        private readonly ILogger _logger;

        public DefaultService	ILoggerFactory loggerFactory) =>
            _logger = loggerFactory.CreateLogger("CustomCategory");

        // ...
    }
}
```

Calling `CreateLogger` with a fixed name can be useful when used in multiple classes/types so the events can be organized by category.

`ILogger<T>` is equivalent to calling `CreateLogger` with the fully qualified type name of `T`.

Log level

The following table lists the `LogLevel` values, the convenience `Log{LogLevel}` extension method, and the suggested usage:

LOGLEVEL	VALUE	METHOD	DESCRIPTION
Trace	0	<code>LogTrace</code>	Contain the most detailed messages. These messages may contain sensitive app data. These messages are disabled by default and should <i>not</i> be enabled in production.

LOGLEVEL	VALUE	METHOD	DESCRIPTION
Debug	1	LogDebug	For debugging and development. Use with caution in production due to the high volume.
Information	2	LogInformation	Tracks the general flow of the app. May have long-term value.
Warning	3	.LogWarning	For abnormal or unexpected events. Typically includes errors or conditions that don't cause the app to fail.
Error	4	.LogError	For errors and exceptions that cannot be handled. These messages indicate a failure in the current operation or request, not an app-wide failure.
Critical	5	LogCritical	For failures that require immediate attention. Examples: data loss scenarios, out of disk space.
None	6		Specifies that no messages should be written.

In the previous table, the `LogLevel` is listed from lowest to highest severity.

The `Log` method's first parameter, `LogLevel`, indicates the severity of the log. Rather than calling `Log(LogLevel, ...)`, most developers call the `Log{LogLevel}` extension methods. The `Log{LogLevel}` extension methods [call the Log method and specify the LogLevel](#). For example, the following two logging calls are functionally equivalent and produce the same log:

```
public void LogDetails()
{
    var logMessage = "Details for log.";

    _logger.Log(LogLevel.Information, AppLogEvents.Details, logMessage);
    _logger.LogInformation(AppLogEvents.Details, logMessage);
}
```

`AppLogEvents.Details` is the event ID, and is implicitly represented by a constant `Int32` value. `AppLogEvents` is a class that exposes various named identifier constants and is displayed in the [Log event ID](#) section.

The following code creates `Information` and `Warning` logs:

```

public async Task<T> GetAsync<T>(string id)
{
    _logger.LogInformation(AppLogEvents.Read, "Reading value for {Id}", id);

    var result = await _repository.GetAsync(id);
    if (result is null)
    {
        _logger.LogWarning(AppLogEvents.ReadNotFound, "GetAsync({Id}) not found", id);
    }

    return result;
}

```

In the preceding code, the first `Log{LogLevel}` parameter, `AppLogEvents.Read`, is the [Log event ID](#). The second parameter is a message template with placeholders for argument values provided by the remaining method parameters. The method parameters are explained in the [message template](#) section later in this article.

Configure the appropriate log level and call the correct `Log{LogLevel}` methods to control how much log output is written to a particular storage medium. For example:

- In production:
 - Logging at the `Trace` or `Debug` levels produces a high-volume of detailed log messages. To control costs and not exceed data storage limits, log `Trace` and `Debug` level messages to a high-volume, low-cost data store. Consider limiting `Trace` and `Debug` to specific categories.
 - Logging at `Warning` through `Critical` levels should produce few log messages.
 - Costs and storage limits usually aren't a concern.
 - Few logs allow more flexibility in data store choices.
- In development:
 - Set to `Warning`.
 - Add `Trace` or `Debug` messages when troubleshooting. To limit output, set `Trace` or `Debug` only for the categories under investigation.

The following JSON sets `Logging:Console:LogLevel:Microsoft:Information`:

```
{
  "Logging": {
    "LogLevel": {
      "Microsoft": "Warning"
    },
    "Console": {
      "LogLevel": {
        "Microsoft": "Information"
      }
    }
  }
}
```

Log event ID

Each log can specify an *event identifier*, the [EventId](#) is a structure with an `Id` and optional `Name` readonly properties. The sample source code uses the `AppLogEvents` class to define event IDs:

```
using Microsoft.Extensions.Logging;

internal static class AppLogEvents
{
    internal EventId Create = new(1000, "Created");
    internal EventId Read = new(1001, "Read");
    internal EventId Update = new(1002, "Updated");
    internal EventId Delete = new(1003, "Deleted");

    // These are also valid EventId instances, as there's
    // an implicit conversion from int to an EventId
    internal const int Details = 3000;
    internal const int Error = 3001;

    internal EventId ReadNotFound = 4000;
    internal EventId UpdateNotFound = 4001;

    // ...
}
```

TIP

For more information on converting an `int` to an `EventId`, see [EventId.Implicit\(Int32 to EventId\) Operator](#).

An event ID associates a set of events. For example, all logs related to reading values from a repository might be `1001`.

The logging provider may log the event ID in an ID field, in the logging message, or not at all. The Debug provider doesn't show event IDs. The console provider shows event IDs in brackets after the category:

```
info: Example.DefaultService.GetAsync[1001]
      Reading value for a1b2c3
warn: Example.DefaultService.GetAsync[4000]
      GetAsync(a1b2c3) not found
```

Some logging providers store the event ID in a field, which allows for filtering on the ID.

Log message template

Each log API uses a message template. The message template can contain placeholders for which arguments are provided. Use names for the placeholders, not numbers. The order of placeholders, not their names, determines which parameters are used to provide their values. In the following code, the parameter names are out of sequence in the message template:

```
string p1 = "param1";
string p2 = "param2";
_logger.LogInformation("Parameter values: {p2}, {p1}", p1, p2);
```

The preceding code creates a log message with the parameter values in sequence:

```
Parameter values: param1, param2
```

NOTE

Be mindful when using multiple placeholders within a single message template, as they're ordinal-based. The names are *not* used to align the arguments to the placeholders.

This approach allows logging providers to implement [semantic or structured logging](#). The arguments themselves are passed to the logging system, not just the formatted message template. This enables logging providers to store the parameter values as fields. Consider the following logger method:

```
_logger.LogInformation("Getting item {Id} at {RunTime}", id, DateTime.Now);
```

For example, when logging to Azure Table Storage:

- Each Azure Table entity can have `ID` and `RunTime` properties.
- Tables with properties simplify queries on logged data. For example, a query can find all logs within a particular `RunTime` range without having to parse the time out of the text message.

Log message template formatting

Log message templates support placeholder formatting. Templates are free to specify [any valid format](#) for the given type argument. For example, consider the following `Information` logger message template:

```
_logger.LogInformation("Logged on {PlaceHolderName:MMMM dd, yyyy}", DateTimeOffset.UtcNow);
// Logged on January 06, 2022
```

In the preceding example, the `DateTimeOffset` instance is the type that corresponds to the `PlaceHolderName` in the logger message template. This name can be anything as the values are ordinal-based. The `MMMM dd, yyyy` format is valid for the `DateTimeOffset` type.

For more information on `DateTime` and `DateTimeOffset` formatting, see [Custom date and time format strings](#).

Log message template formatting examples

Log message templates allow placeholder formatting. The following examples show how to format a message template using the `{}` placeholder syntax. Additionally, an example of escaping the `{}` placeholder syntax is shown with its output. Finally, string interpolation with templating placeholders is also shown:

```
logger.LogInformation("Number: {Number}", 1);           // Number: 1
logger.LogInformation("{Number}: {Number}", 3);         // {Number}: 3
logger.LogInformation($"{{{Number}}}": {{Number}}", 5); // {Number}: 5
```

TIP

You should always use log message template formatting when logging. You should avoid string interpolation as it can cause performance issues.

Log exceptions

The logger methods have overloads that take an exception parameter:

```

public void Test(string id)
{
    try
    {
        if (id == "none")
        {
            throw new Exception("Default Id detected.");
        }
    }
    catch (Exception ex)
    {
        _logger.LogWarning(
            AppLogEvents.Error, ex,
            "Failed to process iteration: {Id}", id);
    }
}

```

Exception logging is provider-specific.

Default log level

If the default log level is not set, the default log level value is `Information`.

For example, consider the following worker service app:

- Created with the .NET Worker templates.
- `appsettings.json` and `appsettings.Development.json` deleted or renamed.

With the preceding setup, navigating to the privacy or home page produces many `Trace`, `Debug`, and `Information` messages with `Microsoft` in the category name.

The following code sets the default log level when the default log level is not set in configuration:

```

class Program
{
    static Task Main(string[] args) =>
        CreateHostBuilder(args).Build().RunAsync();

    static IHostBuilder CreateHostBuilder(string[] args) =>
        Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging => logging.SetMinimumLevel(LogLevel.Warning));
}

```

Filter function

A filter function is invoked for all providers and categories that don't have rules assigned to them by configuration or code:

```

await Host.CreateDefaultBuilder(args)
    .ConfigureLogging(logging =>
        logging.AddFilter((provider, category, logLevel) =>
    {
        return provider.Contains("ConsoleLoggerProvider")
            && (category.Contains("Example") || category.Contains("Microsoft"))
            && logLevel >= LogLevel.Information;
    }))
    .Build()
    .RunAsync();

```

The preceding code displays console logs when the category contains `Example` or `Microsoft` and the log level is `Information` or higher.

Log scopes

A *scope* can group a set of logical operations. This grouping can be used to attach the same data to each log that's created as part of a set. For example, every log created as part of processing a transaction can include the transaction ID.

A scope:

- Is an [IDisposable](#) type that's returned by the [BeginScope](#) method.
- Lasts until it's disposed.

The following providers support scopes:

- [Console](#)
- [AzureAppServicesFile](#) and [AzureAppServicesBlob](#)

Use a scope by wrapping logger calls in a `using` block:

```
public async Task<T> GetAsync<T>(string id)
{
    T result;

    using (_logger.BeginScope("using block message"))
    {
        _logger.LogInformation(
            AppLogEvents.Read, "Reading value for {Id}", id);

        var result = await _repository.GetAsync(id);
        if (result is null)
        {
            _logger.LogWarning(
                AppLogEvents.ReadNotFound, "GetAsync({Id}) not found", id);
        }
    }

    return result;
}
```

The following JSON enables scopes for the console provider:

```
{
    "Logging": {
        "Debug": {
            "LogLevel": {
                "Default": "Information"
            }
        },
        "Console": {
            "IncludeScopes": true,
            "LogLevel": {
                "Microsoft": "Warning",
                "Default": "Information"
            }
        },
        "LogLevel": {
            "Default": "Debug"
        }
    }
}
```

The following code enables scopes for the console provider:

```
await Host.CreateDefaultBuilder(args)
    .ConfigureLogging((_, logging) =>
        logging.ClearProviders()
            .AddConsole(options => options.IncludeScopes = true))
    .Build()
    .RunAsync();
```

Non-host console app

Logging code for apps without a [Generic Host](#) differs in the way [providers are added](#) and [loggers are created](#). In a non-host console app, call the provider's `Add{provider name}` extension method while creating a `LoggerFactory`:

```
using var loggerFactory = LoggerFactory.Create(builder =>
{
    builder
        .AddFilter("Microsoft", LogLevel.Warning)
        .AddFilter("System", LogLevel.Warning)
        .AddFilter("LoggingConsoleApp.Program", LogLevel.Debug)
        .AddConsole();
});

ILogger logger = loggerFactory.CreateLogger<Program>();
logger.LogInformation("Example log message");
```

The `loggerFactory` object is used to create an [ILogger](#) instance.

Create logs in Main

The following code logs in `Main` by getting an `ILogger` instance from DI after building the host:

```
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;

IHost host = Host.CreateDefaultBuilder(args).Build();

var logger = host.Services.GetRequiredService<ILogger<Program>>();
logger.LogInformation("Host created.");

await host.RunAsync();
```

The preceding code relies on two NuGet packages:

- [Microsoft.Extensions.Hosting](#)
- [Microsoft.Extensions.Logging](#)

Its project file would look similar to the following:

```

<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>net6.0</TargetFramework>
    <ImplicitUsings>enable</ImplicitUsings>
    <Nullable>enable</Nullable>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.Extensions.Hosting" Version="6.0.1" />
    <PackageReference Include="Microsoft.Extensions.Logging" Version="6.0.0" />
  </ItemGroup>

</Project>

```

No asynchronous logger methods

Logging should be so fast that it isn't worth the performance cost of asynchronous code. If a logging datastore is slow, don't write to it directly. Consider writing the log messages to a fast store initially, then moving them to the slow store later. For example, when logging to SQL Server, don't do so directly in a `Log` method, since the `Log` methods are synchronous. Instead, synchronously add log messages to an in-memory queue and have a background worker pull the messages out of the queue to do the asynchronous work of pushing data to SQL Server.

Change log levels in a running app

The Logging API doesn't include a scenario to change log levels while an app is running. However, some configuration providers are capable of reloading configuration, which takes immediate effect on logging configuration. For example, the [File Configuration Provider](#) reloads logging configuration by default. If configuration is changed in code while an app is running, the app can call [IConfigurationRoot.Reload](#) to update the app's logging configuration.

NuGet packages

The `ILogger<TCategoriesName>` and `ILoggerFactory` interfaces and implementations are included in the .NET SDK. They are also available in the following NuGet packages:

- The interfaces are in [Microsoft.Extensions.Logging.Abstractions](#).
- The default implementations are in [Microsoft.Extensions.Logging](#).

Apply log filter rules in code

The preferred approach for setting log filter rules is by using [Configuration](#).

The following example shows how to register filter rules in code:

```

await Host.CreateDefaultBuilder(args)
    .ConfigureLogging(logging =>
        logging.AddFilter("System", LogLevel.Debug)
            .AddFilter<DebugLoggerProvider>("Microsoft", LogLevel.Information)
            .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
    .Build()
    .RunAsync();

```

`logging.AddFilter("System", LogLevel.Debug)` specifies the `System` category and log level `Debug`. The filter is applied to all providers because a specific provider was not configured.

`AddFilter<DebugLoggerProvider>("Microsoft", LogLevel.Information)` specifies:

- The `Debug` logging provider.
- Log level `Information` and higher.
- All categories starting with `"Microsoft"`.

See also

- [Logging providers in .NET](#)
- [Implement a custom logging provider in .NET](#)
- [Console log formatting](#)
- [High-performance logging in .NET](#)
- Logging bugs should be created in the github.com/dotnet/runtime repo

Logging providers in .NET

9/20/2022 • 5 minutes to read • [Edit Online](#)

Logging providers persist logs, except for the `Console` provider, which only displays logs as standard output. For example, the Azure Application Insights provider stores logs in Azure Application Insights. Multiple providers can be enabled.

The default .NET Worker app templates:

- Use the [Generic Host](#).
- Call `CreateDefaultBuilder`, which adds the following logging providers:
 - `Console`
 - `Debug`
 - `EventSource`
 - `EventLog` (Windows only)

```
using Microsoft.Extensions.Hosting;

using IHost host = Host.CreateDefaultBuilder(args).Build();

// Application code should start here.

await host.RunAsync();
```

The preceding code shows the `Program` class created with the .NET Worker app templates. The next several sections provide samples based on the .NET Worker app templates, which use the Generic Host.

To override the default set of logging providers added by `Host.CreateDefaultBuilder`, call `ClearProviders` and add the logging providers you want. For example, the following code:

- Calls `ClearProviders` to remove all the `IProvider` instances from the builder.
- Adds the `Console` logging provider.

```
static IHostBuilder CreateHostBuilder(string[] args) =>
    Host.CreateDefaultBuilder(args)
        .ConfigureLogging(logging =>
    {
        logging.ClearProviders();
        logging.AddConsole();
    });
});
```

For additional providers, see:

- [Built-in logging providers](#).
- [Third-party logging providers](#).

Configure a service that depends on `ILogger`

To configure a service that depends on `ILogger<T>`, use constructor injection or provide a factory method. The factory method approach is recommended only if there's no other option. For example, consider a service that needs an `ILogger<T>` instance provided by DI:

```
.ConfigureServices(services =>
    services.AddSingleton<IExampleService>(container =>
        new DefaultExampleService
    {
        Logger = container.GetRequiredService<ILogger<IExampleService>>()
    }));
});
```

The preceding code is a `Func<IServiceProvider, IExampleService>` that runs the first time the DI container needs to construct an instance of `IExampleService`. You can access any of the registered services in this way.

Built-in logging providers

Microsoft Extensions include the following logging providers as part of the runtime libraries:

- [Console](#)
- [Debug](#)
- [EventSource](#)
- [EventLog](#)

The following logging providers are shipped by Microsoft, but not as part of the runtime libraries. They must be installed as additional NuGet packages.

- [AzureAppServicesFile](#) and [AzureAppServicesBlob](#)
- [ApplicationInsights](#)

Console

The `Console` provider logs output to the console.

Debug

The `Debug` provider writes log output by using the `System.Diagnostics.Debug` class, specifically through the `Debug.WriteLine` method and only when the debugger is attached. The `DebugLoggerProvider` creates `DebugLogger` instances, which are implementations of the `ILogger` interface.

Event Source

The `EventSource` provider writes to a cross-platform event source with the name `Microsoft.Extensions.Logging`. On Windows, the provider uses `ETW`.

dotnet trace tooling

The `dotnet-trace` tool is a cross-platform CLI global tool that enables the collection of .NET Core traces of a running process. The tool collects `Microsoft.Extensions.Logging.EventSource` provider data using a `LoggingEventSource`.

See [dotnet-trace](#) for installation instructions. For a diagnostic tutorial using `dotnet-trace`, see [Debug high CPU usage in .NET Core](#).

Windows EventLog

The `EventLog` provider sends log output to the Windows Event Log. Unlike the other providers, the `EventLog` provider does *not* inherit the default non-provider settings. If `EventLog` log settings aren't specified, they default to `LogLevel.Warning`.

To log events lower than `LogLevel.Warning`, explicitly set the log level. The following example sets the Event Log default log level to `LogLevel.Information`:

```
"Logging": {  
    "EventLog": {  
        "LogLevel": {  
            "Default": "Information"  
        }  
    }  
}
```

[AddEventLog overloads](#) can pass in [EventLogSettings](#). If `null` or not specified, the following default settings are used:

- `LogName` : "Application"
- `SourceName` : ".NET Runtime"
- `MachineName` : The local machine name is used.

The following code changes the `SourceName` from the default value of ".NET Runtime" to `CustomLogs`:

```
public class Program  
{  
    static async Task Main(string[] args)  
    {  
        using IHost host = CreateHostBuilder(args).Build();  
  
        // Application code should start here.  
  
        await host.RunAsync();  
    }  
  
    static IHostBuilder CreateHostBuilder(string[] args) =>  
        Host.CreateDefaultBuilder(args)  
            .ConfigureLogging(logging =>  
                logging.AddEventLog(configuration =>  
                    configuration.SourceName = "CustomLogs"));  
    }  
}
```

Azure App Service

The [Microsoft.Extensions.Logging.AzureAppServices](#) provider package writes logs to text files in an Azure App Service app's file system and to [blob storage](#) in an Azure Storage account.

The provider package isn't included in the runtime libraries. To use the provider, add the provider package to the project.

To configure provider settings, use [AzureFileLoggerOptions](#) and [AzureBlobLoggerOptions](#), as shown in the following example:

```

using Microsoft.Extensions.Logging.AzureAppServices;

using IHost host = Host.CreateDefaultBuilder(args)
    .ConfigureLogging(logging =>
        logging.AddAzureWebAppDiagnostics())
    .ConfigureServices(services =>
        services.Configure<AzureFileLoggerOptions>(options =>
    {
        options.FileName = "azure-diagnostics-";
        options.FileSizeLimit = 50 * 1024;
        options.RetainedFileCountLimit = 5;
    })
    .Configure<AzureBlobLoggerOptions>(options =>
    {
        options.BlobName = "log.txt";
    }));
Build();

// Application code should start here.

await host.RunAsync();

```

When deployed to Azure App Service, the app uses the settings in the [App Service logs](#) section of the [App Service](#) page of the Azure portal. When the following settings are updated, the changes take effect immediately without requiring a restart or redeployment of the app.

The default location for log files is in the *D:\home\LogFiles\Application* folder. Additional defaults vary by provider:

- **Application Logging (Filesystem)**: The default filesystem file name is *diagnostics-yyyymmdd.txt*. The default file size limit is 10 MB, and the default maximum number of files retained is 2.
- **Application Logging (Blob)**: The default blob name is *{app-name}/yyyy/mm/dd/hh/{guid}_applicationLog.txt*.

This provider only logs when the project runs in the Azure environment.

Azure log streaming

Azure log streaming supports viewing log activity in real-time from:

- The app server
- The web server
- Failed request tracing

To configure Azure log streaming:

- Navigate to the [App Service logs](#) page from the app's portal page.
- Set **Application Logging (Filesystem)** to **On**.
- Choose the **log Level**. This setting only applies to Azure log streaming.

Navigate to the **Log Stream** page to view logs. The logged messages are logged with the `ILogger` interface.

Azure Application Insights

The [Microsoft.Extensions.Logging.ApplicationInsights](#) provider package writes logs to [Azure Application Insights](#). Application Insights is a service that monitors a web app and provides tools for querying and analyzing the telemetry data. If you use this provider, you can query and analyze your logs by using the Application Insights tools.

For more information, see the following resources:

- [Application Insights overview](#)

- [ApplicationInsightsLoggerProvider](#) for .NET Core `ILogger` logs - Start here if you want to implement the logging provider without the rest of Application Insights telemetry.
- [Application Insights logging adapters](#).
- [Install, configure, and initialize the Application Insights SDK](#) - Interactive tutorial on the Microsoft Learn site.

Logging provider design considerations

If you plan to develop your own implementation of the `ILoggerProvider` interface and corresponding custom implementation of `ILogger`, consider the following points:

- The `ILogger.Log` method is synchronous.
- The lifetime of log state and objects should *not* be assumed.

An implementation of `ILoggerProvider` will create an `ILogger` via its `ILoggerProvider.CreateLogger` method. If your implementation strives to queue logging messages in a non-blocking manner, the messages should first be materialized or the object state that's used to materialize a log entry should be serialized. Doing so avoids potential exceptions from disposed objects.

For more information, see [Implement a custom logging provider in .NET](#).

Third-party logging providers

Here are some third-party logging frameworks that work with various .NET workloads:

- [elmah.io](#) ([GitHub repo](#))
- [Gelf](#) ([GitHub repo](#))
- [JSONLog](#) ([GitHub repo](#))
- [KissLog.net](#) ([GitHub repo](#))
- [Log4Net](#) ([GitHub repo](#))
- [NLog](#) ([GitHub repo](#))
- [NReco.Logging](#) ([GitHub repo](#))
- [Sentry](#) ([GitHub repo](#))
- [Serilog](#) ([GitHub repo](#))
- [Stackdriver](#) ([GitHub repo](#))

Some third-party frameworks can perform [semantic logging, also known as structured logging](#).

Using a third-party framework is similar to using one of the built-in providers:

1. Add a NuGet package to your project.
2. Call an `ILoggerFactory` or `ILoggingBuilder` extension method provided by the logging framework.

For more information, see each provider's documentation. Third-party logging providers aren't supported by Microsoft.

See also

- [Logging in .NET](#).
- [Implement a custom logging provider in .NET](#).
- [High-performance logging in .NET](#).

Compile-time logging source generation

9/20/2022 • 7 minutes to read • [Edit Online](#)

.NET 6 introduces the `LoggerMessageAttribute` type. This attribute is part of the `Microsoft.Extensions.Logging` namespace, and when used, it source-generates performant logging APIs. The source-generation logging support is designed to deliver a highly usable and highly performant logging solution for modern .NET applications. The auto-generated source code relies on the `ILogger` interface in conjunction with `LoggerMessage.Define` functionality.

The source generator is triggered when `LoggerMessageAttribute` is used on `partial` logging methods. When triggered, it is either able to autogenerate the implementation of the `partial` methods it's decorating, or produce compile-time diagnostics with hints about proper usage. The compile-time logging solution is typically considerably faster at run time than existing logging approaches. It achieves this by eliminating boxing, temporary allocations, and copies to the maximum extent possible.

Basic usage

To use the `LoggerMessageAttribute`, the consuming class and method need to be `partial`. The code generator is triggered at compile time, and generates an implementation of the `partial` method.

```
public static partial class Log
{
    [LoggerMessage(
        EventId = 0,
        Level = LogLevel.Critical,
        Message = "Could not open socket to `{hostName}`")]
    public static partial void CouldNotOpenSocket(
        ILogger logger, string hostName);
}
```

In the preceding example, the logging method is `static` and the log level is specified in the attribute definition. When using the attribute in a static context, the `ILogger` instance is required as a parameter. You may choose to use the attribute in a non-static context as well. Consider the following example where the logging method is declared as an instance method. In this context, the logging method gets the logger by accessing an `ILogger` field in the containing class.

```
public partial class InstanceLoggingExample
{
    private readonly ILogger _logger;

    public InstanceLoggingExample(ILogger logger)
    {
        _logger = logger;
    }

    [LoggerMessage(
        EventId = 0,
        Level = LogLevel.Critical,
        Message = "Could not open socket to `{hostName}`")]
    public partial void CouldNotOpenSocket(string hostName);
}
```

Sometimes, the log level needs to be dynamic rather than statically built into the code. You can do this by

omitting the log level from the attribute and instead requiring it as a parameter to the logging method.

```
public static partial class Log
{
    [LoggerMessage(
        EventId = 0,
        Message = "Could not open socket to `{hostName}`")]
    public static partial void CouldNotOpenSocket(
        ILogger logger,
        LogLevel level, /* Dynamic log level as parameter, rather than defined in attribute. */
        string hostName);
}
```

You can omit the logging message and `String.Empty` will be provided for the message. The state will contain the arguments, formatted as key-value pairs.

```
using System.Text.Json;
using Microsoft.Extensions.Logging;

	ILogger<SampleObject> logger = LoggerFactory.Create(
	builder =>
	builder.AddJsonConsole(
		options =>
		options.JsonWriterOptions = new JsonWriterOptions()
		{
			Indented = true
		}))
.CreateLogger<SampleObject>();

logger.CustomLogEvent(LogLevel.Information, "Liana", "California");

public static partial class SampleObject
{
    [LoggerMessage(EventId = 23)]
    public static partial void CustomLogEvent(
        this ILogger logger, LogLevel logLevel,
        string name, string state);
}
```

Consider the example logging output when using the `JsonConsole` formatter.

```
{
    "EventId": 23,
    "LogLevel": "Information",
    "Category": "ConsoleApp.SampleObject",
    "Message": "",
    "State": {
        "Message": "",
        "name": "Liana",
        "state": "California",
        "{OriginalFormat}": ""
    }
}
```

Log method constraints

When using the `LoggerMessageAttribute` on logging methods, there are some constraints that must be followed:

- Logging methods must be `partial` and return `void`.
- Logging method names must *not* start with an underscore.

- Parameter names of logging methods must *not* start with an underscore.
- Logging methods may *not* be defined in a nested type.
- Logging methods *cannot* be generic.
- If a logging method is `static`, the `ILogger` instance is required as a parameter.

The code-generation model depends on code being compiled with a modern C# compiler, version 9 or later. The C# 9.0 compiler became available with .NET 5. To upgrade to a modern C# compiler, edit your project file to target C# 9.0.

```
<PropertyGroup>
  <LangVersion>9.0</LangVersion>
</PropertyGroup>
```

For more information, see [C# language versioning](#).

Log method anatomy

The `ILogger.Log` signature accepts the `LogLevel` and optionally an `Exception`, as shown below.

```
public interface ILogger
{
    void Log<TState>(
        Microsoft.Extensions.Logging.LogLevel logLevel,
        Microsoft.Extensions.Logging.EventId eventId,
        TState state,
        System.Exception? exception,
        Func<TState, System.Exception?, string> formatter);
}
```

As a general rule, the first instance of `ILogger`, `LogLevel`, and `Exception` are treated specially in the log method signature of the source generator. Subsequent instances are treated like normal parameters to the message template:

```
// This is a valid attribute usage
[LoggerMessage(
    EventId = 110, Level = LogLevel.Debug, Message = "M1 {ex3} {ex2}")]
public static partial void ValidLogMethod(
    ILogger logger,
    Exception ex,
    Exception ex2,
    Exception ex3);

// This causes a warning
[LoggerMessage(
    EventId = 0, Level = LogLevel.Debug, Message = "M1 {ex} {ex2}")]
public static partial void WarningLogMethod(
    ILogger logger,
    Exception ex,
    Exception ex2);
```

IMPORTANT

The warnings emitted provide details as to the correct usage of the `LoggerMessageAttribute`. In the preceding example, the `WarningLogMethod` will report a `DiagnosticSeverity.Warning` of `SYSLIB0025`.

Don't include a template for `ex` in the logging message since it is implicitly taken care of.

Case-insensitive template name support

The generator does case-insensitive comparison between items in message template and argument names in the log message. This means that when the `ILogger` enumerates the state, the argument is picked up by message template, which can make the logs nicer to consume:

```
public partial class LoggingExample
{
    private readonly ILogger _logger;

    public LoggingExample(ILogger logger)
    {
        _logger = logger;
    }

    [LoggerMessage(
        EventId = 10,
        Level = LogLevel.Information,
        Message = "Welcome to {City} {Province}!")]
    public partial void LogMethodSupportsPascalCasingOfNames(
        string city, string province);

    public void TestLogging()
    {
        LogMethodSupportsPascalCasingOfNames("Vancouver", "BC");
    }
}
```

Consider the example logging output when using the `JsonConsole` formatter:

```
{
    "EventId": 13,
    "LogLevel": "Information",
    "Category": "LoggingExample",
    "Message": "Welcome to Vancouver BC!",
    "State": {
        "Message": "Welcome to Vancouver BC!",
        "City": "Vancouver",
        "Province": "BC",
        "{OriginalFormat)": "Welcome to {City} {Province}!"
    }
}
```

Indeterminate parameter order

There are no constraints on the ordering of log method parameters. A developer could define the `ILogger` as the last parameter, although it may appear a bit awkward.

```
[LoggerMessage(
    EventId = 110,
    Level = LogLevel.Debug,
    Message = "M1 {ex3} {ex2}")]
static partial void LogMethod(
    Exception ex,
    Exception ex2,
    Exception ex3,
    ILogger logger);
```

TIP

The order of the parameters on a log method is *not* required to correspond to the order of the template placeholders.

Instead, the placeholder names in the template are expected to match the parameters. Consider the following

`JsonConsole` output and the order of the errors.

```
{
    "EventId": 110,
    "LogLevel": "Debug",
    "Category": "ConsoleApp.Program",
    "Message": "M1 System.Exception: Third time's the charm. System.Exception: This is the second error.",
    "State": {
        "Message": "M1 System.Exception: Third time's the charm. System.Exception: This is the second
error.",
        "ex2": "System.Exception: This is the second error.",
        "ex3": "System.Exception: Third time's the charm.",
        "{OriginalFormat)": "M1 {ex3} {ex2}"
    }
}
```

Additional logging examples

The following samples demonstrate how to retrieve the event name, set the log level dynamically, and format logging parameters. The logging methods are:

- `LogWithCustomEventName` : Retrieve event name via `LoggerMessage` attribute.
- `LogWithDynamicLogLevel` : Set log level dynamically, to allow log level to be set based on configuration input.
- `UsingFormatSpecifier` : Use format specifiers to format logging parameters.

```

public partial class LoggingSample
{
    private readonly ILogger _logger;

    public LoggingSample(ILogger logger)
    {
        _logger = logger;
    }

    [LoggerMessage(
        EventId = 20,
        Level = LogLevel.Critical,
        Message = "Value is {value:E}")]
    public static partial void UsingFormatSpecifier(
        ILogger logger, double value);

    [LoggerMessage(
        EventId = 9,
        Level = LogLevel.Trace,
        Message = "Fixed message",
        EventName = "CustomEventName")]
    public partial void LogWithCustomEventName();

    [LoggerMessage(
        EventId = 10,
        Message = "Welcome to {city} {province}!")]
    public partial void LogWithDynamicLogLevel(
        string city, LogLevel level, string province);

    public void TestLogging()
    {
        LogWithCustomEventName();

        LogWithDynamicLogLevel("Vancouver", LogLevel.Warning, "BC");
        LogWithDynamicLogLevel("Vancouver", LogLevel.Information, "BC");

        UsingFormatSpecifier(logger, 12345.6789);
    }
}

```

Consider the example logging output when using the `SimpleConsole` formatter:

```

trce: LoggingExample[9]
    Fixed message
warn: LoggingExample[10]
    Welcome to Vancouver BC!
info: LoggingExample[10]
    Welcome to Vancouver BC!
crit: LoggingExample[20]
    Value is 1.234568E+004

```

Consider the example logging output when using the `JsonConsole` formatter:

```
{
    "EventId": 9,
    "LogLevel": "Trace",
    "Category": "LoggingExample",
    "Message": "Fixed message",
    "State": {
        "Message": "Fixed message",
        "{OriginalFormat)": "Fixed message"
    }
}
{
    "EventId": 10,
    "LogLevel": "Warning",
    "Category": "LoggingExample",
    "Message": "Welcome to Vancouver BC!",
    "State": {
        "Message": "Welcome to Vancouver BC!",
        "city": "Vancouver",
        "province": "BC",
        "{OriginalFormat)": "Welcome to {city} {province}!"
    }
}
{
    "EventId": 10,
    "LogLevel": "Information",
    "Category": "LoggingExample",
    "Message": "Welcome to Vancouver BC!",
    "State": {
        "Message": "Welcome to Vancouver BC!",
        "city": "Vancouver",
        "province": "BC",
        "{OriginalFormat)": "Welcome to {city} {province}!"
    }
}
{
    "EventId": 20,
    "LogLevel": "Critical",
    "Category": "LoggingExample",
    "Message": "Value is 1.234568E+004",
    "State": {
        "Message": "Value is 1.234568E+004",
        "value": 12345.6789,
        "{OriginalFormat)": "Value is {value:E}"
    }
}
}
```

Summary

With the advent of C# source generators, writing highly performant logging APIs is much easier. Using the source generator approach has several key benefits:

- Allows the logging structure to be preserved and enables the exact format syntax required by [Message Templates](#).
- Allows supplying alternative names for the template placeholders and using format specifiers.
- Allows the passing of all original data as-is, without any complication around how it's stored prior to something being done with it (other than creating a `string`).
- Provides logging-specific diagnostics, emits warnings for duplicate event IDs.

Additionally, there are benefits over manually using [LoggerMessage.Define](#):

- Shorter and simpler syntax: Declarative attribute usage rather than coding boilerplate.
- Guided developer experience: The generator gives warnings to help developers do the right thing.

- Support for an arbitrary number of logging parameters. `LoggerMessage.Define` supports a maximum of six.
- Support for dynamic log level. This is not possible with `LoggerMessage.Define` alone.

See also

- [Logging in .NET](#)
- [High-performance logging in .NET](#)
- [Console log formatting](#)
- [NuGet: Microsoft.Extensions.Logging.Abstractions](#)

Implement a custom logging provider in .NET

9/20/2022 • 4 minutes to read • [Edit Online](#)

There are many [logging providers](#) available for common logging needs. You may need to implement a custom [ILoggerProvider](#) when one of the available providers doesn't suit your application needs. In this article, you'll learn how to implement a custom logging provider that can be used to colorize logs in the console.

TIP

The custom logging provider example source code is available in the [Docs Github repo](#). For more information, see [GitHub: .NET Docs - Console Custom Logging](#).

Sample custom logger configuration

The sample creates different color console entries per log level and event ID using the following configuration type:

```
using Microsoft.Extensions.Logging;

public class ColorConsoleLoggerConfiguration
{
    public int EventId { get; set; }

    public Dictionary<LogLevel, ConsoleColor> LogLevelToColorMap { get; set; } = new()
    {
        [LogLevel.Information] = ConsoleColor.Green
    };
}
```

The preceding code sets the default level to `Information`, the color to `Green`, and the `EventId` is implicitly `0`.

Create the custom logger

The `ILogger` implementation category name is typically the logging source. For example, the type where the logger is created:

```

using Microsoft.Extensions.Logging;

public sealed class ColorConsoleLogger : ILogger
{
    private readonly string _name;
    private readonly Func<ColorConsoleLoggerConfiguration> _getCurrentConfig;

    public ColorConsoleLogger(
        string name,
        Func<ColorConsoleLoggerConfiguration> getCurrentConfig) =>
        (_name, _getCurrentConfig) = (name, getCurrentConfig);

    public IDisposable BeginScope<TState>(TState state) => default!;

    public bool IsEnabled(LogLevel logLevel) =>
        _getCurrentConfig().LogLevelToColorMap.ContainsKey(logLevel);

    public void Log<TState>(
        LogLevel logLevel,
        EventId eventId,
        TState state,
        Exception? exception,
        Func<TState, Exception?, string> formatter)
    {
        if (!IsEnabled(logLevel))
        {
            return;
        }

        ColorConsoleLoggerConfiguration config = _getCurrentConfig();
        if (config.EventId == 0 || config.EventId == eventId.Id)
        {
            ConsoleColor originalColor = Console.ForegroundColor;

            Console.ForegroundColor = config.LogLevelToColorMap[logLevel];
            Console.WriteLine($"[{eventId.Id,2}: {logLevel,-12}]");

            Console.ForegroundColor = originalColor;
            Console.Write($"      {_name} - ");

            Console.ForegroundColor = config.LogLevelToColorMap[logLevel];
            Console.Write($"{formatter(state, exception)}");

            Console.ForegroundColor = originalColor;
            Console.WriteLine();
        }
    }
}

```

The preceding code:

- Creates a logger instance per category name.
- Checks `_getCurrentConfig().LogLevelToColorMap.ContainsKey(logLevel)` in `.IsEnabled`, so each `logLevel` has a unique logger. In this implementation, each log level requires an explicit configuration entry in order to log.

```

public bool IsEnabled(LogLevel logLevel) =>
    _getCurrentConfig().LogLevelToColorMap.ContainsKey(logLevel);

```

The logger is instantiated with the `name` and a `Func<ColorConsoleLoggerConfiguration>`, which returns the current config — this handles updates to the config values as monitored through the `IOptionsMonitor<TOptions>.OnChange` callback.

IMPORTANT

The `ILoggerLog` implementation checks if the `config.EventId` value is set. When `config.EventId` is not set or when it matches the exact `logEntry.EventId`, the logger logs in color.

Custom logger provider

The `ILoggerProvider` object is responsible for creating logger instances. It's not necessary to create a logger instance per category, but it makes sense for some loggers, like NLog or log4net. This strategy allows you to choose different logging output targets per category, as in the following example:

```
using System.Collections.Concurrent;
using System.Runtime.Versioning;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Options;

[UnsupportedOSPlatform("browser")]
[ProviderAlias("ColorConsole")]
public sealed class ColorConsoleLoggerProvider : ILoggerProvider
{
    private readonly IDisposable _onChangeToken;
    private ColorConsoleLoggerConfiguration _currentConfig;
    private readonly ConcurrentDictionary<string, ColorConsoleLogger> _loggers =
        new(StringComparer.OrdinalIgnoreCase);

    public ColorConsoleLoggerProvider(
        IOptionsMonitor<ColorConsoleLoggerConfiguration> config)
    {
        _currentConfig = config.CurrentValue;
        _onChangeToken = config.OnChange(updatedConfig => _currentConfig = updatedConfig);
    }

    public ILogger CreateLogger(string categoryName) =>
        _loggers.GetOrAdd(categoryName, name => new ColorConsoleLogger(name, GetCurrentConfig()));

    private ColorConsoleLoggerConfiguration GetCurrentConfig() => _currentConfig;

    public void Dispose()
    {
        _loggers.Clear();
        _onChangeToken.Dispose();
    }
}
```

In the preceding code, `CreateLogger` creates a single instance of the `ColorConsoleLogger` per category name and stores it in the `ConcurrentDictionary< TKey, TValue >`. Additionally, the `IOptionsMonitor< TOptions >` interface is required to update changes to the underlying `ColorConsoleLoggerConfiguration` object.

To control the configuration of the `colorConsoleLogger`, you define an alias on its provider:

```
[UnsupportedOSPlatform("browser")]
[ProviderAlias("ColorConsole")]
public sealed class ColorConsoleLoggerProvider : ILoggerProvider
```

The `colorConsoleLoggerProvider` class defines two class-scoped attributes:

- **UnsupportedOSPlatformAttribute:** The `ColorConsoleLogger` type is *not supported* in the "browser".
- **ProviderAliasAttribute:** Configuration sections can define options using the "ColorConsole" key.

The configuration can be specified with any valid [configuration provider](#). Consider the following `appsettings.json` file:

```
{  
    "Logging": {  
        "ColorConsole": {  
            "LogLevelToColorMap": {  
                "Information": "DarkGreen",  
                "Warning": "Cyan",  
                "Error": "Red"  
            }  
        }  
    }  
}
```

This configures the log levels to the following values:

- `LogLevel.Information`: `ConsoleColor.DarkGreen`
- `LogLevel.Warning`: `ConsoleColor.Cyan`
- `LogLevel.Error`: `ConsoleColor.Red`

The `Information` log level is set to `DarkGreen`, which overrides the default value set in the `ColorConsoleLoggerConfiguration` object.

Usage and registration of the custom logger

By convention, registering services for dependency injection happens as part of the startup routine of an application. The registration occurs in the `Program` class, or could be delegated to a `Startup` class. In this example, you'll register directly from the `Program.cs`.

To add the custom logging provider and corresponding logger, add an `ILoggerProvider` with `ILoggingBuilder` from the `HostingHostBuilderExtensions.ConfigureLogging(IHostBuilder, Action<ILoggingBuilder>)`:

```
using Microsoft.Extensions.DependencyInjection;  
using Microsoft.Extensions.Hosting;  
using Microsoft.Extensions.Logging;  
  
using IHost host = Host.CreateDefaultBuilder(args)  
.ConfigureLogging(builder =>  
    builder.ClearProviders()  
    .AddColorConsoleLogger(configuration =>  
    {  
        // Replace warning value from appsettings.json of "Cyan"  
        configuration.LogLevelToColorMap[LogLevel.Warning] = ConsoleColor.DarkCyan;  
        // Replace warning value from appsettings.json of "Red"  
        configuration.LogLevelToColorMap[LogLevel.Error] = ConsoleColor.DarkRed;  
    })  
    .Build();  
  
var logger = host.Services.GetRequiredService<ILogger<Program>>();  
  
logger.LogDebug(1, "Does this line get hit?"); // Not logged  
logger.LogInformation(3, "Nothing to see here."); // Logs in ConsoleColor.DarkGreen  
logger.LogWarning(5, "Warning... that was odd."); // Logs in ConsoleColor.DarkCyan  
logger.LogError(7, "Oops, there was an error."); // Logs in ConsoleColor.DarkRed  
logger.LogTrace(5!, "== 120."); // Not logged  
  
await host.RunAsync();
```

The `ILoggingBuilder` creates one or more `ILogger` instances. The `ILogger` instances are used by the

framework to log the information.

The configuration from the *appsettings.json* file overrides the following values:

- [LogLevel.Warning: ConsoleColor.DarkCyan](#)
- [LogLevel.Error: ConsoleColor.DarkRed](#)

By convention, extension methods on `ILoggingBuilder` are used to register the custom provider:

```
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.DependencyInjection.Extensions;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Logging.Configuration;

public static class ColorConsoleLoggerExtensions
{
    public static ILoggingBuilder AddColorConsoleLogger(
        this ILoggingBuilder builder)
    {
        builder.AddConfiguration();

        builder.Services.TryAddEnumerable(
            ServiceDescriptor.Singleton<ILoggerProvider, ColorConsoleLoggerProvider>());

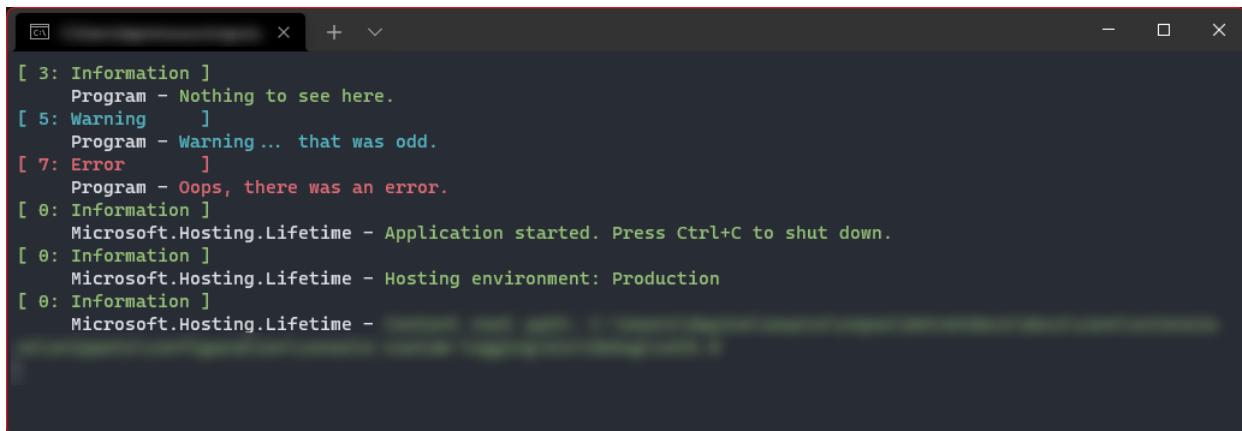
        LoggerProviderOptions.RegisterProviderOptions
            <ColorConsoleLoggerConfiguration, ColorConsoleLoggerProvider>(builder.Services);

        return builder;
    }

    public static ILoggingBuilder AddColorConsoleLogger(
        this ILoggingBuilder builder,
        Action<ColorConsoleLoggerConfiguration> configure)
    {
        builder.AddColorConsoleLogger();
        builder.Services.Configure(configure);

        return builder;
    }
}
```

Running this simple application will render color output to the console window similar to the following image:

A screenshot of a terminal window showing colored log output. The window has a dark theme with light-colored text. The logs are color-coded by severity:

- [3: Information] Program - Nothing to see here.
- [5: Warning] Program - Warning ... that was odd.
- [7: Error] Program - Oops, there was an error.
- [0: Information] Microsoft.Hosting.Lifetime - Application started. Press Ctrl+C to shut down.
- [0: Information] Microsoft.Hosting.Lifetime - Hosting environment: Production
- [0: Information] Microsoft.Hosting.Lifetime -

The colors used are:

- Information: Light green
- Warning: Yellow
- Error: Red
- Information (hosting): Light green

See also

- [Logging in .NET](#)
- [Logging providers in .NET](#)
- [Dependency injection in .NET](#)

- High-performance logging in .NET

High-performance logging in .NET

9/20/2022 • 6 minutes to read • [Edit Online](#)

The [LoggerMessage](#) class exposes functionality to create cacheable delegates that require fewer object allocations and reduced computational overhead compared to [logger extension methods](#), such as [LogInformation](#) and [LogDebug](#). For high-performance logging scenarios, use the [LoggerMessage](#) pattern.

[LoggerMessage](#) provides the following performance advantages over Logger extension methods:

- Logger extension methods require "boxing" (converting) value types, such as `int`, into `object`. The [LoggerMessage](#) pattern avoids boxing by using static [Action](#) fields and extension methods with strongly-typed parameters.
- Logger extension methods must parse the message template (named format string) every time a log message is written. [LoggerMessage](#) only requires parsing a template once when the message is defined.

The sample app demonstrates [LoggerMessage](#) features with a priority queue processing worker service. The app processes work items in priority order. As these operations occur, log messages are generated using the [LoggerMessage](#) pattern.

TIP

All of the logging example source code is available in the [Samples Browser](#) for download. For more information, see [Browse code samples: Logging in .NET](#).

Define a logger message

Use [Define\(LogLevel, EventId, String\)](#) to create an [Action](#) delegate for logging a message. [Define](#) overloads permit passing up to six type parameters to a named format string (template).

The string provided to the [Define](#) method is a template and not an interpolated string. Placeholders are filled in the order that the types are specified. Placeholder names in the template should be descriptive and consistent across templates. They serve as property names within structured log data. We recommend [Pascal casing](#) for placeholder names. For example, `{Item}`, `{DateTime}`.

Each log message is an [Action](#) held in a static field created by [LoggerMessage.Define](#). For example, the sample app creates a field to describe a log message for the processing of work items:

```
private static readonly Action<ILogger, Exception> _failedToProcessWorkItem;
```

For the [Action](#), specify:

- The log level.
- A unique event identifier ([EventId](#)) with the name of the static extension method.
- The message template (named format string).

As work items are dequeued for processing the worker service app sets the:

- Log level to [LogLevel.Critical](#).
- Event id to `13` with the name of the `FailedToProcessWorkItem` method.
- Message template (named format string) to a string.

```
_failedToProcessWorkItem = LoggerMessage.Define(
    LogLevel.Critical,
    new EventId(13, nameof(FailedToProcessWorkItem)),
    "Epic failure processing item!");
```

The `LoggerMessage.Define` method is used to configure and define a `Action` delegate, that represents a log message.

Structured logging stores may use the event name when it's supplied with the event id to enrich logging. For example, `Serilog` uses the event name.

The `Action` is invoked through a strongly-typed extension method. The `PriorityItemProcessed` method logs a message every time a work item is being processed. Whereas, `FailedToProcessWorkItem` is called when (and if) an exception occurs:

```
protected override async Task ExecuteAsync(
    CancellationToken stoppingToken)
{
    using IDisposable? scope = _logger.ProcessingWorkScope(DateTime.Now);
    while (!stoppingToken.IsCancellationRequested)
    {
        WorkItem? nextItem = _priorityQueue.ProcessNextHighestPriority();
        try
        {
            if (nextItem is not null)
            {
                _logger.PriorityItemProcessed(nextItem);
            }
        }
        catch (Exception ex)
        {
            _logger.FailedToProcessWorkItem(ex);
        }

        await Task.Delay(1000, stoppingToken);
    }
}
```

Inspect the app's console output:

```
crit: WorkerServiceOptions.Example.Worker[13]
  Epic failure processing item!
System.Exception: Failed to verify communications.
  at WorkerServiceOptions.Example.Worker.ExecuteAsync(CancellationToken stoppingToken) in
..\\Worker.cs:line 27
```

To pass parameters to a log message, define up to six types when creating the static field. The sample app logs the work item details when processing items by defining a `WorkItem` type for the `Action` field:

```
private static readonly Action<ILogger, WorkItem, Exception> _processingPriorityItem;
```

The delegate's log message template receives its placeholder values from the types provided. The sample app defines a delegate for adding a work item where the item parameter is a `WorkItem`:

```
_processingPriorityItem = LoggerMessage.Define<WorkItem>(
    LogLevel.Information,
    new EventId(1, nameof(PriorityItemProcessed)),
    "Processing priority item: {Item}");
```

The static extension method for logging that a work item is being processed, `PriorityItemProcessed`, receives the work item argument value and passes it to the `Action` delegate:

```
public static void PriorityItemProcessed(
    this ILogger logger, WorkItem workItem) =>
    _processingPriorityItem(logger, workItem, default!);
```

In the worker service's `ExecuteAsync` method, `PriorityItemProcessed` is called to log the message:

```
protected override async Task ExecuteAsync(
    CancellationToken stoppingToken)
{
    using IDisposable? scope = _logger.ProcessingWorkScope(DateTime.Now);
    while (!stoppingToken.IsCancellationRequested)
    {
        WorkItem? nextItem = _priorityQueue.ProcessNextHighestPriority();
        try
        {
            if (nextItem is not null)
            {
                _logger.PriorityItemProcessed(nextItem);
            }
        }
        catch (Exception ex)
        {
            _logger FailedToProcessWorkItem(ex);
        }

        await Task.Delay(1000, stoppingToken);
    }
}
```

Inspect the app's console output:

```
info: WorkerServiceOptions.Example.Worker[1]
      Processing priority item: Priority-Extreme (50db062a-9732-4418-936d-110549ad79e4): 'Verify
communications'
```

Define logger message scope

The `DefineScope(string)` method creates a `Func<TResult>` delegate for defining a `log scope`. `DefineScope` overloads permit passing up to three type parameters to a named format string (template).

As is the case with the `Define` method, the string provided to the `DefineScope` method is a template and not an interpolated string. Placeholders are filled in the order that the types are specified. Placeholder names in the template should be descriptive and consistent across templates. They serve as property names within structured log data. We recommend `Pascal casing` for placeholder names. For example, `{Item}`, `{DateTime}`.

Define a `log scope` to apply to a series of log messages using the `DefineScope` method. Enable `IncludeScopes` in the console logger section of `appsettings.json`:

```
{  
    "Logging": {  
        "Console": {  
            "IncludeScopes": true  
        },  
        "LogLevel": {  
            "Default": "Information",  
            "Microsoft": "Warning",  
            "Microsoft.Hosting.Lifetime": "Information"  
        }  
    }  
}
```

To create a log scope, add a field to hold a `Func<TResult>` delegate for the scope. The sample app creates a field called `_processingWorkScope` (*Internal/LoggerExtensions.cs*):

```
private static Func<ILogger, DateTime, IDisposable> _processingWorkScope;
```

Use `DefineScope` to create the delegate. Up to three types can be specified for use as template arguments when the delegate is invoked. The sample app uses a message template that includes the date time in which processing started:

```
_processingWorkScope =  
    LoggerMessage.DefineScope<DateTime>(  
        "Processing work, started at: {DateTime}");
```

Provide a static extension method for the log message. Include any type parameters for named properties that appear in the message template. The sample app takes in a `DateTime` for a custom time stamp to log and returns `_processingWorkScope`:

```
public static IDisposable ProcessingWorkScope(  
    this ILogger logger, DateTime time) =>  
    _processingWorkScope(logger, time);
```

The scope wraps the logging extension calls in a `using` block:

```

protected override async Task ExecuteAsync(
    CancellationToken stoppingToken)
{
    using IDisposable? scope = _logger.ProcessingWorkScope(DateTime.Now);
    while (!stoppingToken.IsCancellationRequested)
    {
        WorkItem? nextItem = _priorityQueue.ProcessNextHighestPriority();
        try
        {
            if (nextItem is not null)
            {
                _logger.PriorityItemProcessed(nextItem);
            }
        }
        catch (Exception ex)
        {
            _loggerFailedToProcessWorkItem(ex);
        }

        await Task.Delay(1000, stoppingToken);
    }
}
}

```

Inspect the log messages in the app's console output. The following result shows priority ordering of log messages with the log scope message included:

```

info: WorkerServiceOptions.Example.Worker[1]
=> Processing work, started at: 09/25/2020 14:30:45
Processing priority item: Priority-Extreme (f5090ede-a337-4041-b914-f6bc0db5ae64): 'Verify
communications'
info: Microsoft.Hosting.Lifetime[0]
Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
Content root path: ..\worker-service-options
info: WorkerServiceOptions.Example.Worker[1]
=> Processing work, started at: 09/25/2020 14:30:45
Processing priority item: Priority-High (496d440f-2007-4391-b179-09d75ab52373): 'Validate collection'
info: WorkerServiceOptions.Example.Worker[1]
=> Processing work, started at: 09/25/2020 14:30:45
Processing priority item: Priority-Medium (dea9e3f4-d7df-46d2-b7cd-5e0232eb98a5): 'Propagate
selections'
info: WorkerServiceOptions.Example.Worker[1]
=> Processing work, started at: 09/25/2020 14:30:45
Processing priority item: Priority-Medium (089d7f0d-da72-4b55-92fe-57b147838056): 'Enter pooling
[contention]'
info: WorkerServiceOptions.Example.Worker[1]
=> Processing work, started at: 09/25/2020 14:30:45
Processing priority item: Priority-Low (6e68c4be-089f-4450-9080-1ea63fcbb686): 'Health check network'
info: WorkerServiceOptions.Example.Worker[1]
=> Processing work, started at: 09/25/2020 14:30:45
Processing priority item: Priority-Deferred (6f324134-6bb6-455f-81d4-553ab307c421): 'Ping weather
service'
info: WorkerServiceOptions.Example.Worker[1]
=> Processing work, started at: 09/25/2020 14:30:45
Processing priority item: Priority-Deferred (37bf736c-7a26-4a2a-9e56-e89bcf3b8f35): 'Set process
state'

```

Log level guarded optimizations

An additional performance optimization can be made by checking the [LogLevel](#), with

`ILogger.IsEnabled(LogLevel)` before an invocation to the corresponding `Log*` method. When logging isn't configured for the given `LogLevel`, the following statements are true:

- `ILogger.Log` is not called.
- An allocation of `object[]` representing the parameters is avoided.
- Value type boxing is avoided.

For more information:

- [Micro benchmarks in the .NET runtime](#)
- [Background and motivation for log level checks](#)

See also

- [Logging in .NET](#)

Console log formatting

9/20/2022 • 12 minutes to read • [Edit Online](#)

In .NET 5, support for custom formatting was added to console logs in the `Microsoft.Extensions.Logging.Console` namespace. There are three predefined formatting options available: `Simple`, `Systemd`, and `Json`.

IMPORTANT

Previously, the `ConsoleLoggerFormat` enum allowed for selecting the desired log format, either human readable which was the `Default`, or single line which is also known as `Systemd`. However, these were **not** customizable, and are now deprecated.

In this article, you will learn about console log formatters. The sample source code demonstrates how to:

- Register a new formatter
- Select a registered formatter to use
 - Either through code, or [configuration](#)
- Implement a custom formatter
 - Update configuration via `IOptionsMonitor<TOptions>`
 - Enable custom color formatting

TIP

All of the logging example source code is available in the [Samples Browser](#) for download. For more information, see [Browse code samples: Logging in .NET](#).

Register formatter

The `Console` logging provider has several predefined formatters, and exposes the ability to author your own custom formatter. To register any of the available formatters, use the corresponding `Add{Type}Console` extension method:

AVAILABLE TYPES	METHOD TO REGISTER TYPE
<code>ConsoleFormatterNames.Json</code>	<code>ConsoleLoggerExtensions.AddJsonConsole</code>
<code>ConsoleFormatterNames.Simple</code>	<code>ConsoleLoggerExtensions.AddSimpleConsole</code>
<code>ConsoleFormatterNames.Systemd</code>	<code>ConsoleLoggerExtensions.AddSystemdConsole</code>

Simple

To use the `Simple` console formatter, register it with `AddSimpleConsole`:

```

using Microsoft.Extensions.Logging;

namespace Console.ExampleFormatters.Simple;

class Program
{
    static void Main()
    {
        using ILoggerFactory loggerFactory =
            LoggerFactory.Create(builder =>
                builder.AddSimpleConsole(options =>
                {
                    options.IncludeScopes = true;
                    options.SingleLine = true;
                    options.TimestampFormat = "hh:mm:ss ";
                }));
        ILogger<Program> logger = loggerFactory.CreateLogger<Program>();
        using (logger.BeginScope("[scope is enabled]"))
        {
            logger.LogInformation("Hello World!");
            logger.LogInformation("Logs contain timestamp and log level.");
            logger.LogInformation("Each log message is fit in a single line.");
        }
    }
}

```

In the preceding sample source code, the [ConsoleFormatterNames.Simple](#) formatter was registered. It provides logs with the ability to not only wrap information such as time and log level in each log message, but also allows for ANSI color embedding and indentation of messages.

Systemd

The [ConsoleFormatterNames.Systemd](#) console logger:

- Uses the "Syslog" log level format and severities
- Does **not** format messages with colors
- Always logs messages in a single line

This is commonly useful for containers, which often make use of `systemd` console logging. With .NET 5, the `Simple` console logger also enables a compact version that logs in a single line, and also allows for disabling colors as shown in an earlier sample.

```

using Microsoft.Extensions.Logging;

namespace Console.ExampleFormatters.Systemd;

class Program
{
    static void Main()
    {
        using ILoggerFactory loggerFactory =
            LoggerFactory.Create(builder =>
                builder.AddSystemdConsole(options =>
                {
                    options.IncludeScopes = true;
                    options.TimestampFormat = "hh:mm:ss ";
                }));
        
        ILogger<Program> logger = loggerFactory.CreateLogger<Program>();
        using (logger.BeginScope("[scope is enabled]"))
        {
            logger.LogInformation("Hello World!");
            logger.LogInformation("Logs contain timestamp and log level.");
            logger.LogInformation("Systemd console logs never provide color options.");
            logger.LogInformation("Systemd console logs always appear in a single line.");
        }
    }
}

```

Json

To write logs in a JSON format, the `Json` console formatter is used. The sample source code shows how an ASP.NET Core app might register it. Using the `webapp` template, create a new ASP.NET Core app with the [dotnet new](#) command:

```
dotnet new webapp -o Console.ExampleFormatters.Json
```

When running the app, using the template code, you get the default log format below:

```

info: Console.ExampleFormatters.Json.Startup[0]
      Hello .NET friends!
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: https://localhost:5001
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://localhost:5000
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
      Content root path: ..\snippets\logging\console-formatter-json

```

By default, the `simple` console log formatter is selected with default configuration. You change this by calling `AddJsonConsole` in the `Program.cs`.

```

using System.Text.Json;
using Console.ExampleFormatters.Json;

using IHost host = Host.CreateDefaultBuilder(args)
    .ConfigureWebHostDefaults(builder => builder.UseStartup<Startup>())
    .ConfigureLogging(builder =>
        builder.AddJsonConsole(options =>
    {
        options.IncludeScopes = false;
        options.TimestampFormat = "hh:mm:ss ";
        options.JsonWriterOptions = new JsonWriterOptions
        {
            Indented = true
        };
    }))
    .Build();

var logger =
    host.Services
        .GetRequiredService<ILoggerFactory>()
        .CreateLogger<Startup>();

logger.LogInformation("Hello .NET friends!");

await host.RunAsync();

```

Alternatively, you can also configure this using logging configuration, such as that found in the *appsettings.json* file:

```

{
    "Logging": {
        "LogLevel": {
            "Default": "Information",
            "Microsoft": "Warning",
            "Microsoft.Hosting.Lifetime": "Information"
        },
        "Console": {
            "LogLevel": {
                "Default": "Information",
                "Microsoft": "Warning",
                "Microsoft.Hosting.Lifetime": "Information"
            },
            "FormatterName": "json",
            "FormatterOptions": {
                "SingleLine": true,
                "IncludeScopes": true,
                "TimestampFormat": "HH:mm:ss ",
                "UseUtcTimestamp": true,
                "JsonWriterOptions": {
                    "Indented": true
                }
            }
        }
    },
    "AllowedHosts": "*"
}

```

Run the app again, with the above change, the log message is now formatted as JSON:

```
{
    "Timestamp": "02:28:19 ",
    "EventId": 0,
    "LogLevel": "Information",
    "Category": "Console.ExampleFormatters.Json.Startup",
}
```

```
"Message": "Hello .NET friends!",
"State": {
    "Message": "Hello .NET friends!",
    "{OriginalFormat)": "Hello .NET friends!"
}
}
{
    "Timestamp": "02:28:21 ",
    "EventId": 14,
    "LogLevel": "Information",
    "Category": "Microsoft.Hosting.Lifetime",
    "Message": "Now listening on: https://localhost:5001",
    "State": {
        "Message": "Now listening on: https://localhost:5001",
        "address": "https://localhost:5001",
        "{OriginalFormat)": "Now listening on: {address}"
    }
}
{
    "Timestamp": "02:28:21 ",
    "EventId": 14,
    "LogLevel": "Information",
    "Category": "Microsoft.Hosting.Lifetime",
    "Message": "Now listening on: http://localhost:5000",
    "State": {
        "Message": "Now listening on: http://localhost:5000",
        "address": "http://localhost:5000",
        "{OriginalFormat)": "Now listening on: {address}"
    }
}
{
    "Timestamp": "02:28:21 ",
    "EventId": 0,
    "LogLevel": "Information",
    "Category": "Microsoft.Hosting.Lifetime",
    "Message": "Application started. Press Ctrl\u002BC to shut down.",
    "State": {
        "Message": "Application started. Press Ctrl\u002BC to shut down.",
        "{OriginalFormat)": "Application started. Press Ctrl\u002BC to shut down."
    }
}
{
    "Timestamp": "02:28:21 ",
    "EventId": 0,
    "LogLevel": "Information",
    "Category": "Microsoft.Hosting.Lifetime",
    "Message": "Hosting environment: Development",
    "State": {
        "Message": "Hosting environment: Development",
        "envName": "Development",
        "{OriginalFormat)": "Hosting environment: {envName}"
    }
}
{
    "Timestamp": "02:28:21 ",
    "EventId": 0,
    "LogLevel": "Information",
    "Category": "Microsoft.Hosting.Lifetime",
    "Message": "Content root path: .\\snippets\\logging\\console-formatter-json",
    "State": {
        "Message": "Content root path: .\\snippets\\logging\\console-formatter-json",
        "contentRoot": ".\\snippets\\logging\\console-formatter-json",
        "{OriginalFormat)": "Content root path: {contentRoot}"
    }
}
```

TIP

The `Json` console formatter, by default, logs each message in a single line. In order to make it more readable while configuring the formatter, set `JsonWriterOptions.Indented` to `true`.

Caution

When using the Json console formatter, do not pass in log messages that have already been serialized as JSON. The logging infrastructure itself already manages the serialization of log messages, so if you're to pass in a log message that is already serialized—it will be double serialized, thus causing malformed output.

Set formatter with configuration

The previous samples have shown how to register a formatter programmatically. Alternatively, this can be done with [configuration](#). Consider the previous web application sample source code, if you update the `appsettings.json` file rather than calling `ConfigureLogging` in the `Program.cs` file, you could get the same outcome. The updated `appsettings.json` file would configure the formatter as follows:

```
{  
    "Logging": {  
        "LogLevel": {  
            "Default": "Information",  
            "Microsoft": "Warning",  
            "Microsoft.Hosting.Lifetime": "Information"  
        },  
        "Console": {  
            "LogLevel": {  
                "Default": "Information",  
                "Microsoft": "Warning",  
                "Microsoft.Hosting.Lifetime": "Information"  
            },  
            "FormatterName": "json",  
            "FormatterOptions": {  
                "SingleLine": true,  
                "IncludeScopes": true,  
                "TimestampFormat": "HH:mm:ss ",  
                "UseUtcTimestamp": true,  
                "JsonWriterOptions": {  
                    "Indented": true  
                }  
            }  
        }  
    },  
    "AllowedHosts": "*"  
}
```

The two key values that need to be set are `"FormatterName"` and `"FormatterOptions"`. If a formatter with the value set for `"FormatterName"` is already registered, that formatter is selected, and its properties can be configured as long as they are provided as a key inside the `"FormatterOptions"` node. The predefined formatter names are reserved under [ConsoleFormatterNames](#):

- [ConsoleFormatterNames.Json](#)
- [ConsoleFormatterNames.Simple](#)
- [ConsoleFormatterNames.Systemd](#)

Implement a custom formatter

To implement a custom formatter, you need to:

- Create a subclass of [ConsoleFormatter](#), this represents your custom formatter
- Register your custom formatter with
 - [ConsoleLoggerExtensions.AddConsole](#)
 - [ConsoleLoggerExtensions.AddConsoleFormatter<TFormatter,TOptions>\(ILoggingBuilder, Action<TOptions>\)](#)

Create an extension method to handle this for you:

```
using Microsoft.Extensions.Logging;

namespace Console.ExampleFormatters.Custom;

public static class ConsoleLoggerExtensions
{
    public static ILoggingBuilder AddCustomFormatter(
        this ILoggingBuilder builder,
        Action<CustomOptions> configure) =>
        builder.AddConsole(options => options.FormatterName = "customName")
            .AddConsoleFormatter<CustomFormatter, CustomOptions>(configure);
}
```

The `CustomOptions` are defined as follows:

```
using Microsoft.Extensions.Logging.Console;

namespace Console.ExampleFormatters.Custom;

public class CustomOptions : ConsoleFormatterOptions
{
    public string? CustomPrefix { get; set; }
}
```

In the preceding code, the options are a subclass of [ConsoleFormatterOptions](#).

The `AddConsoleFormatter` API:

- Registers a subclass of `ConsoleFormatter`
- Handles configuration:
 - Uses a change token to synchronize updates, based on the [options pattern](#), and the [IOptionsMonitor](#) interface

```
using Microsoft.Extensions.Logging;

namespace Console.ExampleFormatters.Custom;

class Program
{
    static void Main()
    {
        using ILoggerFactory loggerFactory =
            LoggerFactory.Create(builder =>
                builder.AddCustomFormatter(options =>
                    options.CustomPrefix = " ~~~~~ "));

        ILogger<Program> logger = loggerFactory.CreateLogger<Program>();
        using (logger.BeginScope("TODO: Add logic to enable scopes"))
        {
            logger.LogInformation("Hello World!");
            logger.LogInformation("TODO: Add logic to enable timestamp and log level info.");
        }
    }
}
```

Define a `CustomerFormatter` subclass of `ConsoleFormatter`:

```

using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Logging.Abstractions;
using Microsoft.Extensions.Logging.Console;
using Microsoft.Extensions.Options;

namespace Console.ExampleFormatters.Custom;

public sealed class CustomFormatter : ConsoleFormatter, IDisposable
{
    private readonly IDisposable _optionsReloadToken;
    private CustomOptions _formatterOptions;

    public CustomFormatter(IOptionsMonitor<CustomOptions> options)
        // Case insensitive
        : base("customName") =>
        (_optionsReloadToken, _formatterOptions) =
            (options.OnChange(ReloadLoggerOptions), options.CurrentValue);

    private void ReloadLoggerOptions(CustomOptions options) =>
        _formatterOptions = options;

    public override void Write<TState>(
        in LogEntry<TState> logEntry,
        IExternalScopeProvider scopeProvider,
        TextWriter textWriter)
    {
        string? message =
            logEntry.Formatter?.Invoke(
                logEntry.State, logEntry.Exception);

        if (message is null)
        {
            return;
        }

        CustomLogicGoesHere(textWriter);
        textWriter.WriteLine(message);
    }

    private void CustomLogicGoesHere(TextWriter textWriter)
    {
        textWriter.Write(_formatterOptions.CustomPrefix);
    }

    public void Dispose() => _optionsReloadToken?.Dispose();
}

```

The preceding `CustomFormatter.Write<TState>` API dictates what text gets wrapped around each log message. A standard `ConsoleFormatter` should be able to wrap around scopes, time stamps, and severity level of logs at a minimum. Additionally, you can encode ANSI colors in the log messages, and provide desired indentations as well. The implementation of the `CustomFormatter.Write<TState>` lacks these capabilities.

For inspiration on further customizing formatting, see the existing implementations in the `Microsoft.Extensions.Logging.Console` namespace:

- [SimpleConsoleFormatter](#).
- [SystemdConsoleFormatter](#)
- [JsonConsoleFormatter](#)

Custom configuration options

To further customize the logging extensibility, your derived `ConsoleFormatterOptions` class can be configured from any [configuration provider](#). For example, you could use the [JSON configuration provider](#) to define your custom options. First define your `ConsoleFormatterOptions` subclass.

```

using Microsoft.Extensions.Logging.Console;

namespace Console.ExampleFormatters.CustomWithConfig;

public class CustomWrappingConsoleFormatterOptions : ConsoleFormatterOptions
{
    public string? CustomPrefix { get; set; }

    public string? CustomSuffix { get; set; }
}

```

The preceding console formatter options class defines two custom properties, representing a prefix and suffix. Next, define the `appsettings.json` file that will configure your console formatter options.

```
{
    "Logging": {
        "LogLevel": {
            "Default": "Information",
            "Microsoft": "Warning",
            "Microsoft.Hosting.Lifetime": "Information"
        },
        "Console": {
            "LogLevel": {
                "Default": "Information",
                "Microsoft": "Warning",
                "Microsoft.Hosting.Lifetime": "Information"
            },
            "FormatterName": "CustomTimePrefixingFormatter",
            "FormatterOptions": {
                "CustomPrefix": "|-<[",
                "CustomSuffix": "]>-|",
                "SingleLine": true,
                "IncludeScopes": true,
                "TimestampFormat": "HH:mm:ss.ffff",
                "UseUtcTimestamp": true,
                "JsonWriterOptions": {
                    "Indented": true
                }
            }
        }
    },
    "AllowedHosts": "*"
}
```

In the preceding JSON config file:

- The `"Logging"` node defines a `"Console"`.
- The `"Console"` node specifies a `"FormatterName"` of `"CustomTimePrefixingFormatter"`, which maps to a custom formatter.
- The `"FormatterOptions"` node defines a `"CustomPrefix"`, and `"CustomSuffix"`, as well as a few other derived options.

TIP

The `$.Logging.Console.FormatterOptions` JSON path is reserved, and will map to a custom `ConsoleFormatterOptions` when added using the `AddConsoleFormatter` extension method. This provides the ability to define custom properties, in addition to the ones available.

Consider the following `CustomDatePrefixingFormatter`:

```

using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Logging.Abstractions;
using Microsoft.Extensions.Logging.Console;
using Microsoft.Extensions.Options;

namespace Console.ExampleFormatters.CustomWithConfig;

public sealed class CustomTimePrefixingFormatter : ConsoleFormatter, IDisposable
{
    private readonly IDisposable _optionsReloadToken;
    private CustomWrappingConsoleFormatterOptions _formatterOptions;

    public CustomTimePrefixingFormatter(IOptionsMonitor<CustomWrappingConsoleFormatterOptions> options)
        // Case insensitive
        : base(nameof(CustomTimePrefixingFormatter)) =>
        (_optionsReloadToken, _formatterOptions) =
            (options.OnChange(ReloadLoggerOptions), options.CurrentValue);

    private void ReloadLoggerOptions(CustomWrappingConsoleFormatterOptions options) =>
        _formatterOptions = options;

    public override void Write<TState>(
        in LogEntry<TState> logEntry,
        IExternalScopeProvider scopeProvider,
        TextWriter textWriter)
    {
        string message =
            logEntry.Formatter(
                logEntry.State, logEntry.Exception);

        if (message == null)
        {
            return;
        }

        WritePrefix(textWriter);
        textWriter.Write(message);
        WriteSuffix(textWriter);
    }

    private void WritePrefix(TextWriter textWriter)
    {
        DateTime now = _formatterOptions.UseUtcTimestamp
            ? DateTime.UtcNow
            : DateTime.Now;

        textWriter.Write($"{_formatterOptions.CustomPrefix}
{now.ToString(_formatterOptions.TimestampFormat)}");
    }

    private void WriteSuffix(TextWriter textWriter) => textWriter.WriteLine($"{_formatterOptions.CustomSuffix}");
}

public void Dispose() => _optionsReloadToken?.Dispose();
}

```

In the preceding formatter implementation:

- The `CustomWrappingConsoleFormatterOptions` are monitored for change, and updated accordingly.
- Messages that are written are wrapped with the configured prefix, and suffix.
- A timestamp is added after the prefix, but before the message using the configured `ConsoleFormatterOptions.UseUtcTimestamp` and `ConsoleFormatterOptions.TimestampFormat` values.

To use custom configuration options, with custom formatter implementations, add when calling `ConfigureLogging(IHostBuilder, Action<HostBuilderContext, ILoggingBuilder>)`.

```

using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;

namespace Console.ExampleFormatters.CustomWithConfig;

class Program
{
    static void Main(string[] args)
    {
        using IHost host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(builder =>
            {
                builder.AddConsole()
                    .AddConsoleFormatter
                    <CustomTimePrefixingFormatter, CustomWrappingConsoleFormatterOptions>();
            })
            .Build();

        ILoggerFactory loggerFactory = host.Services.GetRequiredService<ILoggerFactory>();
        ILogger<Program> logger = loggerFactory.CreateLogger<Program>();

        using (logger.BeginScope("Logging scope"))
        {
            logger.LogInformation("Hello World!");
            logger.LogInformation("The .NET developer community happily welcomes you.");
        }
    }
}

```

The following console output is similar to what you might expect to see from using this `CustomTimePrefixingFormatter`.

```

|-[ 15:03:15.6179 Hello World! ]>-|
|-[ 15:03:15.6347 The .NET developer community happily welcomes you. ]>-|

```

Implement custom color formatting

In order to properly enable color capabilities in your custom logging formatter, you can extend the [SimpleConsoleFormatterOptions](#) as it has a [SimpleConsoleFormatterOptions.ColorBehavior](#) property that can be useful for enabling colors in logs.

Create a `CustomColorOptions` that derives from `SimpleConsoleFormatterOptions`:

```

using Microsoft.Extensions.Logging.Console;

namespace Console.ExampleFormatters.Custom;

public class CustomColorOptions : SimpleConsoleFormatterOptions
{
    public string? CustomPrefix { get; set; }
}

```

Next, write some extension methods in a `TextWriterExtensions` class that allow for conveniently embedding ANSI coded colors within formatted log messages:

```

namespace Console.ExampleFormatters.Custom;

public static class TextWriterExtensions

```

```

{
    const string DefaultForegroundColor = "\x1B[39m\x1B[22m";
    const string DefaultBackgroundColor = "\x1B[49m";

    public static void WriteWithColor(
        this TextWriter textWriter,
        string message,
        ConsoleColor? background,
        ConsoleColor? foreground)
    {
        // Order:
        // 1. background color
        // 2. foreground color
        // 3. message
        // 4. reset foreground color
        // 5. reset background color

        var backgroundColor = background.HasValue ? GetBackgroundColorEscapeCode(background.Value) : null;
        var foregroundColor = foreground.HasValue ? GetForegroundColorEscapeCode(foreground.Value) : null;

        if (backgroundColor != null)
        {
            textWriter.Write(backgroundColor);
        }
        if (foregroundColor != null)
        {
            textWriter.Write(foregroundColor);
        }

        textWriter.WriteLine(message);

        if (foregroundColor != null)
        {
            textWriter.Write(DefaultForegroundColor);
        }
        if (backgroundColor != null)
        {
            textWriter.Write(DefaultBackgroundColor);
        }
    }
}

static string GetForegroundColorEscapeCode(ConsoleColor color) =>
    color switch
    {
        ConsoleColor.Black => "\x1B[30m",
        ConsoleColor.DarkRed => "\x1B[31m",
        ConsoleColor.DarkGreen => "\x1B[32m",
        ConsoleColor.DarkYellow => "\x1B[33m",
        ConsoleColor.DarkBlue => "\x1B[34m",
        ConsoleColor.DarkMagenta => "\x1B[35m",
        ConsoleColor.DarkCyan => "\x1B[36m",
        ConsoleColor.Gray => "\x1B[37m",
        ConsoleColor.Red => "\x1B[1m\x1B[31m",
        ConsoleColor.Green => "\x1B[1m\x1B[32m",
        ConsoleColor.Yellow => "\x1B[1m\x1B[33m",
        ConsoleColor.Blue => "\x1B[1m\x1B[34m",
        ConsoleColor.Magenta => "\x1B[1m\x1B[35m",
        ConsoleColor.Cyan => "\x1B[1m\x1B[36m",
        ConsoleColor.White => "\x1B[1m\x1B[37m",
        _ => DefaultForegroundColor
    };
}

static string GetBackgroundColorEscapeCode(ConsoleColor color) =>
    color switch
    {
        ConsoleColor.Black => "\x1B[40m",
        ConsoleColor.DarkRed => "\x1B[41m",
        ConsoleColor.DarkGreen => "\x1B[42m",

```

```
ConsoleColor.DarkYellow => "\x1B[43m",
ConsoleColor.DarkBlue => "\x1B[44m",
ConsoleColor.DarkMagenta => "\x1B[45m",
ConsoleColor.DarkCyan => "\x1B[46m",
ConsoleColor.Gray => "\x1B[47m",

_ => DefaultBackgroundColor
};

}
```

A custom color formatter that handles applying custom colors could be defined as follows:

```

using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Logging.Abstractions;
using Microsoft.Extensions.Logging.Console;
using Microsoft.Extensions.Options;

namespace Console.ExampleFormatters.Custom;

public sealed class CustomColorFormatter : ConsoleFormatter, IDisposable
{
    private readonly IDisposable _optionsReloadToken;
    private CustomColorOptions _formatterOptions;

    private bool ConsoleColorFormattingEnabled =>
        _formatterOptions.ColorBehavior == LoggerColorBehavior.Enabled ||
        _formatterOptions.ColorBehavior == LoggerColorBehavior.Default &&
        System.Console.IsOutputRedirected == false;

    public CustomColorFormatter(IOptionsMonitor<CustomColorOptions> options)
        // Case insensitive
        : base("customName") =>
            (_optionsReloadToken, _formatterOptions) =
                (options.OnChange(ReloadLoggerOptions), options.CurrentValue);

    private void ReloadLoggerOptions(CustomColorOptions options) =>
        _formatterOptions = options;

    public override void Write<TState>(
        in LogEntry<TState> logEntry,
        IExternalScopeProvider scopeProvider,
        TextWriter textWriter)
    {
        if (logEntry.Exception is null)
        {
            return;
        }

        string? message =
            logEntry.Formatter?.Invoke(
                logEntry.State, logEntry.Exception);

        if (message is null)
        {
            return;
        }

        CustomLogicGoesHere(textWriter);
        textWriter.WriteLine(message);
    }

    private void CustomLogicGoesHere(TextWriter textWriter)
    {
        if (ConsoleColorFormattingEnabled)
        {
            textWriter.WriteLineWithColor(
                _formatterOptions.CustomPrefix ?? string.Empty,
                ConsoleColor.Black,
                ConsoleColor.Green);
        }
        else
        {
            textWriter.WriteLine(_formatterOptions.CustomPrefix);
        }
    }

    public void Dispose() => _optionsReloadToken?.Dispose();
}

```

When you run the application, the logs will show the `CustomPrefix` message in the color green when `FormatterOptions.ColorBehavior` is `Enabled`.

NOTE

When `LoggerColorBehavior` is `Disabled`, log messages do *not* interpret embedded ANSI color codes within log messages. Instead, they output the raw message. For example, consider the following:

```
logger.LogInformation("Random log \x1B[42mwith green background\x1B[49m message");
```

This would output the verbatim string, and it is *not* colorized.

```
Random log \x1B[42mwith green background\x1B[49m message
```

See also

- [Logging in .NET](#)
- [Implement a custom logging provider in .NET](#)
- [High-performance logging in .NET](#)

.NET Generic Host

9/20/2022 • 7 minutes to read • [Edit Online](#)

The Worker Service templates create a .NET Generic Host, [HostBuilder](#). The Generic Host can be used with other types of .NET applications, such as Console apps.

A *host* is an object that encapsulates an app's resources and lifetime functionality, such as:

- Dependency injection (DI)
- Logging
- Configuration
- App shutdown
- `IHostedService` implementations

When a host starts, it calls `IHostedService.StartAsync` on each implementation of `IHostedService` registered in the service container's collection of hosted services. In a worker service app, all `IHostedService` implementations that contain `BackgroundService` instances have their `BackgroundService.ExecuteAsync` methods called.

The main reason for including all of the app's interdependent resources in one object is lifetime management: control over app startup and graceful shutdown. This is achieved with the [Microsoft.Extensions.Hosting](#) NuGet package.

Set up a host

The host is typically configured, built, and run by code in the `Program` class. The `Main` method:

- Calls a `CreateDefaultBuilder()` method to create and configure a builder object.
- Calls `Build()` to create an `IHost` instance.
- Calls `Run` or `RunAsync` method on the host object.

The .NET Worker Service templates generate the following code to create a Generic Host:

```
IHost host = Host.CreateDefaultBuilder(args)
    .ConfigureServices((hostContext, services) =>
{
    services.AddHostedService<Worker>();
})
.Build();

host.Run();
```

Default builder settings

The `CreateDefaultBuilder` method:

- Sets the content root to the path returned by `GetCurrentDirectory()`.
- Loads `host configuration` from:
 - Environment variables prefixed with `DOTNET_`.
 - Command-line arguments.
- Loads app configuration from:

- *appsettings.json*.
 - *appsettings.{Environment}.json*.
 - Secret Manager when the app runs in the `Development` environment.
 - Environment variables.
 - Command-line arguments.
- Adds the following logging providers:
 - Console
 - Debug
 - EventSource
 - EventLog (only when running on Windows)
 - Enables scope validation and [dependency validation](#) when the environment is `Development`.

The `ConfigureServices` method exposes the ability to add services to the [`Microsoft.Extensions.DependencyInjection.IServiceCollection`](#) instance. Later, these services can be made available from dependency injection.

Framework-provided services

The following services are registered automatically:

- [`IHostApplicationLifetime`](#)
- [`IHostLifetime`](#)
- [`IHostEnvironment`](#)

[`IHostApplicationLifetime`](#)

Inject the [`IHostApplicationLifetime`](#) service into any class to handle post-startup and graceful shutdown tasks. Three properties on the interface are cancellation tokens used to register app start and app stop event handler methods. The interface also includes a [`StopApplication\(\)`](#) method.

The following example is an [`IHostedService`](#) implementation that registers [`IHostApplicationLifetime`](#) events:

```

using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;

namespace AppLifetime.Example;

public sealed class ExampleHostedService : IHostedService
{
    private readonly ILogger _logger;

    public ExampleHostedService(
        ILogger<ExampleHostedService> logger,
        IHostApplicationLifetime appLifetime)
    {
        _logger = logger;

        appLifetime.ApplicationStarted.Register(OnStarted);
        appLifetime.ApplicationStopping.Register(OnStopping);
        appLifetime.ApplicationStopped.Register(OnStopped);
    }

    public Task StartAsync(CancellationToken cancellationToken)
    {
        _logger.LogInformation("1. StartAsync has been called.");

        return Task.CompletedTask;
    }

    public Task StopAsync(CancellationToken cancellationToken)
    {
        _logger.LogInformation("4. StopAsync has been called.");

        return Task.CompletedTask;
    }

    private void OnStarted()
    {
        _logger.LogInformation("2. OnStarted has been called.");
    }

    private void OnStopping()
    {
        _logger.LogInformation("3. OnStopping has been called.");
    }

    private void OnStopped()
    {
        _logger.LogInformation("5. OnStopped has been called.");
    }
}

```

The Worker Service template could be modified to add the `ExampleHostedService` implementation:

```

using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using AppLifetime.Example;

using IHost host = Host.CreateDefaultBuilder(args)
    .ConfigureServices(_> services =>
        services.AddHostedService<ExampleHostedService>())
    .Build();

await host.RunAsync();

```

The application would write the following sample output:

```
// Sample output:  
//   info: ExampleHostedService[0]  
//     1. StartAsync has been called.  
//   info: ExampleHostedService[0]  
//     2. OnStarted has been called.  
//   info: Microsoft.Hosting.Lifetime[0]  
//     Application started.Press Ctrl+C to shut down.  
//   info: Microsoft.Hosting.Lifetime[0]  
//     Hosting environment: Production  
//   info: Microsoft.Hosting.Lifetime[0]  
//     Content root path: ..\app-lifetime\bin\Debug\net6.0  
//   info: ExampleHostedService[0]  
//     3. OnStopping has been called.  
//   info: Microsoft.Hosting.Lifetime[0]  
//     Application is shutting down...  
//   info: ExampleHostedService[0]  
//     4. StopAsync has been called.  
//   info: ExampleHostedService[0]  
//     5. OnStopped has been called.
```

IHostLifetime

The [IHostLifetime](#) implementation controls when the host starts and when it stops. The last implementation registered is used. `Microsoft.Extensions.Hosting.Internal.ConsoleLifetime` is the default [IHostLifetime](#) implementation. For more information on the lifetime mechanics of shutdown, see [Host shutdown](#).

IHostEnvironment

Inject the [IHostEnvironment](#) service into a class to get information about the following settings:

- [IHostEnvironment.ApplicationName](#)
- [IHostEnvironment.ContentRootFileProvider](#)
- [IHostEnvironment.ContentRootPath](#)
- [IHostEnvironment.EnvironmentName](#)

Host configuration

Host configuration is used to configure properties of the [IHostEnvironment](#) implementation.

The host configuration is available in `HostBuilderContext.Configuration` within the `ConfigureAppConfiguration` method. When calling the `ConfigureAppConfiguration` method, the `HostBuilderContext` and `IConfigurationBuilder` are passed into the `configureDelegate`. The `configureDelegate` is defined as an `Action<HostBuilderContext, IConfigurationBuilder>`. The host builder context exposes the `Configuration` property, which is an instance of `IConfiguration`. It represents the configuration built from the host, whereas the `IConfigurationBuilder` is the builder object used to configure the app.

TIP

After `ConfigureAppConfiguration` is called the `HostBuilderContext.Configuration` is replaced with the `app config`.

To add host configuration, call `ConfigureHostConfiguration` on `IHostBuilder`. `ConfigureHostConfiguration` can be called multiple times with additive results. The host uses whichever option sets a value last on a given key.

The following example creates host configuration:

```

using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.Hosting;

using IHost host = Host.CreateDefaultBuilder(args)
    .ConfigureHostConfiguration(configHost =>
{
    configHost.SetBasePath(Directory.GetCurrentDirectory());
    configHost.AddJsonFile("hostsettings.json", optional: true);
    configHost.AddEnvironmentVariables(prefix: "PREFIX_");
    configHost.AddCommandLine(args);
})
.Build();

// Application code should start here.

await host.RunAsync();

```

App configuration

App configuration is created by calling [ConfigureAppConfiguration](#) on `IHostBuilder`.

`ConfigureAppConfiguration` can be called multiple times with additive results. The app uses whichever option sets a value last on a given key.

The configuration created by `ConfigureAppConfiguration` is available in [HostBuilderContext.Configuration](#) for subsequent operations and as a service from DI. The host configuration is also added to the app configuration.

For more information, see [Configuration in .NET](#).

Host shutdown

A hosted service process can be stopped in the following ways:

- If someone doesn't call [Run](#) or [HostingAbstractionsHostExtensions.WaitForShutdown](#) and the app exits normally with `Main` completing.
- If the app crashes.
- If the app is forcefully shut down using [SIGKILL](#) (or [CTRL+Z](#)).

All of these scenarios aren't handled directly by the hosting code. The owner of the process needs to deal with them the same as any application. There are several additional ways in which a hosted service process can be stopped:

- If `ConsoleLifetime` is used, it listens for the following signals and attempts to stop the host gracefully.
 - [SIGINT](#) (or [CTRL+C](#)).
 - [SIGQUIT](#) (or [CTRL+BREAK](#) on Windows, [CTRL+\](#) on Unix).
 - [SIGTERM](#) (sent by other apps, such as `docker stop`).
- If the app calls [Environment.Exit](#).

These scenarios are handled by the built-in hosting logic, specifically the `ConsoleLifetime` class.

`ConsoleLifetime` tries to handle the "shutdown" signals SIGINT, SIGQUIT, and SIGTERM to allow for a graceful exit to the application.

Before .NET 6, there wasn't a way for .NET code to gracefully handle SIGTERM. To work around this limitation, `ConsoleLifetime` would subscribe to [System.AppDomain.ProcessExit](#). When `ProcessExit` was raised, `ConsoleLifetime` would signal the host to stop and block the `ProcessExit` thread, waiting for the host to stop. This would allow for the clean-up code in the application to run — for example, [IHost.StopAsync](#) and code after [HostingAbstractionsHostExtensions.Run](#) in the `Main` method.

This caused other issues because SIGTERM wasn't the only way `ProcessExit` was raised. It is also raised by code in the application calling `Environment.Exit`. `Environment.Exit` isn't a graceful way of shutting down a process in the `Microsoft.Extensions.Hosting` app model. It raises the `ProcessExit` event and then exits the process. The end of the `Main` method doesn't get executed. Background and foreground threads are terminated, and `finally` blocks are *not* executed.

Since `ConsoleLifetime` blocked `ProcessExit` while waiting for the host to shut down, this behavior led to [deadlocks](#) from `Environment.Exit` also blocks waiting for the call to `ProcessExit`. Additionally, since the SIGTERM handling was attempting to gracefully shut down the process, `ConsoleLifetime` would set the `ExitCode` to `0`, which [clobbered](#) the user's exit code passed to `Environment.Exit`.

In .NET 6, [POSIX signals](#) are supported and handled. This allows for `ConsoleLifetime` to handle SIGTERM gracefully, and no longer get involved when `Environment.Exit` is invoked.

TIP

For .NET 6+, `ConsoleLifetime` no longer has logic to handle scenario `Environment.Exit`. Apps that call `Environment.Exit` and need to perform clean-up logic can subscribe to `ProcessExit` themselves. Hosting will no longer attempt to gracefully stop the host in this scenario.

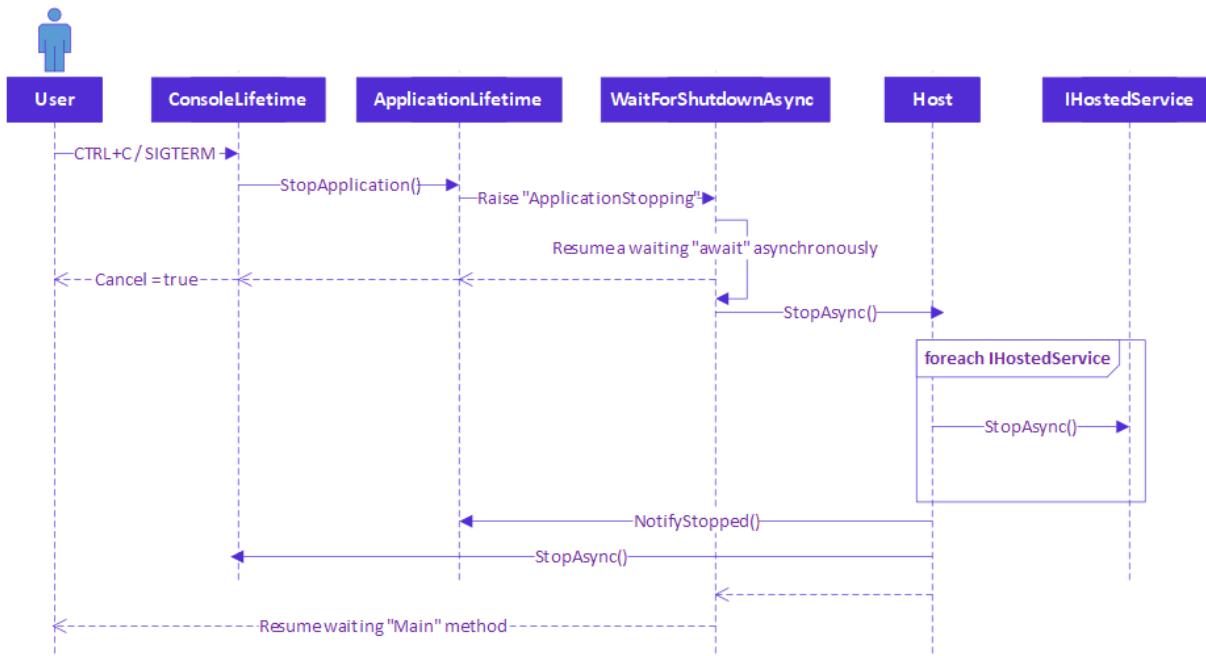
If your application uses hosting, and you want to gracefully stop the host, you can call `IHostApplicationLifetime.StopApplication` instead of `Environment.Exit`.

Hosting shutdown process

The following sequence diagram shows how the signals are handled internally in the hosting code. Most users don't need to understand this process. But for developers that need a deep understanding, this may help you get started.

After the host has been started, when a user calls `Run` or `WaitForShutdown`, a handler gets registered for `IApplicationLifetime.ApplicationStopping`. Execution is paused in `WaitForShutdown`, waiting for the `ApplicationStopping` event to be raised. This is how the `Main` method doesn't return right away, and the app stays running until `Run` or `WaitForShutdown` returns.

When a signal is sent to the process, it initiates the following sequence:



1. The control flows from `ConsoleLifetime` to `ApplicationLifetime` to raise the `ApplicationStopping` event. This signals `WaitForShutdownAsync` to unblock the `Main` execution code. In the meantime, the POSIX signal handler returns with `cancel = true` since this POSIX signal has been handled.
2. The `Main` execution code starts executing again and tells the host to `StopAsync()`, which in turn stops all the hosted services, and raises any other stopped events.
3. Finally, `WaitForShutdown` exits, allowing for any application clean up code to execute, and for the `Main` method to exit gracefully.

See also

- [Dependency injection in .NET](#)
- [Logging in .NET](#)
- [Configuration in .NET](#)
- [Worker Services in .NET](#)
- [ASP.NET Core Web Host](#)
- Generic host bugs should be created in the github.com/dotnet/extensions repo

Network programming in .NET

9/20/2022 • 2 minutes to read • [Edit Online](#)

.NET provides a layered, extensible, and managed implementation of Internet services that can be quickly and easily integrated into your apps. Your network apps can build on pluggable protocols to automatically take advantage of various Internet protocols, or they can use a managed implementation of cross-platform socket interfaces to work with the network on the socket level.

Internet apps

Internet apps can be classified broadly into two kinds: client apps that request information and server apps that respond to information requests from clients. The classic Internet client-server app is the World Wide Web, where people use browsers to access documents and other data stored on web servers worldwide.

Apps are not limited to just one of these roles; for instance, the familiar middle-tier app server responds to requests from clients by requesting data from another server, in which case it is acting as both a server and a client.

The client app requests by identifying the requested Internet resource and the communication protocol to use for the request and response. If necessary, the client also provides any additional data required to complete the request, such as proxy location or authentication information (user name, password, and so on). Once the request is formed, the request can be sent to the server.

Identifying resources

.NET uses a uniform resource identifier (URI) to identify the requested Internet resource and communication protocol. The URI consists of at least three, and possibly four, fragments: the scheme identifier, which identifies the communications protocol for the request and response; the server identifier, which consists of either a domain name system (DNS) hostname or a TCP address that uniquely identifies the server on the Internet; the path identifier, which locates the requested information on the server; and an optional query string, which passes information from the client to the server.

The [System.Uri](#) type is used as a representation of a uniform resource identifier (URI) and easy access to the parts of the URI. To create a `Uri` instance you can pass it a string:

```
const string uriString =
    "https://docs.microsoft.com/en-us/dotnet/path?key=value#bookmark";

Uri canonicalUri = new(UriString);
Console.WriteLine(canonicalUri.Host);
Console.WriteLine(canonicalUri.PathAndQuery);
Console.WriteLine(canonicalUri.Fragment);
// Sample output:
//     docs.microsoft.com
//     /en-us/dotnet/path?key=value
//     #bookmark
```

The `Uri` class automatically performs validation and canonicalization per [RFC 3986](#). These validation and canonicalization rules are used to ensure that a URI is well-formed and that the URI is in a canonical form. However, this modifies some URIs in a way that might break their end-users.

To bypass these validation rules, set [UriCreationOptions.DangerousDisablePathAndQueryCanonicalization](#) to

`true`, which means no validation and no transformation of the input will take place past the authority. As a side effect, `Uri` instances created with this option don't support `Uri.Fragments` and this property will always be empty. Moreover, `Uri.GetComponents` may not be used for `UriComponents.Path` or `UriComponents.Query` and will throw `InvalidOperationException`.

```
const string uriString =
    "https://localhost:5001/path%4A?query%4A#/foo";

UriCreationOptions creationOptions = new()
{
    DangerousDisablePathAndQueryCanonicalization = true
};

Uri uri = new(uriString, creationOptions);
Console.WriteLine(uri);
Console.WriteLine(uri.AbsolutePath);
Console.WriteLine(uri.Query);
Console.WriteLine(uri.PathAndQuery);
Console.WriteLine(uri.Fragment);
// Sample output:
//     https://localhost:5001/path%4A?query%4A#/foo
//     /path%4A
//     ?query%4A#/foo
//     /path%4A?query%4A#/foo

Uri canonicalUri = new(uriString);
Console.WriteLine(canonicalUri.PathAndQuery);
Console.WriteLine(canonicalUri.Fragment);
// Sample output:
//     /pathJ?queryJ
//     #/foo
```

WARNING

Disabling canonicalization also means that reserved characters will not be escaped (for example, space characters will not be changed to `%20`), which may corrupt the HTTP request and makes the application subject to request smuggling. Only set this option if you have ensured that the URI string is already sanitized.

See also

- [Runtime configuration options for networking](#)
- [HTTP support in .NET](#)
- [Sockets in .NET](#)
- [TCP in .NET](#)
- [Tutorial: Make HTTP requests in a .NET console app using C#](#)
- [.NET Networking improvements](#)

Network availability

9/20/2022 • 2 minutes to read • [Edit Online](#)

The [System.Net.NetworkInformation](#) namespace enables you to gather information about network events, changes, statistics, and properties. In this article, you'll learn how to use the [System.Net.NetworkInformation.NetworkChange](#) class to determine whether the network address or availability has changed. Additionally, you'll see about the network statistics and properties on an interface or protocol basis. Finally, you'll use the [System.Net.NetworkInformation.Ping](#) class to determine whether a remote host is reachable.

Network change events

The [System.Net.NetworkInformation.NetworkChange](#) class enables you to determine whether the network address or availability has changed. To use this class, create an event handler to process the change, and associate it with a [NetworkAddressChangedEventHandler](#) or a [NetworkAvailabilityChangedEventHandler](#).

```
NetworkChange.NetworkAvailabilityChanged += OnNetworkAvailabilityChanged;

static void OnNetworkAvailabilityChanged(
    object? sender, NetworkAvailabilityEventArgs networkAvailability) =>
    Console.WriteLine($"Network is available: {networkAvailability.IsAvailable}");

Console.WriteLine(
    "Listening changes in network availability. Press any key to continue.");
Console.ReadLine();

NetworkChange.NetworkAvailabilityChanged -= OnNetworkAvailabilityChanged;
```

The preceding C# code:

- Registers an event handler for the [NetworkChange.NetworkAvailabilityChanged](#) event.
- The event handler simply writes the availability status to the console.
- A message is written to the console letting the user know that the code is listening for changes in network availability and waits for a key press to exit.
- Unregisters the event handler.

```

NetworkChange.NetworkAddressChanged += OnNetworkAddressChanged;

static void OnNetworkAddressChanged(
    object? sender, EventArgs args)
{
    foreach ((string name, OperationalStatus status) in
        NetworkInterface.GetAllNetworkInterfaces()
        .Select(networkInterface =>
            (networkInterface.Name, networkInterface.OperationalStatus)))
    {
        Console.WriteLine(
            $"{name} is {status}");
    }
}

Console.WriteLine(
    "Listening for address changes. Press any key to continue.");
Console.ReadLine();

NetworkChange.NetworkAddressChanged -= OnNetworkAddressChanged;

```

The preceding C# code:

- Registers an event handler for the [NetworkChange.NetworkAddressChanged](#) event.
- The event handler iterates over [NetworkInterface.GetAllNetworkInterfaces\(\)](#), writing their name and operational status to the console.
- A message is written to the console letting the user know that the code is listening for changes in network availability and waits for a key press to exit.
- Unregisters the event handler.

Network statistics and properties

You can gather network statistics and properties on an interface or protocol basis. The [NetworkInterface](#), [NetworkInterfaceType](#), and [PhysicalAddress](#) classes give information about a particular network interface, while the [IPInterfaceProperties](#), [IPGlobalProperties](#), [IPGlobalStatistics](#), [TcpStatistics](#), and [UdpStatistics](#) classes give information about layer 3 and layer 4 packets.

```

ShowStatistics(NetworkInterfaceComponent.IPv4);
ShowStatistics(NetworkInterfaceComponent.IPv6);

static void ShowStatistics(NetworkInterfaceComponent version)
{
    var properties = IPGlobalProperties.GetIPGlobalProperties();
    var stats = version switch
    {
        NetworkInterfaceComponent.IPv4 => properties.GetTcpIPv4Statistics(),
        _ => properties.GetTcpIPv6Statistics()
    };

    Console.WriteLine($"TCP/{version} Statistics");
    Console.WriteLine($" Minimum Transmission Timeout : {stats.MinimumTransmissionTimeout:#,#}");
    Console.WriteLine($" Maximum Transmission Timeout : {stats.MaximumTransmissionTimeout:#,#}");
    Console.WriteLine(" Connection Data");
    Console.WriteLine($" Current : {stats.CurrentConnections:#,#}");
    Console.WriteLine($" Cumulative : {stats.CumulativeConnections:#,#}");
    Console.WriteLine($" Initiated : {stats.ConnectionsInitiated:#,#}");
    Console.WriteLine($" Accepted : {stats.ConnectionsAccepted:#,#}");
    Console.WriteLine($" Failed Attempts : {stats.FailedConnectionAttempts:#,#}");
    Console.WriteLine($" Reset : {stats.ResetConnections:#,#}");
    Console.WriteLine(" Segment Data");
    Console.WriteLine($" Received : {stats.SegmentsReceived:#,#}");
    Console.WriteLine($" Sent : {stats.SegmentsSent:#,#}");
    Console.WriteLine($" Retransmitted : {stats.SegmentsResent:#,#}");
    Console.WriteLine();
}

```

The preceding C# code:

- Calls a custom `ShowStatistics` method to display the statistics for each protocol.
- The `ShowStatistics` method calls `IPGlobalProperties.GetIPGlobalProperties()`, and depending on the given `NetworkInterfaceComponent` will either call `IPGlobalProperties.GetIPv4GlobalStatistics()` or `IPGlobalProperties.GetIPv6GlobalStatistics()`.
- The `TcpStatistics` are written to the console.

Determine if a remote host is reachable

You can use the `Ping` class to determine whether a remote host is up, on the network, and reachable.

```

using Ping ping = new();

string hostName = "stackoverflow.com";
PingReply reply = await ping.SendPingAsync(hostName);
Console.WriteLine($"Ping status for ({hostName}): {reply.Status}");
if (reply is { Status: IPStatus.Success })
{
    Console.WriteLine($"Address: {reply.Address}");
    Console.WriteLine($"Roundtrip time: {reply.RoundtripTime}");
    Console.WriteLine($"Time to live: {reply.Options?.Ttl}");
    Console.WriteLine();
}

```

The preceding C# code:

- Instantiate a `Ping` object.
- Calls `Ping.SendPingAsync(String)` with the `"stackoverflow.com"` hostname parameter.
- The status of the ping is written to the console.

See also

- [Network programming in .NET](#)
- [NetworkInterface](#)

Internet Protocol version 6 (IPv6) overview

9/20/2022 • 8 minutes to read • [Edit Online](#)

The Internet Protocol version 6 (IPv6) is a suite of standard protocols for the network layer of the Internet. IPv6 is designed to solve many of the problems of the current version of the Internet Protocol suite (known as IPv4) about address depletion, security, auto-configuration, extensibility, and so on. IPv6 expands the capabilities of the Internet to enable new kinds of applications, including peer-to-peer and mobile applications. The following are the main issues of the current IPv4 protocol:

- Rapid depletion of the address space.

This has led to the use of Network Address Translators (NATs) that map multiple private addresses to a single public IP address. The main problems created by this mechanism are processing overhead and lack of end-to-end connectivity.

- Lack of hierarchy support.

Because of its inherent predefined class organization, IPv4 lacks true hierarchical support. It is impossible to structure the IP addresses in a way that truly maps the network topology. This crucial design flaw creates the need for large routing tables to deliver IPv4 packets to any location on the Internet.

- Complex network configuration.

With IPv4, addresses must be assigned statically or using a configuration protocol such as DHCP. In an ideal situation, hosts would not have to rely on the administration of a DHCP infrastructure. Instead, they would be able to configure themselves based on the network segment in which they are located.

- Lack of built-in authentication and confidentiality.

IPv4 does not require support for any mechanism that provides authentication or encryption of the exchanged data. This changes with IPv6. Internet Protocol security (IPSec) is an IPv6 support requirement.

A new protocol suite must satisfy the following basic requirements:

- Large-scale routing and addressing with low overhead.
- Auto-configuration for various connecting situations.
- Built-in authentication and confidentiality.

IPv6 addressing

With IPv6, addresses are 128 bits long. One reason for such a large address space is to subdivide the available addresses into a hierarchy of routing domains that reflect the Internet's topology. Another reason is to map the addresses of network adapters (or interfaces) that connect devices to the network. IPv6 features an inherent capability to resolve addresses at their lowest level, which is at the network interface level, and also has auto-configuration capabilities.

Text representation

The following are the three conventional forms used to represent the IPv6 addresses as text strings:

- **Colon-hexadecimal form:**

This is the preferred form `n:n:n:n:n:n:n:n`. Each `n` represents the hexadecimal value of one of the eight 16-bit elements of the address. For example: `3FFE:FFFF:7654:FEDA:1245:BA98:3210:4562`.

- **Compressed form:**

Due to the address length, it is common to have addresses containing a long string of zeros. To simplify writing these addresses, use the compressed form, in which a single contiguous sequence of 0 blocks is represented by a double-colon symbol (::). This symbol can appear only once in an address. For example, the multicast address FFED:0:0:0:0:BA98:3210:4562 in compressed form is

FFED::BA98:3210:4562 . The unicast address 3FFE:FFFF:0:0:8:800:20C4:0 in compressed form is 3FFE:FFFF::8:800:20C4:0 . The loopback address 0:0:0:0:0:0:1 in compressed form is ::1 . The unspecified address 0:0:0:0:0:0:0 in compressed form is :: .

- **Mixed form:**

This form combines IPv4 and IPv6 addresses. In this case, the address format is n:n:n:n:n:n:d.d.d.d , where each n represents the hexadecimal values of the six IPv6 high-order 16-bit address elements, and each d represents the decimal value of an IPv4 address.

Address types

The leading bits in the address define the specific IPv6 address type. The variable-length field containing these leading bits is called a Format Prefix (FP).

An IPv6 unicast address is divided into two parts. The first part contains the address prefix, and the second part contains the interface identifier. A concise way to express an IPv6 address/prefix combination is as follows: ipv6-address/prefix-length.

The following is an example of an address with a 64-bit prefix.

3FFE:FFFF:0:CD30:0:0:0:0/64 .

The prefix in this example is 3FFE:FFFF:0:CD30 . The address can also be written in a compressed form, as 3FFE:FFFF:0:CD30::/64 .

IPv6 defines the following address types:

- **Unicast address:**

An identifier for a single interface. A packet sent to this address is delivered to the identified interface. The unicast addresses are distinguished from the multicast addresses by the value of the high-order octet.

The multicast addresses' high-order octet has the hexadecimal value of FF. Any other value for this octet identifies a unicast address. The following are different types of unicast addresses:

- **Link-local addresses:**

These addresses are used on a single link and have the following format: FE80::*InterfaceID* .

Link-local addresses are used between nodes on a link for auto-address configuration, neighbor discovery, or when no routers are present. A link-local address is used primarily at startup and when the system has not yet acquired addresses of larger scope.

- **Site-local addresses:**

These addresses are used on a single site and have the following format: FEC0::*SubnetID*:*InterfaceID* . The site-local addresses are used for addressing inside a site without the need for a global prefix.

- **Global IPv6 unicast addresses:**

These addresses can be used across the Internet and have the following format:

010(FP, 3 bits) TLA ID (13 bits) Reserved (8 bits) NLA ID (24 bits) SLA ID (16 bits) *InterfaceID* (64 bits)

- **Multicast address:**

An identifier for a set of interfaces (typically belonging to different nodes). A packet sent to this address is delivered to all the interfaces identified by the address. The multicast address types supersede the IPv4 broadcast addresses.

- **Anycast address:**

An identifier for a set of interfaces (typically belonging to different nodes). A packet sent to this address is delivered to only one interface identified by the address. This is the nearest interface as identified by routing metrics. Anycast addresses are taken from the unicast address space and are not syntactically distinguishable. The addressed interface performs the distinction between unicast and anycast addresses as a function of its configuration.

In general, a node always has a link-local address. It might have a site-local address and one or more global addresses.

IPv6 routing

A flexible routing mechanism is a benefit of IPv6. Due to how IPv4 network IDs were and are allocated, large routing tables need to be maintained by the routers that are on the Internet backbones. These routers must know all the routes to forward packets that are potentially directed to any node on the Internet. With its ability to aggregate addresses, IPv6 allows flexible addressing and drastically reduces the size of routing tables. In this new addressing architecture, intermediate routers must keep track only of the local portion of their network to forward the messages appropriately.

Neighbor discovery

Some of the features provided by *neighbor discovery* are:

- **Router discovery:** This allows hosts to identify local routers.
- **Address resolution:** This allows nodes to resolve a link-layer address for a corresponding next-hop address (a replacement for Address Resolution Protocol [ARP]).
- **Address auto-configuration:** This allows hosts to automatically configure site-local and global addresses.

Neighbor discovery uses Internet Control Message Protocol for IPv6 (ICMPv6) messages that include:

- **Router advertisement:** Sent by a router on a pseudo-periodic basis or in response to a router solicitation. IPv6 routers use router advertisements to advertise their availability, address prefixes, and other parameters.
- **Router solicitation:** Sent by a host to request that routers on the link send a router advertisement immediately.
- **Neighbor solicitation:** Sent by nodes for address resolution, duplicate address detection, or to verify that a neighbor is still reachable.
- **Neighbor advertisement:** Sent by nodes to respond to a neighbor solicitation or to notify neighbors of a change in link-layer address.
- **Redirect:** Sent by routers to indicate a better next-hop address to a particular destination for a sending node.

IPv6 auto-configuration

One important goal for IPv6 is to support node Plug and Play. That is, it should be possible to plug a node into an IPv6 network and have it automatically configured without any human intervention.

Auto-configuration types

IPv6 supports the following types of auto-configuration:

- **Stateful auto-configuration:**

This type of configuration requires a certain level of human intervention because it needs a Dynamic

Host Configuration Protocol for IPv6 (DHCPv6) server for the installation and administration of the nodes. The DHCPv6 server keeps a list of nodes to which it supplies configuration information. It also maintains state information so the server knows how long each address is in use, and when it might be available for reassignment.

- **Stateless auto-configuration:**

This type of configuration is suitable for small organizations and individuals. In this case, each host determines its addresses from the contents of received router advertisements. Using the IEEE EUI-64 standard to define the network ID portion of the address, it is reasonable to assume the uniqueness of the host address on the link.

Regardless of how the address is determined, the node must verify that its potential address is unique to the local link. This is done by sending a neighbor solicitation message to the potential address. If the node receives any response, it knows that the address is already in use and must determine another address.

IPv6 mobility

The proliferation of mobile devices has introduced a new requirement: A device must be able to arbitrarily change locations on the IPv6 Internet and still maintain existing connections. To provide this functionality, a mobile node is assigned a home address at which it can always be reached. When the mobile node is at home, it connects to the home link and uses its home address. When the mobile node is away from home, a home agent, which is usually a router, relays messages between the mobile node and the nodes with which it is communicating.

Disable or enable IPv6

To use the IPv6 protocol, ensure that you are running a version of the operating system that supports IPv6 and ensure that the operating system and the networking classes are configured properly.

Configuration Steps

The following table lists various configurations

OS IPV6 ENABLED?	CODE IPV6 ENABLED?	DESCRIPTION
✗ No	✗ No	Can parse IPv6 addresses.
✗ No	✓ Yes	Can parse IPv6 addresses.
✓ Yes	✗ No	Can parse IPv6 addresses and resolve IPv6 addresses using name resolution methods not marked obsolete.
✓ Yes	✓ Yes	Can parse and resolve IPv6 addresses using all methods including those marked obsolete.

IPv6 is enabled by default. To configure this switch in an environment variable, use the `DOTNET_SYSTEM_NET_DISABLEIPV6` environment variable. For more information, see [.NET environment variables](#): `DOTNET_SYSTEM_NET_DISABLEIPV6`.

See also

- [Networking in .NET](#)
- [Sockets in .NET](#)
- [System.AppContext](#)

HTTP support in .NET

9/20/2022 • 3 minutes to read • [Edit Online](#)

Hypertext Transfer Protocol (or HTTP) is a protocol for requesting resources from a web server. The [System.Net.Http.HttpClient](#) class exposes the ability to send HTTP requests and receive HTTP responses from a resource identified by a URI. There are many types of resources that are available on the web, and HTTP defines a set of request methods for accessing these resources.

HTTP request methods

The request methods are differentiated via several factors, first by their *verb* but also by the following characteristics:

- A request method is *idempotent* if it can be successfully processed multiple times without changing the result. For more information, see [RFC 7231: Section 4.2.2 Idempotent Methods](#).
- A request method is *cacheable* when its corresponding response can be stored for reuse. For more information, see [RFC 7231: Section 4.2.3 Cacheable Methods](#).
- A request method is considered a *safe method* if it doesn't modify the state of a resource. All *safe methods* are also *idempotent*, but not all *idempotent* methods are considered *safe*. For more information, see [RFC 7231: Section 4.2.1 Safe Methods](#).

HTTP VERB	IS IDEMPOTENT	IS CACHEABLE	IS SAFE
GET	✓ Yes	✓ Yes	✓ Yes
POST	✗ No	⚠ [†] Rarely	✗ No
PUT	✓ Yes	✗ No	✗ No
PATCH	✗ No	✗ No	✗ No
DELETE	✓ Yes	✗ No	✗ No
HEAD	✓ Yes	✓ Yes	✓ Yes
OPTIONS	✓ Yes	✗ No	✓ Yes
TRACE	✓ Yes	✗ No	✓ Yes
CONNECT	✗ No	✗ No	✗ No

[†]The POST method is only cacheable when the appropriate Cache-Control or Expires response headers are present. This is very uncommon in practice.

HTTP status codes

.NET provides comprehensive support for the HTTP protocol, which makes up the majority of all internet traffic, with the [HttpClient](#). For more information, see [Make HTTP requests with the HttpClient class](#). Applications

receive HTTP protocol errors by catching an [HttpRequestException](#) with the [HttpRequestException.StatusCode](#) set to a corresponding [HttpStatusCode](#). The [HttpResponseMessage](#) contains a [HttpResponseMessage.StatusCode](#) property that can be used to determine non-error status codes. For more information, see [RFC 9110, HTTP Semantics: Status Codes](#).

Informational status codes

The informational status codes reflect an interim response. The client should continue to use the same request and discard the response.

HTTP STATUS CODE	HTTPSTATUSCODE
100	HttpStatusCode.Continue
101	HttpStatusCode.SwitchingProtocols
102	HttpStatusCode.Processing
103	HttpStatusCode.EarlyHints

Successful status codes

The successful status codes indicate that the client's request was successfully received, understood, and accepted.

HTTP STATUS CODE	HTTPSTATUSCODE
200	HttpStatusCode.OK
201	HttpStatusCode.Created
202	HttpStatusCode.Accepted
203	HttpStatusCode.NonAuthoritativeInformation
204	HttpStatusCode.NoContent
205	HttpStatusCode.ResetContent
206	HttpStatusCode.PartialContent
207	HttpStatusCode.MultiStatus
208	HttpStatusCode.AlreadyReported
226	HttpStatusCode.IMUsed

Redirection status codes

Redirection status codes require the user agent to take action in order to fulfill the request. With the appropriate headers, automatic redirection is possible.

HTTP STATUS CODE	HTTPSTATUSCODE
300	HttpStatusCode.MultipleChoices or HttpStatusCode.Ambiguous
301	HttpStatusCode.MovedPermanently or HttpStatusCode.Moved
302	HttpStatusCode.Found or HttpStatusCode.Redirect
303	HttpStatusCode.SeeOther or HttpStatusCode.RedirectMethod
304	HttpStatusCode.NotModified
305	HttpStatusCode.UseProxy
306	HttpStatusCode.Unused
307	HttpStatusCode.TemporaryRedirect or HttpStatusCode.RedirectKeepVerb
308	HttpStatusCode.PermanentRedirect

Client error status codes

The client error status codes indicate that the client's request was invalid.

HTTP STATUS CODE	HTTPSTATUSCODE
400	HttpStatusCode.BadRequest
401	HttpStatusCode.Unauthorized
402	HttpStatusCode.PaymentRequired
403	HttpStatusCode.Forbidden
404	HttpStatusCode.NotFound
405	HttpStatusCode.MethodNotAllowed
406	HttpStatusCode.NotAcceptable
407	HttpStatusCode.ProxyAuthenticationRequired
408	HttpStatusCode.RequestTimeout
409	HttpStatusCode.Conflict
410	HttpStatusCode.Gone

HTTP STATUS CODE	HTTPSTATUSCODE
411	HttpStatusCode.LengthRequired
412	HttpStatusCode.PreconditionFailed
413	HttpStatusCode.RequestEntityTooLarge
414	HttpStatusCode.RequestUriTooLong
415	HttpStatusCode.UnsupportedMediaType
416	HttpStatusCode.RequestedRangeNotSatisfiable
417	HttpStatusCode.ExpectationFailed
418	I'm a teapot ☐
421	HttpStatusCode.MisdirectedRequest
422	HttpStatusCode.UnprocessableEntity
423	HttpStatusCode.Locked
424	HttpStatusCode.FailedDependency
426	HttpStatusCode.UpgradeRequired
428	HttpStatusCode.PreconditionRequired
429	HttpStatusCode.TooManyRequests
431	HttpStatusCode.RequestHeaderFieldsTooLarge
451	HttpStatusCode.UnavailableForLegalReasons

Server error status codes

The server error status codes indicate that the server encountered an unexpected condition that prevented it from fulfilling the request.

HTTP STATUS CODE	HTTPSTATUSCODE
500	HttpStatusCode.InternalServerError
501	HttpStatusCode.NotImplemented
502	HttpStatusCode.BadGateway
503	HttpStatusCode.ServiceUnavailable

HTTP STATUS CODE	HTTPSTATUSCODE
504	HttpStatusCode.GatewayTimeout
505	HttpStatusCode.HttpVersionNotSupported
506	HttpStatusCode.VariantAlsoNegotiates
507	HttpStatusCode.InsufficientStorage
508	HttpStatusCode.LoopDetected
510	HttpStatusCode.NotExtended
511	HttpStatusCode.NetworkAuthenticationRequired

See also

- [Make HTTP requests with the HttpClient class](#)
- [IHttpClientFactory with .NET](#)
- [Guidelines for using HttpClient](#)
- [.NET Networking improvements](#)

Guidelines for using HttpClient

9/20/2022 • 2 minutes to read • [Edit Online](#)

The [System.Net.Http.HttpClient](#) class sends HTTP requests and receives HTTP responses from a resource identified by a URI. An [HttpClient](#) instance is a collection of settings that's applied to all requests executed by that instance, and each instance uses its own connection pool, which isolates its requests from others. Starting in .NET Core 2.1, the [SocketsHttpHandler](#) class provides the implementation, making behavior consistent across all platforms.

If you're using .NET 5+ (including .NET Core), there are some considerations to keep in mind if you're using [HttpClient](#).

DNS behavior

[HttpClient](#) only resolves DNS entries when a connection is created. It does not track any time to live (TTL) durations specified by the DNS server. If DNS entries change regularly, which can happen in some container scenarios, the client won't respect those updates. To solve this issue, you can limit the lifetime of the connection by setting the [PooledConnectionLifetime](#) property, so that DNS lookup is required when the connection is replaced. Consider the following example:

```
var handler = new SocketsHttpHandler
{
    PooledConnectionLifetime = TimeSpan.FromMinutes(15) // Recreate every 15 minutes
};
var client = new HttpClient(handler);
```

The preceding `HttpClient` is configured to reuse connections for 15 minutes. After the timespan specified by [PooledConnectionLifetime](#) has elapsed, the connection is closed and a new one is created.

Pooled connections

The connection pool for an [HttpClient](#) is linked to its underlying [HttpMessageHandler](#). When the [HttpClient](#) instance is disposed, it disposes all previously used connections. If you later send a request to the same server, a new connection is created. There's also a performance penalty because it needs a new TCP port. If the rate of requests is high, or if there are any firewall limitations, that can **exhaust the available sockets** because of default TCP cleanup timers.

Recommended use

- In .NET Core and .NET 5+:
 - Use a `static` or *singleton* `HttpClient` instance with [PooledConnectionLifetime](#) set to the desired interval, such as two minutes, depending on expected DNS changes. This solves both the socket exhaustion and DNS changes problems without adding the overhead of [IHttpClientFactory](#). If you need to be able to mock your handler, you can register it separately.
 - Using [IHttpClientFactory](#), you can have multiple, differently configured clients for different use cases. However, be aware that the factory-created clients are intended to be short-lived, and once the client is created, the factory no longer has control over it.

The factory pools [HttpMessageHandler](#) instances, and, if its lifetime hasn't expired, a handler can

be reused from the pool when the factory creates a new [HttpClient](#) instance. This reuse avoids any socket exhaustion issues.

If you desire the configurability that [IHttpClientFactory](#) provides, we recommend using the [typed-client approach](#).

- In .NET Framework, use [IHttpClientFactory](#) to manage your `HttpClient` instances. If you create a new client instance for each request, you can exhaust available sockets.

TIP

If your app requires cookies, consider disabling automatic cookie handling or avoiding [IHttpClientFactory](#). Pooling the [HttpMessageHandler](#) instances results in sharing of [CookieContainer](#) objects. Unanticipated [CookieContainer](#) object sharing often results in incorrect code.

See also

- [HTTP support in .NET](#)
- [Create HttpClient instances using IHttpClientFactory](#)
- [Make HTTP requests with the HttpClient](#)
- [Use IHttpClientFactory to implement resilient HTTP requests](#)

Make HTTP requests with the HttpClient class

9/20/2022 • 16 minutes to read • [Edit Online](#)

In this article, you'll learn how to make HTTP requests and handle responses with the `HttpClient` class.

IMPORTANT

All of the example HTTP requests target one of the following URLs:

- <https://jsonplaceholder.typicode.com>: Free fake API for testing and prototyping.
- <https://www.example.com>: This domain is for use in illustrative examples in documents.

HTTP endpoints commonly return JavaScript Object Notation (JSON) data, but not always. For convenience, the optional [System.Net.Http.Json](#) NuGet package provides several extension methods for `HttpClient` and `HttpContent` that perform automatic serialization and deserialization using `System.Text.Json`. The examples that follow call attention to places where these extensions are available.

TIP

All of the source code from this article is available in the [GitHub: .NET Docs](#) repository.

Create an `HttpClient`

Most of the following examples reuse the same `HttpClient` instance, and therefore only need to be configured once. To create an `HttpClient`, use the `HttpClient` class constructor. For more information, see [Guidelines for using HttpClient](#).

```
using HttpClient todoClient = new()
{
    BaseAddress = new Uri("https://jsonplaceholder.typicode.com")
};
```

The preceding code:

- Instantiates a new `HttpClient` instance with the object initializer syntax, and as a [using declaration](#).
- Sets the `HttpClient.BaseAddress` to `"https://jsonplaceholder.typicode.com"`.

This `HttpClient` instance will always use the base address when making subsequent requests. To apply additional configuration consider:

- Setting `HttpClient.DefaultRequestHeaders`.
- Applying a non-default `HttpClient.Timeout`.
- Specifying the `HttpClient.DefaultRequestVersion`.

TIP

Alternatively, you can create `HttpClient` instances using a factory-pattern approach that allows you to configure any number of clients and consume them as dependency injection services. For more information, see [IHttpClientFactory with .NET](#).

Make an HTTP request

To make an HTTP request, you call any of the following APIs:

HTTP VERB	API
GET	<code>HttpClient.GetAsync</code>
GET	<code>HttpClient.GetByteArrayAsync</code>
GET	<code>HttpClient.GetStreamAsync</code>
GET	<code>HttpClient.GetStringAsync</code>
POST	<code>HttpClient.PostAsync</code>
PUT	<code>HttpClient.PutAsync</code>
PATCH	<code>HttpClient.PatchAsync</code>
DELETE	<code>HttpClient.DeleteAsync</code>
+ USER SPECIFIED	<code>HttpClient.SendAsync</code>

[†]A `USER SPECIFIED` request indicates that the `SendAsync` method accepts any valid `HttpMethod`.

WARNING

Making HTTP requests is considered network I/O-bound work. While there is a synchronous `HttpClient.Send` method, it is recommended to use the asynchronous APIs instead, unless you have good reason not to.

HTTP content

The `HttpContent` type is used to represent an HTTP entity body and corresponding content headers. For HTTP verbs (or request methods) that require a body, `POST`, `PUT`, and `PATCH`, you use the `HttpContent` class to specify the body of the request. Most examples show how to prepare the `StringContent` subclass with a JSON payload, but additional subclasses exist for different `content (MIME) types`.

- `ByteArrayContent`: Provides HTTP content based on a byte array.
- `FormUrlEncodedContent`: Provides HTTP content for name/value tuples encoded using `"application/x-www-form-urlencoded"` MIME type.
- `JsonContent`: Provides HTTP content based on JSON.
- `MultipartContent`: Provides a collection of `HttpContent` objects that get serialized using the `"multipart/*"` MIME type specification.
- `MultipartFormDataContent`: Provides a container for content encoded using `"multipart/form-data"` MIME type.
- `ReadOnlyMemoryContent`: Provides HTTP content based on a `ReadOnlyMemory<T>`.
- `StreamContent`: Provides HTTP content based on a stream.
- `StringContent`: Provides HTTP content based on a string.

The `HttpContent` class is also used to represent the response body of the `HttpResponseMessage`, accessible on

the `HttpResponseMessage.Content` property.

HTTP Get

A `GET` request shouldn't send a body and is used (as the verb indicates) to retrieve (or get) data from a resource. To make an HTTP `GET` request, given an `HttpClient` and a URI, use the `HttpClient.GetAsync` method:

```
static async Task GetAsync(HttpClient client)
{
    using HttpResponseMessage response = await client.GetAsync("todos/3");

    response.EnsureSuccessStatusCode()
        .WriteRequestToConsole();

    var jsonResponse = await response.Content.ReadAsStringAsync();
    WriteLine($"{{jsonResponse}}\n");

    // Expected output:
    //   GET https://jsonplaceholder.typicode.com/todos/3 HTTP/ 1.1
    //   {
    //     "userId": 1,
    //     "id": 3,
    //     "title": "fugiat veniam minus",
    //     "completed": false
    //   }
}
```

The preceding code:

- Makes a `GET` request to `"https://jsonplaceholder.typicode.com/todos/3"`.
- Ensures that the response is successful.
- Writes the request details to the console.
- Reads the response body as a string.
- Writes the JSON response body to the console.

The `WriteRequestToConsole` is a custom extension method that isn't part of the framework, but if you're curious how it's written, consider the following C# code:

```
static class HttpResponseMessageExtensions
{
    internal static void WriteRequestToConsole(this HttpResponseMessage response)
    {
        if (response == null)
        {
            return;
        }

        var request = response.RequestMessage;
        Write($"{request?.Method} ");
        Write($"{request?.RequestUri} ");
        WriteLine($"HTTP/{request?.Version}");
    }
}
```

HTTP Get from JSON

The <https://jsonplaceholder.typicode.com/todos> endpoint returns a JSON array of "todo" objects. Their JSON structure resembles the following:

```
[
  {
    "userId": 1,
    "id": 1,
    "title": "example title",
    "completed": false
  },
  {
    "userId": 1,
    "id": 2,
    "title": "another example title",
    "completed": true
  }
]
```

The C# `Todo` object is defined as follows:

```
public record class Todo(
    int? UserId = null,
    int? Id = null,
    string? Title = null,
    bool? Completed = null);
```

It's a `record class` type, with optional `Id`, `Title`, `Completed`, and `UserId` properties. For more information on the `record` type, see [Introduction to record types in C#](#). To automatically deserialize `GET` requests into strongly typed C# object, use the `GetFromJsonAsync` extension method that's part of the `System.Net.Http.Json` NuGet package.

```
static async Task GetFromJsonAsync(HttpClient client)
{
    var todos = await client.GetFromJsonAsync<List<Todo>>(
        "todos?userId=1&completed=false");

    WriteLine("GET https://jsonplaceholder.typicode.com/todos?userId=1&completed=false HTTP/1.1");
    todos.ForEach.WriteLine();
    WriteLine();

    // Expected output:
    //   GET https://jsonplaceholder.typicode.com/todos?userId=1&completed=false HTTP/1.1
    //   Todo { UserId = 1, Id = 1, Title = delectus aut autem, Completed = False }
    //   Todo { UserId = 1, Id = 2, Title = quis ut nam facilis et officia qui, Completed = False }
    //   Todo { UserId = 1, Id = 3, Title = fugiat veniam minus, Completed = False }
    //   Todo { UserId = 1, Id = 5, Title = laboriosam mollitia et enim quasi adipisci quia provident illum, Completed = False }
    //   Todo { UserId = 1, Id = 6, Title = qui ullam ratione quibusdam voluptatem quia omnis, Completed = False }
    //   Todo { UserId = 1, Id = 7, Title = illo expedita consequatur quia in, Completed = False }
    //   Todo { UserId = 1, Id = 9, Title = molestiae perspiciatis ipsa, Completed = False }
    //   Todo { UserId = 1, Id = 13, Title = et doloremque nulla, Completed = False }
    //   Todo { UserId = 1, Id = 18, Title = dolorum est consequatur ea mollitia in culpa, Completed = False
}
```

In the preceding code:

- A `GET` request is made to `"https://jsonplaceholder.typicode.com/todos?userId=1&completed=false"`.
 - The query string represents the filtering criteria for the request.
- The response is automatically deserialized into a `List<Todo>` when successful.
- The request details are written to the console, along with each `Todo` object.

HTTP Post

A `POST` request sends data to the server for processing. The `Content-Type` header of the request signifies what **MIME type** the body is sending. To make an HTTP `POST` request, given an `HttpClient` and a URI, use the `HttpClient.PostAsync` method:

```
static async Task PostAsync(HttpClient client)
{
    using StringContent jsonContent = new(
        JsonSerializer.Serialize(new
        {
            userId = 77,
            id = 1,
            title = "write code sample",
            completed = false
        })),
        Encoding.UTF8,
        "application/json");

    using HttpResponseMessage response = await client.PostAsync(
        "todos",
        jsonContent);

    response.EnsureSuccessStatusCode()
        .WriteRequestToConsole();

    var jsonResponse = await response.Content.ReadAsStringAsync();
    WriteLine($"{jsonResponse}\n");

    // Expected output:
    // POST https://jsonplaceholder.typicode.com/todos HTTP/1.1
    // {
    //     "userId": 77,
    //     "id": 201,
    //     "title": "write code sample",
    //     "completed": false
    // }
}
```

The preceding code:

- Prepares a `StringContent` instance with the JSON body of the request (MIME type of `"application/json"`).
- Makes a `POST` request to `"https://jsonplaceholder.typicode.com/todos"`.
- Ensures that the response is successful, and writes the request details to the console.
- Writes the response body as a string to the console.

HTTP Post as JSON

To automatically serialize `POST` request arguments and deserialize responses into strongly-typed C# objects, use the `PostAsJsonAsync` extension method that's part of the `System.Net.Http.Json` NuGet package.

```

static async Task PostAsJsonAsync(HttpClient client)
{
    using HttpResponseMessage response = await client.PostAsJsonAsync(
        "todos",
        new Todo(UserId: 9, Id: 99, Title: "Show extensions", Completed: false));

    response.EnsureSuccessStatusCode()
        .WriteRequestToConsole();

    var todo = await response.Content.ReadFromJsonAsync<Todo>();
    WriteLine($"{todo}\n");

    // Expected output:
    // POST https://jsonplaceholder.typicode.com/todos HTTP/1.1
    // Todo { UserId = 9, Id = 201, Title = Show extensions, Completed = False }
}

```

The preceding code:

- Serializes the `Todo` instance as JSON, and makes a `POST` request to `"https://jsonplaceholder.typicode.com/todos"`.
- Ensures that the response is successful, and writes the request details to the console.
- Deserializes the response body into a `Todo` instance, and writes the `Todo` to the console.

HTTP Put

The `PUT` request method either replaces an existing resource or creates a new one using request body payload.

To make an HTTP `PUT` request, given an `HttpClient` and a URI, use the `HttpClient.PutAsync` method:

```

static async Task PutAsync(HttpClient client)
{
    using StringContent jsonContent = new(
        JsonSerializer.Serialize(new
        {
            userId = 1,
            id = 1,
            title = "foo bar",
            completed = false
        }),
        Encoding.UTF8,
        "application/json");

    using HttpResponseMessage response = await client.PutAsync(
        "todos/1",
        jsonContent);

    response.EnsureSuccessStatusCode()
        .WriteRequestToConsole();

    var jsonResponse = await response.Content.ReadAsStringAsync();
    WriteLine($"{jsonResponse}\n");

    // Expected output:
    // PUT https://jsonplaceholder.typicode.com/todos/1 HTTP/1.1
    // {
    //     "userId": 1,
    //     "id": 1,
    //     "title": "foo bar",
    //     "completed": false
    // }
}

```

The preceding code:

- Prepares a `StringContent` instance with the JSON body of the request (MIME type of "application/json").
- Makes a `PUT` request to "https://jsonplaceholder.typicode.com/todos/1".
- Ensures that the response is successful, and writes the request details and JSON response body to the console.

HTTP Put as JSON

To automatically serialize `PUT` request arguments and deserialize responses into strongly typed C# objects, use the `PutAsJsonAsync` extension method that's part of the `System.Net.Http.Json` NuGet package.

```
static async Task PutAsJsonAsync(HttpClient client)
{
    using HttpResponseMessage response = await client.PutAsJsonAsync(
        "todos/5",
        new Todo{Title: "partially update todo", Completed: true});

    response.EnsureSuccessStatusCode()
        .WriteRequestToConsole();

    var todo = await response.Content.ReadFromJsonAsync<Todo>();
    WriteLine($"{todo}\n");

    // Expected output:
    //   PUT https://jsonplaceholder.typicode.com/todos/5 HTTP/1.1
    //   Todo { UserId = , Id = 5, Title = partially update todo, Completed = True }
}
```

The preceding code:

- Serializes the `Todo` instance as JSON, and makes a `PUT` request to "https://jsonplaceholder.typicode.com/todos/5".
- Ensures that the response is successful, and writes the request details to the console.
- Deserializes the response body into a `Todo` instance, and writes the `Todo` to the console.

HTTP Patch

The `PATCH` request is a partial update to an existing resource. It won't create a new resource, and it's not intended to replace an existing resource. Instead, it updates a resource only partially. To make an HTTP `PATCH` request, given an `HttpClient` and a URI, use the `HttpClient.PatchAsync` method:

```

static async Task PatchAsync(HttpClient client)
{
    using StringContent jsonContent = new(
        JsonSerializer.Serialize(new
        {
            completed = true
        }),
        Encoding.UTF8,
        "application/json");

    using HttpResponseMessage response = await client.PatchAsync(
        "todos/1",
        jsonContent);

    response.EnsureSuccessStatusCode()
        .WriteRequestToConsole();

    var jsonResponse = await response.Content.ReadAsStringAsync();
    WriteLine($"{jsonResponse}\n");

    // Expected output
    // PATCH https://jsonplaceholder.typicode.com/todos/1 HTTP/1.1
    // {
    //     "userId": 1,
    //     "id": 1,
    //     "title": "delectus aut autem",
    //     "completed": true
    // }
}

```

The preceding code:

- Prepares a `StringContent` instance with the JSON body of the request (MIME type of `"application/json"`).
- Makes a `PATCH` request to `"https://jsonplaceholder.typicode.com/todos/1"`.
- Ensures that the response is successful, and writes the request details and JSON response body to the console.

No extension methods exist for `PATCH` requests in the `System.Net.Http.Json` NuGet package.

HTTP Delete

A `DELETE` request deletes an existing resource. A `DELETE` request is *idempotent* but not *safe*, meaning multiple `DELETE` requests to the same resources yield the same result, but the request will affect the state of the resource. To make an HTTP `DELETE` request, given an `HttpClient` and a URI, use the `HttpClient.DeleteAsync` method:

```

static async Task DeleteAsync(HttpClient client)
{
    using HttpResponseMessage response = await client.DeleteAsync("todos/1");

    response.EnsureSuccessStatusCode()
        .WriteRequestToConsole();

    var jsonResponse = await response.Content.ReadAsStringAsync();
    WriteLine($"{jsonResponse}\n");

    // Expected output
    // DELETE https://jsonplaceholder.typicode.com/todos/1 HTTP/1.1
    // {}
}

```

The preceding code:

- Makes a `DELETE` request to `"https://jsonplaceholder.typicode.com/todos/1"`.
- Ensures that the response is successful, and writes the request details to the console.

TIP

The response to a `DELETE` request (just like a `PUT` request) may or may not include a body.

HTTP Head

The `HEAD` request is similar to a `GET` request. Instead of returning the resource, it only returns the headers associated with the resource. A response to the `HEAD` request doesn't return a body. To make an HTTP `HEAD` request, given an `HttpClient` and a URI, use the `HttpClient.SendAsync` method with the `HttpMethod` set to `HttpMethod.Head`:

```
static async Task HeadAsync(HttpClient client)
{
    using HttpRequestMessage request = new(
        HttpMethod.Head,
        "https://www.example.com");

    using HttpResponseMessage response = await client.SendAsync(request);

    response.EnsureSuccessStatusCode()
        .WriteRequestToConsole();

    foreach (var header in response.Headers)
    {
        WriteLine($"{header.Key}: {string.Join(", ", header.Value)}");
    }
    WriteLine();

    // Expected output:
    // HEAD https://www.example.com/ HTTP/1.1
    // Accept-Ranges: bytes
    // Age: 550374
    // Cache-Control: max-age=604800
    // Date: Wed, 10 Aug 2022 17:24:55 GMT
    // ETag: "3147526947"
    // Server: ECS, (cha / 80E2)
    // X-Cache: HIT
}
```

The preceding code:

- Makes a `HEAD` request to `"https://www.example.com/"`.
- Ensures that the response is successful, and writes the request details to the console.
- Iterates over all of the response headers, writing each one to the console.

HTTP Options

The `OPTIONS` request is used to identify which HTTP methods a server or endpoint supports. To make an HTTP `OPTIONS` request, given an `HttpClient` and a URI, use the `HttpClient.SendAsync` method with the `HttpMethod` set to `HttpMethod.Options`:

```

static async Task OptionsAsync(HttpClient client)
{
    using HttpRequestMessage request = new(
        HttpMethod.Options,
        "https://www.example.com");

    using HttpResponseMessage response = await client.SendAsync(request);

    response.EnsureSuccessStatusCode()
        .WriteRequestToConsole();

    foreach (var header in response.Content.Headers)
    {
        WriteLine($"{header.Key}: {string.Join(", ", header.Value)}");
    }
    WriteLine();

    // Expected output
    // OPTIONS https://www.example.com/ HTTP/1.1
    // Allow: OPTIONS, GET, HEAD, POST
    // Content-Type: text/html; charset=utf-8
    // Expires: Wed, 17 Aug 2022 17:28:42 GMT
    // Content-Length: 0
}

```

The preceding code:

- Sends an `OPTIONS` HTTP request to `"https://www.example.com/"`.
- Ensures that the response is successful, and writes the request details to the console.
- Iterates over all of the response content headers, writing each one to the console.

HTTP Trace

The `TRACE` request can be useful for debugging as it provides application-level loop-back of the request message. To make an HTTP `TRACE` request, create an `HttpRequestMessage` using the `HttpMethod.Trace`:

```

using HttpRequestMessage request = new(
    HttpMethod.Trace,
    "{ValidRequestUri}");

```

Caution

The `TRACE` HTTP verb is not supported by all HTTP servers. It can expose a security vulnerability if used unwisely. For more information, see [Open Web Application Security Project \(OWASP\): Cross Site Tracing](#).

Handle an HTTP response

Whenever you're handling an HTTP response, you interact with the `HttpResponseMessage` type. Several members are used when evaluating the validity of a response. The HTTP status code is available via the `HttpResponseMessage.StatusCode` property. Imagine that you've sent a request given a client instance:

```

using HttpResponseMessage response = await client.SendAsync(request);

```

To ensure that the `response` is `OK` (HTTP status code 200), you can evaluate it as shown in the following example:

```
if (response is { StatusCode: HttpStatusCode.OK })
{
    // Omitted for brevity...
}
```

There are additional HTTP status codes that represent a successful response, such as `CREATED` (HTTP status code 201), `ACCEPTED` (HTTP status code 202), `NO CONTENT` (HTTP status code 204), and `RESET CONTENT` (HTTP status code 205). You can use the [HttpResponseMessage.IsSuccessStatusCode](#) property to evaluate these codes as well, which ensures that the response status code is within the range 200-299:

```
if (response.IsSuccessStatusCode)
{
    // Omitted for brevity...
}
```

If you need to have the framework throw the [HttpRequestException](#), you can call the [HttpResponseMessage.EnsureSuccessStatusCode\(\)](#) method:

```
response.EnsureSuccessStatusCode();
```

This code will throw an `HttpRequestException` if the response status code is not within the 200-299 range.

HTTP response errors

The HTTP response object ([HttpResponseMessage](#)), when not successful, contains information about the error. The [HttpWebResponse.StatusCode](#) property can be used to evaluate the error code.

For more information, see [Client error status codes](#) and [Server error status codes](#).

HTTP valid content responses

With a valid response, you can access the response body using the [Content](#) property. The body is available as an [HttpContent](#) instance, which you can use to access the body as a stream, byte array, or string:

```
await using Stream responseStream =
    await response.Content.ReadAsStreamAsync();
```

In the preceding code, the `responseStream` can be used to read the response body.

```
byte[] responseByteArray = await response.Content.ReadAsByteArrayAsync();
```

In the preceding code, the `responseByteArray` can be used to read the response body.

```
string responseString = await response.Content.ReadAsStringAsync();
```

In the preceding code, the `responseString` can be used to read the response body.

Finally, when you know an HTTP endpoint returns JSON, you can deserialize the response body into any valid C# object by using the [System.Net.Http.Json](#) NuGet package:

```
T? result = await response.Content.ReadFromJsonAsync<T>();
```

In the preceding code, `result` is the response body deserialized as the type `T`.

HTTP proxy

An HTTP proxy can be configured in one of two ways. A default is specified on the [HttpClient.DefaultProxy](#) property. Alternatively, you can specify a proxy on the [HttpClientHandler.Proxy](#) property.

Global default proxy

The `HttpClient.DefaultProxy` is a static property that determines the default proxy that all `HttpClient` instances use if no proxy is set explicitly in the [HttpClientHandler](#) passed through its constructor.

The default instance returned by this property will initialize following a different set of rules depending on your platform:

- **For Windows:** Reads proxy configuration from environment variables or, if those are not defined, from the user's proxy settings.
- **For macOS:** Reads proxy configuration from environment variables or, if those are not defined, from the system's proxy settings.
- **For Linux:** Reads proxy configuration from environment variables or, in case those are not defined, this property initializes a non-configured instance that bypasses all addresses.

The environment variables used for `DefaultProxy` initialization on Windows and Unix-based platforms are:

- `HTTP_PROXY` : the proxy server used on HTTP requests.
- `HTTPS_PROXY` : the proxy server used on HTTPS requests.
- `ALL_PROXY` : the proxy server used on HTTP and/or HTTPS requests in case `HTTP_PROXY` and/or `HTTPS_PROXY` are not defined.
- `NO_PROXY` : a comma-separated list of hostnames that should be excluded from proxying. Asterisks are not supported for wildcards; use a leading dot in case you want to match a subdomain. Examples:
`NO_PROXY=.example.com` (with leading dot) will match `www.example.com`, but will not match `example.com`.
`NO_PROXY=example.com` (without leading dot) will not match `www.example.com`. This behavior might be revisited in the future to match other ecosystems better.

On systems where environment variables are case-sensitive, the variable names may be all lowercase or all uppercase. The lowercase names are checked first.

The proxy server may be a hostname or IP address, optionally followed by a colon and port number, or it may be an `http` URL, optionally including a username and password for proxy authentication. The URL must be start with `http`, not `https`, and cannot include any text after the hostname, IP, or port.

Proxy per client

The [HttpClientHandler.Proxy](#) property identifies the [WebProxy](#) object to use to process requests to Internet resources. To specify that no proxy should be used, set the `Proxy` property to the proxy instance returned by the [GlobalProxySelection.GetEmptyWebProxy\(\)](#) method.

The local computer or application config file may specify that a default proxy be used. If the `Proxy` property is specified, then the proxy settings from the `Proxy` property override the local computer or application config file and the handler will use the proxy settings specified. If no proxy is specified in a config file and the `Proxy` property is unspecified, the handler uses the proxy settings inherited from the local computer. If there are no proxy settings, the request is sent directly to the server.

The [HttpClientHandler](#) class parses a proxy bypass list with wildcard characters inherited from local computer settings. For example, the `HttpClientHandler` class will parse a bypass list of `"nt*"` from browsers as a regular expression of `"nt.*"`. So a URL of `http://nt.com` would bypass the proxy using the `HttpClientHandler` class.

The `HttpClientHandler` class supports local proxy bypass. The class considers a destination to be local if any of the following conditions are met:

1. The destination contains a flat name (no dots in the URL).
2. The destination contains a loopback address ([Loopback](#) or [IPv6Loopback](#)) or the destination contains an [IPAddress](#) assigned to the local computer.
3. The domain suffix of the destination matches the local computer's domain suffix ([DomainName](#)).

For more information about configuring a proxy, see:

- [WebProxy.Address](#)
- [WebProxy.BypassProxyOnLocal](#)
- [WebProxy.BypassArrayList](#)

See also

- [HTTP support in .NET](#)
- [Guidelines for using HttpClient](#)
- [IHttpClientFactory with .NET](#)
- [Use HTTP/3 with HttpClient](#)
- [Test web APIs with the HttpRepl](#)

IHttpClientFactory with .NET

9/20/2022 • 10 minutes to read • [Edit Online](#)

In this article, you'll learn how to use the `IHttpClientFactory` to create `HttpClient` types with various .NET fundamentals, such as dependency injection (DI), logging, and configuration. The `HttpClient` type was introduced in .NET Framework 4.5, which was released in 2012. In other words, it's been around for a while. `HttpClient` is used for making HTTP requests and handling HTTP responses from web resources identified by a `Uri`. The HTTP protocol makes up the vast majority of all internet traffic.

With modern application development principles driving best practices, the `IHttpClientFactory` serves as a factory abstraction that can create `HttpClient` instances with custom configurations. `IHttpClientFactory` was introduced in .NET Core 2.1. Common HTTP-based .NET workloads can take advantage of resilient and transient-fault-handling third-party middleware with ease.

NOTE

If your app requires cookies, it might be better not to use `IHttpClientFactory` in your app. For alternative ways of managing clients, see [Guidelines for using HTTP clients](#).

The `IHttpClientFactory` type

All of the sample source code in this article relies on the `Microsoft.Extensions.Http` NuGet package.

Additionally, [The Internet Chuck Norris Database](#) free API is used to make HTTP `GET` requests for "nerdy" jokes.

When you call any of the `AddHttpClient` extension methods, you're adding the `IHttpClientFactory` and related services to the `IServiceCollection`. The `IHttpClientFactory` type offers the following benefits:

- Exposes the `HttpClient` class as a DI-ready type.
- Provides a central location for naming and configuring logical `HttpClient` instances.
- Codifies the concept of outgoing middleware via delegating handlers in `HttpClient`.
- Provides extension methods for Polly-based middleware to take advantage of delegating handlers in `HttpClient`.
- Manages the pooling and lifetime of underlying `HttpClientHandler` instances. Automatic management avoids common Domain Name System (DNS) problems that occur when manually managing `HttpClient` lifetimes.
- Adds a configurable logging experience (via `ILogger`) for all requests sent through clients created by the factory.

Consumption patterns

There are several ways `IHttpClientFactory` can be used in an app:

- [Basic usage](#)
- [Named clients](#)
- [Typed clients](#)
- [Generated clients](#)

The best approach depends upon the app's requirements.

Basic usage

To register the `IHttpClientFactory`, call `AddHttpClient`:

```
using BasicHttp.Example;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;

using IHost host = Host.CreateDefaultBuilder(args)
    .ConfigureServices(services =>
{
    services.AddHttpClient();
    services.AddTransient<JokeService>();
})
.Build();
```

Consuming services can require the `IHttpClientFactory` as a constructor parameter with [DI](#). The following code uses `IHttpClientFactory` to create an `HttpClient` instance:

```
using System.Net.Http.Json;
using Microsoft.Extensions.Logging;
using Shared;

namespace BasicHttp.Example;

public class JokeService
{
    private readonly IHttpClientFactory _httpClientFactory = null!;
    private readonly ILogger<JokeService> _logger = null!;

    public JokeService(
        IHttpClientFactory httpClientFactory,
        ILogger<JokeService> logger) =>
        (_httpClientFactory, _logger) = (httpClientFactory, logger);

    public async Task<string> GetRandomJokeAsync()
    {
        // Create the client
        HttpClient client = _httpClientFactory.CreateClient();

        try
        {
            // Make HTTP GET request
            // Parse JSON response deserialize into ChuckNorrisJoke type
            ChuckNorrisJoke? result = await client.GetFromJsonAsync<ChuckNorrisJoke>(
                "https://api.icndb.com/jokes/random?limitTo=[nerdy]",
                DefaultJsonSerialization.Options);

            if (result?.Value?.Joke is not null)
            {
                return result.Value.Joke;
            }
        }
        catch (Exception ex)
        {
            _logger.LogError("Error getting something fun to say: {Error}", ex);
        }

        return "Oops, something has gone wrong - that's not funny at all!";
    }
}
```

Using `IHttpClientFactory` like in the preceding example is a good way to refactor an existing app. It has no impact on how `HttpClient` is used. In places where `HttpClient` instances are created in an existing app, replace

those occurrences with calls to [CreateClient](#).

Named clients

Named clients are a good choice when:

- The app requires many distinct uses of `HttpClient`.
- Many `HttpClient` instances have different configuration.

Configuration for a named `HttpClient` can be specified during registration in `ConfigureServices`:

```
using NamedHttp.Example;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;

using IHost host = Host.CreateDefaultBuilder(args)
    .ConfigureServices((context, services) =>
{
    string httpClientName = context.Configuration["JokeHttpClientName"];
    services.AddHttpClient(
        httpClientName,
        client =>
    {
        // Set the base address of the named client.
        client.BaseAddress = new Uri("https://api.icndb.com/");

        // Add a user-agent default request header.
        client.DefaultRequestHeaders.UserAgent.ParseAdd("dotnet-docs");
    });
    services.AddTransient<JokeService>();
})
.Build();
```

In the preceding code, the client is configured with:

- A name that's pulled from the configuration under the `"JokeHttpClientName"`.
- The base address `https://api.icndb.com/`.
- A `"User-Agent"` header.

You can use configuration to specify HTTP client names, which is helpful to avoid misnaming clients when adding and creating. In this example, the `appsettings.json` file is used to configure the HTTP client name:

```
{
    "JokeHttpClientName": "ChuckNorrisJokeApi"
}
```

It's easy to extend this configuration and store more details about how you'd like your HTTP client to function.

For more information, see [Configuration in .NET](#).

Create client

Each time `CreateClient` is called:

- A new instance of `HttpClient` is created.
- The configuration action is called.

To create a named client, pass its name into `CreateClient`:

```

using System.Net.Http.Json;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.Logging;
using Shared;

namespace NamedHttp.Example;

public class JokeService
{
    private readonly IHttpClientFactory _httpClientFactory = null!;
    private readonly IConfiguration _configuration = null!;
    private readonly ILogger<JokeService> _logger = null!;

    public JokeService(
        IHttpClientFactory httpClientFactory,
        IConfiguration configuration,
        ILogger<JokeService> logger) =>
        (_httpClientFactory, _configuration, _logger) =
            (httpClientFactory, configuration, logger);

    public async Task<string> GetRandomJokeAsync()
    {
        // Create the client
        string httpClientName = _configuration["JokeHttpClientName"];
        HttpClient client = _httpClientFactory.CreateClient(httpClientName);

        try
        {
            // Make HTTP GET request
            // Parse JSON response deserialize into ChuckNorrisJoke type
            ChuckNorrisJoke? result = await client.GetFromJsonAsync<ChuckNorrisJoke>(
                "jokes/random?limitTo=[nerdy]",
                DefaultJsonSerialization.Options);

            if (result?.Value?.Joke is not null)
            {
                return result.Value.Joke;
            }
        }
        catch (Exception ex)
        {
            _logger.LogError("Error getting something fun to say: {Error}", ex);
        }

        return "Oops, something has gone wrong - that's not funny at all!";
    }
}

```

In the preceding code, the HTTP request doesn't need to specify a hostname. The code can pass just the path, since the base address configured for the client is used.

Typed clients

Typed clients:

- Provide the same capabilities as named clients without the need to use strings as keys.
- Provide [IntelliSense](#) and compiler help when consuming clients.
- Provide a single location to configure and interact with a particular `HttpClient`. For example, a single typed client might be used:
 - For a single backend endpoint.
 - To encapsulate all logic dealing with the endpoint.
- Work with DI and can be injected where required in the app.

A typed client accepts an `HttpClient` parameter in its constructor:

```
using System.Net.Http.Json;
using Microsoft.Extensions.Logging;
using Shared;

namespace TypedHttp.Example;

public sealed class JokeService
{
    private readonly HttpClient _httpClient = null!;
    private readonly ILogger<JokeService> _logger = null!;

    public JokeService(
        HttpClient httpClient,
        ILogger<JokeService> logger) =>
        (_httpClient, _logger) = (httpClient, logger);

    public async Task<string> GetRandomJokeAsync()
    {
        try
        {
            // Make HTTP GET request
            // Parse JSON response deserialize into ChuckNorrisJoke type
            ChuckNorrisJoke? result = await _httpClient.GetFromJsonAsync<ChuckNorrisJoke>(
                "https://api.icndb.com/jokes/random?limitTo=[nerdy]",
                DefaultJsonSerialization.Options);

            if (result?.Value?.Joke is not null)
            {
                return result.Value.Joke;
            }
        }
        catch (Exception ex)
        {
            _logger.LogError("Error getting something fun to say: {Error}", ex);
        }

        return "Oops, something has gone wrong - that's not funny at all!";
    }
}
```

In the preceding code:

- The configuration is set when the typed client is added to the service collection.
- The `HttpClient` is assigned as a class-scoped variable (field), and used with exposed APIs.

API-specific methods can be created that expose `HttpClient` functionality. For example, the `GetRandomJokeAsync` method encapsulates code to retrieve a random joke.

The following code calls `AddHttpClient` in `ConfigureServices` to register a typed client class:

```

using TypedHttp.Example;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;

using IHost host = Host.CreateDefaultBuilder(args)
    .ConfigureServices(services =>
{
    services.AddHttpClient<JokeService>(
        client =>
    {
        // Set the base address of the named client.
        client.BaseAddress = new Uri("https://api.icndb.com/");

        // Add a user-agent default request header.
        client.DefaultRequestHeaders.UserAgent.ParseAdd("dotnet-docs");
    });
})
.Build();

JokeService jokeService =

```

The typed client is registered as transient with DI. In the preceding code, `AddHttpClient` registers `JokeService` as a transient service. This registration uses a factory method to:

1. Create an instance of `HttpClient`.
2. Create an instance of `JokeService`, passing in the instance of `HttpClient` to its constructor.

Generated clients

`IHttpClientFactory` can be used in combination with third-party libraries such as [Refit](#). Refit is a REST library for .NET. It allows for declarative REST API definitions, mapping interface methods to endpoints. An implementation of the interface is generated dynamically by the `RestService`, using `HttpClient` to make the external HTTP calls.

Consider the following `record` types:

```

namespace Shared;

public record IdentifiableJokeValue(
    int Id, string Joke);

```

```

namespace Shared;

public record ChuckNorrisJoke(
    string Type,
    IdentifiableJokeValue Value);

```

The following example relies on the [Refit.HttpClientFactory](#) NuGet package, and is a simple interface:

```

using Refit;
using Shared;

namespace GeneratedHttp.Example;

public interface IJokeService
{
    [Get("/jokes/random?limitTo=[nerdy]")]
    Task<ChuckNorrisJoke> GetRandomJokeAsync();
}

```

The preceding C# interface:

- Defines a method named `GetRandomJokeAsync` that returns a `Task<ChuckNorrisJoke>` instance.
- Declares a `Refit.GetAttribute` attribute with the path and query string to the external API.

A typed client can be added, using Refit to generate the implementation:

```
using GeneratedHttp.Example;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using Refit;
using Shared;

using IHost host = Host.CreateDefaultBuilder(args)
    .ConfigureServices(services =>
{
    services.AddRefitClient<IJokeService>()
        .ConfigureHttpClient(client =>
    {
        // Set the base address of the named client.
        client.BaseAddress = new Uri("https://api.icndb.com/");

        // Add a user-agent default request header.
        client.DefaultRequestHeaders.UserAgent.ParseAdd("dotnet-docs");
    });
})
.Build();
```

The defined interface can be consumed where necessary, with the implementation provided by DI and Refit.

Make POST, PUT, and DELETE requests

In the preceding examples, all HTTP requests use the `GET` HTTP verb. `HttpClient` also supports other HTTP verbs, including:

- `POST`
- `PUT`
- `DELETE`
- `PATCH`

For a complete list of supported HTTP verbs, see [HttpMethod](#).

The following example shows how to make an HTTP `POST` request:

```
public async Task CreateItemAsync(Item item)
{
    using StringContent json = new(
        JsonSerializer.Serialize(item, DefaultJsonSerialization.Options),
        Encoding.UTF8,
        MediaTypeNames.Application.Json);

    using HttpResponseMessage httpResponse =
        await _httpClient.PostAsync("/api/items", json);

    httpResponse.EnsureSuccessStatusCode();
}
```

In the preceding code, the `CreateItemAsync` method:

- Serializes the `Item` parameter to JSON using `System.Text.Json`. This uses an instance of `JsonSerializerOptions` to configure the serialization process.
- Creates an instance of `StringContent` to package the serialized JSON for sending in the HTTP request's body.
- Calls `PostAsync` to send the JSON content to the specified URL. This is a relative URL that gets added to the `HttpClient.BaseAddress`.
- Calls `EnsureSuccessStatusCode` to throw an exception if the response status code does not indicate success.

`HttpClient` also supports other types of content. For example, `MultipartContent` and `StreamContent`. For a complete list of supported content, see [HttpContent](#).

The following example shows an HTTP `PUT` request:

```
public async Task UpdateItemAsync(Item item)
{
    using StringContent json = new(
        JsonSerializer.Serialize(item, DefaultJsonSerialization.Options),
        Encoding.UTF8,
        MediaTypeNames.Application.Json);

    using HttpResponseMessage httpResponse =
        await _httpClient.PutAsync($""/api/items/{item.Id}", json);

    httpResponse.EnsureSuccessStatusCode();
}
```

The preceding code is very similar to the `POST` example. The `UpdateItemAsync` method calls `PutAsync` instead of `PostAsync`.

The following example shows an HTTP `DELETE` request:

```
public async Task DeleteItemAsync(Guid id)
{
    using HttpResponseMessage httpResponse =
        await _httpClient.DeleteAsync($""/api/items/{id}");

    httpResponse.EnsureSuccessStatusCode();
}
```

In the preceding code, the `DeleteItemAsync` method calls `DeleteAsync`. Because HTTP DELETE requests typically contain no body, the `DeleteAsync` method doesn't provide an overload that accepts an instance of `HttpContent`.

To learn more about using different HTTP verbs with `HttpClient`, see [HttpClient](#).

HttpClient lifetime management

A new `HttpClient` instance is returned each time `CreateClient` is called on the `IHttpClientFactory`. One `HttpClientHandler` instance is created per client. The factory manages the lifetimes of the `HttpClientHandler` instances.

`IHttpClientFactory` pools the `HttpClientHandler` instances created by the factory to reduce resource consumption. An `HttpClientHandler` instance may be reused from the pool when creating a new `HttpClient` instance if its lifetime hasn't expired.

Pooling of handlers is desirable as each handler typically manages its own underlying HTTP connection. Creating more handlers than necessary can result in connection delays. Some handlers also keep connections open indefinitely, which can prevent the handler from reacting to DNS changes.

The default handler lifetime is two minutes. To override the default value, call `SetHandlerLifetime` for each client, on the `IServiceCollection`:

```
services.AddHttpClient("Named.Client")
    .SetHandlerLifetime(TimeSpan.FromMinutes(5));
```

IMPORTANT

You can generally treat `HttpClient` instances as objects that **do not** require disposal. Disposal cancels outgoing requests and guarantees that the `HttpClient` instance can't be used after calling `Dispose`. `IHttpClientFactory` tracks and disposes resources used to create `HttpClient` instances, specifically the `HttpMessageHandler`.

Keeping a single `HttpClient` instance alive for a long duration is a common pattern used before the inception of `IHttpClientFactory`. For information about which strategy to use in your app, see [Guidelines for using HTTP clients](#).

Configure the `HttpMessageHandler`

It may be necessary to control the configuration of the inner `HttpMessageHandler` used by a client.

An `IHttpClientBuilder` is returned when adding named or typed clients. The `ConfigurePrimaryHttpMessageHandler` extension method can be used to define a delegate on the `IServiceCollection`. The delegate is used to create and configure the primary `HttpMessageHandler` used by that client:

```
services.AddHttpClient("Named.Client")
    .ConfigurePrimaryHttpMessageHandler(() =>
{
    return new HttpClientHandler
    {
        AllowAutoRedirect = false,
        UseDefaultCredentials = true
    };
});
```

Configuring the `HttpClientHandler` lets you specify a proxy for the `HttpClient` instance. For more information, see [Proxy per client](#).

Additional configuration

There are several additional configuration options for controlling the `IHttpClientHandler`:

METHOD	DESCRIPTION
AddHttpMessageHandler	Adds an additional message handler for a named <code>HttpClient</code> .
AddTypedClient	Configures the binding between the <code>TClient</code> and the named <code>HttpClient</code> associated with the <code>IHttpClientBuilder</code> .
ConfigureHttpClient	Adds a delegate that will be used to configure a named <code>HttpClient</code> .

METHOD	DESCRIPTION
ConfigureHttpMessageHandlerBuilder	Adds a delegate that will be used to configure message handlers using HttpMessageHandlerBuilder for a named HttpClient .
ConfigurePrimaryHttpMessageHandler	Configures the primary HttpMessageHandler from the dependency injection container for a named HttpClient .
RedactLoggedHeaders	Sets the collection of HTTP header names for which values should be redacted before logging.
SetHandlerLifetime	Sets the length of time that a HttpMessageHandler instance can be reused. Each named client can have its own configured handler lifetime value.

See also

- [Dependency injection in .NET](#)
- [Logging in .NET](#)
- [Configuration in .NET](#)
- [IHttpClientFactory](#)
- [HttpClient](#)
- [Make HTTP requests with the HttpClient](#)
- [Implement HTTP retry with exponential backoff](#)

Use HTTP/3 with HttpClient

9/20/2022 • 3 minutes to read • [Edit Online](#)

[HTTP/3](#) is the third and upcoming major version of HTTP. HTTP/3 uses the same semantics as HTTP/1.1 and HTTP/2: the same request methods, status codes, and message fields apply to all versions. The differences are in the underlying transport. Both HTTP/1.1 and HTTP/2 use TCP as their transport. HTTP/3 uses a new transport technology developed alongside HTTP/3 called [QUIC](#).

HTTP/3 and QUIC have a number of benefits compared to HTTP/1.1 and HTTP/2:

- Faster response time of the first request. QUIC and HTTP/3 negotiate the connection in fewer round-trips between the client and the server. The first request reaches the server faster.
- Improved experience when there is connection packet loss. HTTP/2 multiplexes multiple requests via one TCP connection. Packet loss on the connection affects all requests. This problem is called "head-of-line blocking". Because QUIC provides native multiplexing, lost packets only impact the requests where data has been lost.
- Supports transitioning between networks. This feature is useful for mobile devices where it is common to switch between WIFI and cellular networks as a mobile device changes location. Currently, HTTP/1.1 and HTTP/2 connections fail with an error when switching networks. An app or web browser must retry any failed HTTP requests. HTTP/3 allows the app or web browser to seamlessly continue when a network changes. `HttpClient` and Kestrel do not support network transitions in .NET 6. It may be available in a future release.

IMPORTANT

HTTP/3 is available in .NET 6 as a *preview feature* because the HTTP/3 specification is not finalized and behavioral or performance issues may exist in HTTP/3 with .NET 6.

For more information on preview features, see [the preview features specification](#).

Apps configured to take advantage of HTTP/3 should be designed to also support HTTP/1.1 and HTTP/2. If issues are identified in HTTP/3, we recommend disabling HTTP/3 until the issues are resolved in a future release of .NET.

HttpClient settings

HTTP/3 support is in preview, and needs to be enabled via a configuration flag which can be set in the project with:

```
<ItemGroup>
    <RuntimeHostConfigurationOption Include="System.Net.SocketsHttpHandler.Http3Support" Value="true" />
</ItemGroup>
```

Or using `ApplicationContext.SetSwitch`.

The HTTP version can be configured by setting `HttpRequestMessage.Version` to 3.0. However, because not all routers, firewalls, and proxies properly support HTTP/3, we recommend configuring HTTP/3 together with HTTP/1.1 and HTTP/2. In `HttpClient`, this can be done by specifying:

- `HttpRequestMessage.Version` to 1.1.
- `HttpRequestMessage.VersionPolicy` to `HttpVersionPolicy.RequestVersionOrHigher`.

The reason for requiring a configuration flag for HTTP/3 is to protect apps from future breakage when using

version policy `RequestVersionOrHigher`. When calling a server that currently uses HTTP/1.1 and HTTP/2, if the server later upgrades to HTTP/3, the client would try to use HTTP/3 and potentially be incompatible as the standard is not final and therefore may change after .NET 6 is released.

Platform dependencies

HTTP/3 uses QUIC as its transport protocol. The .NET implementation of HTTP/3 uses [MsQuic](#) to provide QUIC functionality. MsQuic is included in specific builds of windows and as a library for Linux. If the platform that `HttpClient` is running on doesn't have all the requirements for HTTP/3 then it's disabled.

Windows

- Windows 11 Build 22000 (version 21H2) or later.
- TLS 1.3 or later connection.

Linux

On Linux, `libmsquic` is published via Microsoft's official Linux package repository packages.microsoft.com. To consume it, it must be added manually. See [Linux Software Repository for Microsoft Products](#). After configuring the package feed, it can be installed via the package manager of your distro, for example, for Ubuntu:

```
sudo apt install libmsquic=1.9*
```

NOTE

.NET 6 is only compatible with the 1.9.x versions of `libmsquic`. `Libmsquic` 2.x is not compatible due to breaking changes. `Libmsquic` will receive updates to 1.9.x when needed to incorporate security fixes.

macOS

HTTP/3 is not currently supported on macOS but may be available in a future release.

Using `HttpClient`

Include the following in the project file to enable HTTP/3 with `HttpClient`:

```
<ItemGroup>
    <RuntimeHostConfigurationOption Include="System.Net.SocketsHttpHandler.Http3Support" Value="true" />
</ItemGroup>
```

The following code example uses [top-level statements](#) and demonstrates how to specify HTTP3 in the request:

```
// See https://aka.ms/new-console-template for more information
using System.Net;

var client = new HttpClient()
{
    DefaultRequestVersion = HttpVersion.Version30,
    DefaultVersionPolicy = HttpVersionPolicy.RequestVersionExact
};

Console.WriteLine("--- Localhost:5001 ---");
HttpResponseMessage resp = await client.GetAsync("https://localhost:5001/");
string body = await resp.Content.ReadAsStringAsync();
Console.WriteLine(
    $"status: {resp.StatusCode}, version: {resp.Version}, " +
    $"body: {body.Substring(0, Math.Min(100, body.Length))}");
```

HTTP/3 Server

HTTP/3 is supported by ASP.NET with the Kestrel server in .NET 6. For more information, see [use HTTP/3 with the ASP.NET Core Kestrel web server](#).

Public test servers

Cloudflare hosts a site for HTTP/3 which can be used to test the client against at <https://cloudflare-quic.com/>

See also

- [HttpClient](#)
- [HTTP/3 support in Kestrel](#)

Rate limit an HTTP handler in .NET

9/20/2022 • 9 minutes to read • [Edit Online](#)

IMPORTANT

This content relies on NuGet packages that are currently in PREVIEW. Some information relates to prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, express or implied, with respect to the information provided here.

In this article, you'll learn how to create a client-side HTTP handler that rate limits the number of requests it sends. You'll see an `HttpClient` that accesses the `"www.example.com"` resource. Resources are consumed by apps that rely on them, and when an app makes too many requests to a single resource, it can lead to *resource contention*. Resource contention occurs when a resource is consumed by too many apps, and the resource is unable to serve all of the apps that are requesting it. This can result in a poor user experience, and in some cases, it can even lead to a denial of service (DoS) attack. For more information on DoS, see [OWASP: Denial of Service](#).

What is rate limiting?

Rate limiting is the concept of limiting how much a resource can be accessed. For example, you may know that a database your app accesses can safely handle 1,000 requests per minute, but it may not handle much more than that. You can put a rate limiter in your app that only allows 1,000 requests every minute and rejects any more requests before they can access the database. Thus, rate limiting your database and allowing your app to handle a safe number of requests. This is a common pattern in distributed systems, where you may have multiple instances of an app running, and you want to ensure that they don't all try to access the database at the same time. There are multiple different rate-limiting algorithms to control the flow of requests.

To use rate limiting in .NET, you'll reference the `System.Threading.RateLimiting` NuGet package.

Implement a `DelegatingHandler` subclass

To control the flow of requests, you implement a custom `DelegatingHandler` subclass. This is a type of `HttpMessageHandler` that allows you to intercept and handle requests before they're sent to the server. You can also intercept and handle responses before they're returned to the caller. In this example, you'll implement a custom `DelegatingHandler` subclass that limits the number of requests that can be sent to a single resource. Consider the following custom `ClientSideRateLimitedHandler` class:

```

internal sealed class ClientSideRateLimitedHandler
    : DelegatingHandler, IAsyncDisposable
{
    private readonly RateLimiter _rateLimiter;

    public ClientSideRateLimitedHandler(RateLimiter limiter)
        : base(new HttpClientHandler()) => _rateLimiter = limiter;

    protected override async Task<HttpResponseMessage> SendAsync(
        HttpRequestMessage request, CancellationToken cancellationToken)
    {
        using RateLimitLease lease = await _rateLimiter.WaitAsync(
            permitCount: 1, cancellationToken);

        if (lease.IsAcquired)
        {
            return await base.SendAsync(request, cancellationToken);
        }

        var response = new HttpResponseMessage(HttpStatusCode.TooManyRequests);
        if (lease.TryGetMetadata(
            MetadataName.RetryAfter, out TimeSpan retryAfter))
        {
            response.Headers.Add(
                "Retry-After",
                ((int)retryAfter.TotalSeconds).ToString(
                    NumberFormatInfo.InvariantInfo));
        }

        return response;
    }

    async ValueTask IAsyncDisposable.DisposeAsync()
    {
        await _rateLimiter.DisposeAsync().ConfigureAwait(false);

        Dispose(disposing: false);
        GC.SuppressFinalize(this);
    }

    protected override void Dispose(bool disposing)
    {
        base.Dispose(disposing);

        if (disposing)
        {
            _rateLimiter.Dispose();
        }
    }
}

```

The preceding C# code:

- Inherits the `DelegatingHandler` type.
- Implements the `IAsyncDisposable` interface.
- Defines a `RateLimiter` field that is assigned from the constructor.
- The `SendAsync` method is overridden to intercept and handle requests before they're sent to the server.
- The `DisposeAsync()` method is overridden to dispose of the `RateLimiter` instance.

Looking a bit closer at the `SendAsync` method, you'll see that it:

- Relies on the `RateLimiter` instance to acquire a `RateLimitLease` from the `WaitAsync`.
- When the `lease.IsAcquired` property is `true`, the request is sent to the server.

- Otherwise, an `HttpResponseMessage` is returned with a `429` status code, and if the `lease` contains a `RetryAfter` value, the `Retry-After` header is set to that value.

Emulate many concurrent requests

To put this custom `DelegatingHandler` subclass to the test, you'll create a console app that emulates many concurrent requests. This `Program` class creates an `HttpClient` with the custom `ClientSideRateLimitedHandler`:

```

var options = new TokenBucketRateLimiterOptions(
    tokenLimit: 8,
    queueProcessingOrder: QueueProcessingOrder.OldestFirst,
    queueLimit: 3,
    replenishmentPeriod: TimeSpan.FromMilliseconds(1),
    tokensPerPeriod: 2,
    autoReplenishment: true);

// Create an HTTP client with the client-side rate limited handler.
using HttpClient client = new(
    handler: new ClientSideRateLimitedHandler(
        limiter: new TokenBucketRateLimiter(options)));

// Create 100 urls with a unique query string.
var oneHundredUrls = Enumerable.Range(0, 100).Select(
    i => $"https://example.com?iteration={i:0#}");

// Flood the HTTP client with requests.
var floodOneThroughFortyNineTask = Parallel.ForEachAsync(
    source: oneHundredUrls.Take(0..49),
    body: (url, cancellationToken) => GetAsync(client, url, cancellationToken));

var floodFiftyThroughOneHundredTask = Parallel.ForEachAsync(
    source: oneHundredUrls.Take(^50..),
    body: (url, cancellationToken) => GetAsync(client, url, cancellationToken));

await Task.WhenAll(
    floodOneThroughFortyNineTask,
    floodFiftyThroughOneHundredTask);

static async ValueTask GetAsync(
    HttpClient client, string url, CancellationToken cancellationToken)
{
    using var response =
        await client.GetAsync(url, cancellationToken);

    Console.WriteLine(
        $"URL: {url}, HTTP status code: {response.StatusCode} ({(int)response.StatusCode})");
}

```

In the preceding console app:

- The `TokenBucketRateLimiterOptions` are configured with a token limit of `8`, and queue processing order of `oldestFirst`, a queue limit of `3`, and replenishment period of `1` millisecond, a tokens per period value of `2`, and an auto-replenish value of `true`.
- An `HttpClient` is created with the `ClientSideRateLimitedHandler` that is configured with the `TokenBucketRateLimiter`.
- To emulate 100 requests, `Enumerable.Range` creates 100 URLs, each with a unique query string parameter.
- Two `Task` objects are assigned from the `Parallel.ForEachAsync` method, splitting the URLs into two groups.
- The `HttpClient` is used to send a `GET` request to each URL, and the response is written to the console.
- `Task.WhenAll` waits for both tasks to complete.

Since the `HttpClient` is configured with the `ClientSideRateLimitedHandler`, not all requests will make it to the server resource. You can test this assertion by running the console app. You'll see that only a fraction of the total number of requests are sent to the server, and the rest are rejected with an HTTP status code of `429`. Try altering the `options` object used to create the `TokenBucketRateLimiter` to see how the number of requests that are sent to the server changes.

Consider the following example output:

```
URL: https://example.com?iteration=06, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=60, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=55, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=59, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=57, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=11, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=63, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=13, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=62, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=65, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=64, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=67, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=14, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=68, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=16, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=69, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=70, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=71, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=17, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=18, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=72, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=73, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=74, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=19, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=75, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=76, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=79, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=77, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=21, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=78, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=81, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=22, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=80, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=20, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=82, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=83, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=23, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=84, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=24, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=85, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=86, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=25, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=87, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=26, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=88, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=89, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=27, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=90, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=28, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=91, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=94, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=29, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=93, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=96, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=92, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=95, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=31, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=30, HTTP status code: TooManyRequests (429)
```

```
URL: https://example.com?iteration=97, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=98, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=99, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=32, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=33, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=34, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=35, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=36, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=37, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=38, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=39, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=40, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=41, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=42, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=43, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=44, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=45, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=46, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=47, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=48, HTTP status code: TooManyRequests (429)
URL: https://example.com?iteration=15, HTTP status code: OK (200)
URL: https://example.com?iteration=04, HTTP status code: OK (200)
URL: https://example.com?iteration=54, HTTP status code: OK (200)
URL: https://example.com?iteration=08, HTTP status code: OK (200)
URL: https://example.com?iteration=00, HTTP status code: OK (200)
URL: https://example.com?iteration=51, HTTP status code: OK (200)
URL: https://example.com?iteration=10, HTTP status code: OK (200)
URL: https://example.com?iteration=66, HTTP status code: OK (200)
URL: https://example.com?iteration=56, HTTP status code: OK (200)
URL: https://example.com?iteration=52, HTTP status code: OK (200)
URL: https://example.com?iteration=12, HTTP status code: OK (200)
URL: https://example.com?iteration=53, HTTP status code: OK (200)
URL: https://example.com?iteration=07, HTTP status code: OK (200)
URL: https://example.com?iteration=02, HTTP status code: OK (200)
URL: https://example.com?iteration=01, HTTP status code: OK (200)
URL: https://example.com?iteration=61, HTTP status code: OK (200)
URL: https://example.com?iteration=05, HTTP status code: OK (200)
URL: https://example.com?iteration=09, HTTP status code: OK (200)
URL: https://example.com?iteration=03, HTTP status code: OK (200)
URL: https://example.com?iteration=58, HTTP status code: OK (200)
URL: https://example.com?iteration=50, HTTP status code: OK (200)
```

You'll notice that the first logged entries are always the immediately returned 429 responses, and the last entries are always the 200 responses. This is because the rate limit is encountered client-side and avoids making an HTTP call to a server. This is a good thing because it means that the server isn't flooded with requests. It also means that the rate limit is enforced consistently across all clients.

Note also that each URL's query string is unique: examine the `iteration` parameter to see that it's incremented by one for each request. This parameter helps to illustrate that the 429 responses aren't from the first requests, but rather from the requests that are made after the rate limit is reached. The 200 responses arrive later but these requests were made earlier—before the limit was reached.

To have a better understanding of the various rate-limiting algorithms, try rewriting this code to accept a different `RateLimiter` implementation. In addition to the `TokenBucketRateLimiter` you could try:

- `ConcurrencyLimiter`
- `FixedWindowRateLimiter`
- `PartitionedRateLimiter`
- `SlidingWindowRateLimiter`

Summary

In this article, you learned how to implement a custom `ClientSideRateLimitedHandler`. This pattern could be used to implement a rate-limited HTTP client for resources that you know have API limits. In this way, you're preventing your client app from making unnecessary requests to the server, and you're also preventing your app from being blocked by the server. Additionally, with the use of metadata to store retry timing values, you could also implement automatic retry logic.

See also

- [Announcing Rate Limiting for .NET](#)
- [Rate limiting middleware in ASP.NET Core](#)
- [Azure Architecture: Rate limiting pattern](#)
- [Automatic retry logic in .NET](#)

Sockets in .NET

9/20/2022 • 2 minutes to read • [Edit Online](#)

The [System.Net.Sockets](#) namespace contains a managed, cross-platform socket networking implementation. All other network-access classes in the [System.Net](#) namespace are built on top of this implementation of sockets.

The [Socket](#) class is a managed-code version of the socket services provided relying on native interoperability with Linux, macOS, or Windows. In most cases, the `Socket` class methods simply marshal data into their native counterparts and handle any necessary security checks.

The `Socket` class supports two basic modes, synchronous and asynchronous. In synchronous mode, calls to functions that perform network operations (such as [SendAsync](#) and [ReceiveAsync](#)) wait until the operation completes before returning control to the calling program. In asynchronous mode, these calls return immediately.

See also

- [Use Sockets to send and receive data](#)
- [Networking in .NET](#)
- [System.Net.Sockets](#)
- [Socket](#)

Use Sockets to send and receive data

9/20/2022 • 4 minutes to read • [Edit Online](#)

Before you can use a socket to communicate with remote devices, the socket must be initialized with protocol and network address information. The constructor for the [Socket](#) class has parameters that specify the address family, socket type, and protocol type that the socket uses to make connections. When connecting a client socket to a server socket, the client will use an [IPEndPoint](#) object to specify the network address of the server.

Create an IP endpoint

When working with [System.Net.Sockets](#), you represent a network endpoint as an [IPEndPoint](#) object. The [IPEndPoint](#) is constructed with an [IPAddress](#) and its corresponding port number. Before you can initiate a conversation through a [Socket](#), you create a data pipe between your app and the remote destination.

TCP/IP uses a network address and a service port number to uniquely identify a service. The network address identifies a specific network destination; the port number identifies the specific service on that device to connect to. The combination of network address and service port is called an endpoint, which is represented in the .NET by the [EndPoint](#) class. A descendant of [EndPoint](#) is defined for each supported address family; for the IP address family, the class is [IPEndPoint](#).

The [Dns](#) class provides domain-name services to apps that use TCP/IP internet services. The [GetHostEntryAsync](#) method queries a DNS server to map a user-friendly domain name (such as "host.contoso.com") to a numeric Internet address (such as `192.168.1.1`). [GetHostEntryAsync](#) returns a [Task<IPHostEntry>](#) that when awaited contains a list of addresses and aliases for the requested name. In most cases, you can use the first address returned in the [AddressList](#) array. The following code gets an [IPAddress](#) containing the IP address for the server `host.contoso.com`.

```
IPHostEntry ipHostInfo = await Dns.GetHostEntryAsync("host.contoso.com");
IPAddress ipAddress = ipHostInfo.AddressList[0];
```

TIP

For manual testing and debugging purposes, you can typically use the [GetHostEntryAsync](#) method to get given the [Dns.GetHostName\(\)](#) value to resolve the localhost name to an IP address.

The Internet Assigned Numbers Authority (IANA) defines port numbers for common services. For more information, see [IANA: Service Name and Transport Protocol Port Number Registry](#). Other services can have registered port numbers in the range 1,024 to 65,535. The following code combines the IP address for `host.contoso.com` with a port number to create a remote endpoint for a connection.

```
IPEndPoint ipEndPoint = new(ipAddress, 11_000);
```

After determining the address of the remote device and choosing a port to use for the connection, the app can establish a connection with the remote device.

Create a [Socket](#) client

With the [endPoint](#) object created, create a client socket to connect to the server. Once the socket is connected, it

can send and receive data from the server socket connection.

```
using Socket client = new(
    ipEndPoint.AddressFamily,
    SocketType.Stream,
    ProtocolType.Tcp);

await client.ConnectAsync(ipEndPoint);
while (true)
{
    // Send message.
    var message = "Hi friends !<|EOM|>";
    var messageBytes = Encoding.UTF8.GetBytes(message);
    _ = await client.SendAsync(messageBytes, SocketFlags.None);
    Console.WriteLine($"Socket client sent message: \"{message}\"");

    // Receive ack.
    var buffer = new byte[1_024];
    var received = await client.ReceiveAsync(buffer, SocketFlags.None);
    var response = Encoding.UTF8.GetString(buffer, 0, received);
    if (response == "<|ACK|>")
    {
        Console.WriteLine(
            $"Socket client received acknowledgment: \"{response}\"");
        break;
    }
    // Sample output:
    //     Socket client sent message: "Hi friends !<|EOM|>"
    //     Socket client received acknowledgment: "<|ACK|>"
}

client.Shutdown(SocketShutdown.Both);
```

The preceding C# code:

- Instantiates a new `Socket` object with a given `endPoint` instances address family, the `SocketType.Stream`, and `ProtocolType.Tcp`.
- Calls the `Socket.ConnectAsync` method with the `endPoint` instance as an argument.
- In a `while` loop:
 - Encodes and sends a message to the server using `Socket.SendAsync`.
 - Writes the sent message to the console.
 - Initializes a buffer to receive data from the server using `Socket.ReceiveAsync`.
 - When the `response` is an acknowledgment, it is written to the console and the loop is exited.
- Finally, the `client` socket calls `Socket.Shutdown` given `SocketShutdown.Both`, which shuts down both send and receive operations.

Create a `Socket` server

To create the server socket, the `endPoint` object can listen for incoming connections on any IP address but the port number must be specified. Once the socket is created, the server can accept incoming connections and communicate with clients.

```

using Socket listener = new(
    ipEndPoint.AddressFamily,
    SocketType.Stream,
    ProtocolType.Tcp);

listener.Bind(ipEndPoint);
listener.Listen(100);

var handler = await listener.AcceptAsync();
while (true)
{
    // Receive message.
    var buffer = new byte[1_024];
    var received = await handler.ReceiveAsync(buffer, SocketFlags.None);
    var response = Encoding.UTF8.GetString(buffer, 0, received);

    var eom = "<|EOM|>";
    if (response.IndexOf(eom) > -1 /* is end of message */)
    {
        Console.WriteLine(
            $"Socket server received message: \'{response.Replace(eom, "")}\'');

        var ackMessage = "<|ACK|>";
        var echoBytes = Encoding.UTF8.GetBytes(ackMessage);
        await handler.SendAsync(echoBytes, 0);
        Console.WriteLine(
            $"Socket server sent acknowledgment: \'{ackMessage}\'');

        break;
    }
    // Sample output:
    //     Socket server received message: "Hi friends !"
    //     Socket server sent acknowledgment: "<|ACK|>"
}

```

The preceding C# code:

- Instantiates a new `Socket` object with a given `endPoint` instances address family, the `SocketType.Stream`, and `ProtocolType.Tcp`.
- The `listener` calls the `Socket.Bind` method with the `endPoint` instance as an argument to associate the socket with the network address.
- The `Socket.Listen()` method is called to listen for incoming connections.
- The `listener` calls the `Socket.AcceptAsync` method to accept an incoming connection on the `handler` socket.
- In a `while` loop:
 - Calls `Socket.ReceiveAsync` to receive data from the client.
 - When the data is received, it's decoded and written to the console.
 - If the `response` message ends with `<|EOM|>`, an acknowledgment is sent to the client using the `Socket.SendAsync`.

See also

- [Sockets in .NET](#)
- [Networking in .NET](#)
- [System.Net.Sockets](#)
- [Socket](#)

TCP in .NET

9/20/2022 • 2 minutes to read • [Edit Online](#)

To use the Transmission Control Protocol (TCP) services in .NET, use the [TcpClient](#) and [TcpListener](#) classes. These protocol classes are built on top of the [System.Net.Sockets.Socket](#) class and take care of the details of transferring data.

The protocol classes use the underlying `Socket` class to provide simple access to network services without the overhead of maintaining state information or knowing the details of setting up protocol-specific sockets. To use asynchronous `Socket` methods, you can use the asynchronous methods supplied by the [NetworkStream](#) class. To access features of the `Socket` class not exposed by the protocol classes, you must use the `Socket` class.

`TcpClient` and `TcpListener` represent the network using the `NetworkStream` class. You use the [GetStream](#) method to return the network stream, and then call the stream's [NetworkStream.ReadAsync](#) and [NetworkStream.WriteAsync](#) methods. The `NetworkStream` does not own the protocol classes' underlying socket, so closing it does not affect the socket.

See also

- [Use TcpClient and TcpListener](#)
- [Networking in .NET](#)

Use TcpClient and TcpListener

9/20/2022 • 4 minutes to read • [Edit Online](#)

The [TcpClient](#) class requests data from an internet resource using TCP. The methods and properties of [TcpClient](#) abstract the details for creating a [Socket](#) for requesting and receiving data using TCP. Because the connection to the remote device is represented as a stream, data can be read and written with .NET Framework stream-handling techniques.

The TCP protocol establishes a connection with a remote endpoint and then uses that connection to send and receive data packets. TCP is responsible for ensuring that data packets are sent to the endpoint and assembled in the correct order when they arrive.

Create an IP endpoint

When working with [System.Net.Sockets](#), you represent a network endpoint as an [IPEndPoint](#) object. The [IPEndPoint](#) is constructed with an [IPAddress](#) and its corresponding port number. Before you can initiate a conversation through a [Socket](#), you create a data pipe between your app and the remote destination.

TCP/IP uses a network address and a service port number to uniquely identify a service. The network address identifies a specific network destination; the port number identifies the specific service on that device to connect to. The combination of network address and service port is called an endpoint, which is represented in the .NET by the [EndPoint](#) class. A descendant of [EndPoint](#) is defined for each supported address family; for the IP address family, the class is [IPEndPoint](#).

The [Dns](#) class provides domain-name services to apps that use TCP/IP internet services. The [GetHostEntryAsync](#) method queries a DNS server to map a user-friendly domain name (such as "host.contoso.com") to a numeric Internet address (such as `192.168.1.1`). [GetHostEntryAsync](#) returns a [Task<IPHostEntry>](#) that when awaited contains a list of addresses and aliases for the requested name. In most cases, you can use the first address returned in the [AddressList](#) array. The following code gets an [IPAddress](#) containing the IP address for the server `host.contoso.com`.

```
IPHostEntry ipHostInfo = await Dns.GetHostEntryAsync("host.contoso.com");
IPAddress ipAddress = ipHostInfo.AddressList[0];
```

TIP

For manual testing and debugging purposes, you can typically use the [GetHostEntryAsync](#) method to get given the [Dns.GetHostName\(\)](#) value to resolve the localhost name to an IP address.

The Internet Assigned Numbers Authority (IANA) defines port numbers for common services. For more information, see [IANA: Service Name and Transport Protocol Port Number Registry](#). Other services can have registered port numbers in the range 1,024 to 65,535. The following code combines the IP address for `host.contoso.com` with a port number to create a remote endpoint for a connection.

```
IPEndPoint ipEndPoint = new(ipAddress, 11_000);
```

After determining the address of the remote device and choosing a port to use for the connection, the app can establish a connection with the remote device.

Create a `TcpClient`

The `TcpClient` class provides TCP services at a higher level of abstraction than the `Socket` class. `TcpClient` is used to create a client connection to a remote host. Knowing how to get an `IPEndPoint`, let's assume you have an `IPAddress` to pair with your desired port number. The following example demonstrates setting up a `TcpClient` to connect to a time server on TCP port 13:

```
var ipEndPoint = new IPEndPoint(ipAddress, 13);

using TcpClient client = new();
await client.ConnectAsync(ipEndPoint);
await using NetworkStream stream = client.GetStream();

var buffer = new byte[1_024];
int received = await stream.ReadAsync(buffer);

var message = Encoding.UTF8.GetString(buffer, 0, received);
Console.WriteLine($"Message received: \"{message}\"");
// Sample output:
//     Message received: " 8/22/2022 9:07:17 AM "
```

The preceding C# code:

- Creates an `IPEndPoint` from a known `IPAddress` and port.
- Instantiate a new `TcpClient` object.
- Connects the `client` to the remote TCP time server on port 13 using `TcpClient.ConnectAsync`.
- Uses a `NetworkStream` to read data from the remote host.
- Declares a read buffer of `1_024` bytes.
- Reads data from the `stream` into the read buffer.
- Writes the results as a string to the console.

Since the client knows that the message is small, the entire message can be read into the read buffer in one operation. With larger messages, or messages with an indeterminate length, the client should use the buffer more appropriately and read in a `while` loop.

IMPORTANT

When sending and receiving messages, the `Encoding` should be known ahead of time to both server and client. For example, if the server communicates using `ASCIIEncoding` but the client attempts to use `UTF8Encoding`, the messages will be malformed.

Create a `TcpListener`

The `TcpListener` type is used to monitor a TCP port for incoming requests and then create either a `Socket` or a `TcpClient` that manages the connection to the client. The `Start` method enables listening, and the `Stop` method disables listening on the port. The `AcceptTcpClientAsync` method accepts incoming connection requests and creates a `TcpClient` to handle the request, and the `AcceptSocketAsync` method accepts incoming connection requests and creates a `Socket` to handle the request.

The following example demonstrates creating a network time server using a `TcpListener` to monitor TCP port 13. When an incoming connection request is accepted, the time server responds with the current date and time from the host server.

```

var ipEndPoint = new IPEndPoint(IPAddress.Any, 13);
TcpListener listener = new(ipEndPoint);

try
{
    listener.Start();

    using TcpClient handler = await listener.AcceptTcpClientAsync();
    await using NetworkStream stream = handler.GetStream();

    var message = $"@ {DateTime.Now} @";
    var dateTimeBytes = Encoding.UTF8.GetBytes(message);
    await stream.WriteAsync(dateTimeBytes);

    Console.WriteLine($"Sent message: \'{message}\'");
    // Sample output:
    //      Sent message: " 8/22/2022 9:07:17 AM "
}

finally
{
    listener.Stop();
}

```

The preceding C# code:

- Creates an `IPEndPoint` with `IPAddress.Any` and port.
- Instantiate a new `TcpListener` object.
- Calls the `Start` method to start listening on the port.
- Uses a `TcpClient` from the `AcceptTcpClientAsync` method to accept incoming connection requests.
- Encodes the current date and time as a string message.
- Uses a `NetworkStream` to write data to the connected client.
- Writes the sent message to the console.
- Finally, calls the `Stop` method to stop listening on the port.

See also

- [TCP in .NET](#)
- [Networking in .NET](#)
- [TcpClient](#)
- [TcpListener](#)
- [NetworkStream](#)

File globbing in .NET

9/20/2022 • 5 minutes to read • [Edit Online](#)

In this article, you'll learn how to use file globbing with the [Microsoft.Extensions.FileSystemGlobbing](#) NuGet package. A *glob* is a term used to define patterns for matching file and directory names based on wildcards. Globbing is the act of defining one or more glob patterns, and yielding files from either inclusive or exclusive matches.

Patterns

To match files in the file system based on user-defined patterns, start by instantiating a [Matcher](#) object. A [Matcher](#) can be instantiated with no parameters, or with a [System.StringComparison](#) parameter, which is used internally for comparing patterns to file names. The [Matcher](#) exposes the following additive methods:

- [Matcher.AddExclude](#)
- [Matcher.AddInclude](#)

Both [AddExclude](#) and [AddInclude](#) methods can be called any number of times, to add various file name patterns to either exclude or include from results. Once you've instantiated a [Matcher](#) and added patterns, it's then used to evaluate matches from a starting directory with the [Matcher.Execute](#) method.

Extension methods

The [Matcher](#) object has several extension methods.

Multiple exclusions

To add multiple exclude patterns, you can use:

```
Matcher matcher = new();
matcher.AddExclude("*.txt");
matcher.AddExclude("*.asciidoc");
matcher.AddExclude("*.md");
```

Alternatively, you can use the [MatcherExtensions.AddExcludePatterns\(Matcher, IEnumerable<String>\[\]\)](#) to add multiple exclude patterns in a single call:

```
Matcher matcher = new();
matcher.AddExcludePatterns(new [] { "*.txt", "*.asciidoc", "*.md" });
```

This extension method iterates over all of the provided patterns calling [AddExclude](#) on your behalf.

Multiple inclusions

To add multiple include patterns, you can use:

```
Matcher matcher = new();
matcher.AddInclude("*.txt");
matcher.AddInclude("*.asciidoc");
matcher.AddInclude("*.md");
```

Alternatively, you can use the [MatcherExtensions.AddIncludePatterns\(Matcher, IEnumerable<String>\[\]\)](#) to add

multiple include patterns in a single call:

```
Matcher matcher = new();
matcher.AddIncludePatterns(new[] { "*.txt", "*.asciidoc", "*.md" });
```

This extension method iterates over all of the provided patterns calling [AddInclude](#) on your behalf.

Get all matching files

To get all matching files, you have to call [Matcher.Execute\(DirectoryInfoBase\)](#) either directly or indirectly. To call it directly, you need a search directory:

```
Matcher matcher = new();
matcher.AddIncludePatterns(new[] { "*.txt", "*.asciidoc", "*.md" });

string searchDirectory = "../starting-folder/";

PatternMatchingResult result = matcher.Execute(
    new DirectoryInfoWrapper(
        new DirectoryInfo(searchDirectory)));

// Use result.HasMatches and results.Files.
// The files in the results object are file paths relative to the search directory.
```

The preceding C# code:

- Instantiates a [Matcher](#) object.
- Calls [AddIncludePatterns\(Matcher, IEnumerable<String>\[\]\)](#) to add several file name patterns to include.
- Declares and assigns the search directory value.
- Instantiates a [DirectoryInfo](#) from the given `searchDirectory`.
- Instantiates a [DirectoryInfoWrapper](#) from the `DirectoryInfo` it wraps.
- Calls `Execute` given the `DirectoryInfoWrapper` instance to yield a [PatternMatchingResult](#) object.

NOTE

The `DirectoryInfoWrapper` type is defined in the `Microsoft.Extensions.FileSystemGlobbing.Abstractions` namespace, and the `DirectoryInfo` type is defined in the `System.IO` namespace. To avoid unnecessary `using` statements, you can use the provided extension methods.

There is another extension method that yields an `IEnumerable<string>` representing the matching files:

```
Matcher matcher = new();
matcher.AddIncludePatterns(new[] { "*.txt", "*.asciidoc", "*.md" });

string searchDirectory = "../starting-folder/";

IEnumerable<string> matchingFiles = matcher.GetResultsInFullPath(searchDirectory);

// Use matchingFiles if there are any found.
// The files in this collection are fully qualified file system paths.
```

The preceding C# code:

- Instantiates a [Matcher](#) object.
- Calls [AddIncludePatterns\(Matcher, IEnumerable<String>\[\]\)](#) to add several file name patterns to include.
- Declares and assigns the search directory value.

- Calls `GetResultsInFullPath` given the `searchDirectory` value to yield all matching files as a `IEnumerable<string>`.

Match overloads

The `PatternMatchingResult` object represents a collection of `FilePatternMatch` instances, and exposes a `boolean` value indicating whether the result has matches—`PatternMatchingResult.HasMatches`.

With a `Matcher` instance, you can call any of the various `Match` overloads to get a pattern matching result. The `Match` methods invert the responsibility on the caller to provide a file or a collection of files in which to evaluate for matches. In other words, the caller is responsible for passing the file to match on.

IMPORTANT

When using any of the `Match` overloads, there is no file system I/O involved. All of the file globbing is done in memory with the include and exclude patterns of the `matcher` instance. The parameters of the `Match` overloads do not have to be fully qualified paths. The current directory (`Directory.GetCurrentDirectory()`) is used when not specified.

To match a single file:

```
Matcher matcher = new();
matcher.AddInclude("/**/*.md");

PatternMatchingResult result = matcher.Match("file.md");
```

The preceding C# code:

- Matches any file with the `.md` file extension, at an arbitrary directory depth.
- If a file named `file.md` exists in a subdirectory from the current directory:
 - `result.HasMatches` would be `true`.
 - and `result.Files` would have one match.

The additional `Match` overloads work in similar ways.

Pattern formats

The patterns that are specified in the `AddExclude` and `AddInclude` methods can use the following formats to match multiple files or directories.

- Exact directory or file name
 - `some-file.txt`
 - `path/to/file.txt`
- Wildcards `*` in file and directory names that represent zero to many characters not including separator characters.

VALUE	DESCRIPTION
<code>*.txt</code>	All files with <code>.txt</code> file extension.
<code>*.*</code>	All files with an extension.
<code>*</code>	All files in top-level directory.

VALUE	DESCRIPTION
<code>.*</code>	File names beginning with '..'.
<code>*word*</code>	All files with 'word' in the filename.
<code>readme.*</code>	All files named 'readme' with any file extension.
<code>styles/*.css</code>	All files with extension '.css' in the directory 'styles/'.
<code>scripts/**/*</code>	All files in 'scripts/' or one level of subdirectory under 'scripts/'.
<code>images**/*</code>	All files in a folder with name that is or begins with 'images'.

- Arbitrary directory depth (`/**/`).

VALUE	DESCRIPTION
<code>**/*</code>	All files in any subdirectory.
<code>dir/**/*</code>	All files in any subdirectory under 'dir/'.

- Relative paths.

To match all files in a directory named "shared" at the sibling level to the base directory given to [Matcher.Execute\(DirectoryInfoBase\)](#), use `../shared/*`.

Examples

Consider the following example directory, and each file within its corresponding folder.

```

parent
|   file.md
|   README.md
|
└── child
    |   file.MD
    |   index.js
    |   more.md
    |   sample.mtext
    |
    └── assets
        |   image.png
        |   image.svg
    |
    └── grandchild
        |   file.md
        |   style.css
        |   sub.text

```

TIP

Some file extensions are in uppercase, while others are in lowercase. By default, `StringComparer.OrdinalIgnoreCase` is used. To specify different string comparison behavior, use the `Matcher.Matcher(StringComparison)` constructor.

To get all of the markdown files, where the file extension is either `.md` or `.mtext`, regardless of character case:

```
Matcher matcher = new();
matcher.AddIncludePatterns(new[] { "**/*.md", "**/*.mtext" });

foreach (string file in matcher.GetResultsInFullPath("parent"))
{
    Console.WriteLine(file);
}
```

Running the application would output results similar to the following:

```
C:\app\parent\file.md
C:\app\parent\README.md
C:\app\parent\child\file.MD
C:\app\parent\child\more.md
C:\app\parent\child\sample.mtext
C:\app\parent\child\grandchild\file.md
```

To get any files in an `assets` directory at an arbitrary depth:

```
Matcher matcher = new();
matcher.AddInclude("*/assets/**/*");

foreach (string file in matcher.GetResultsInFullPath("parent"))
{
    Console.WriteLine(file);
}
```

Running the application would output results similar to the following:

```
C:\app\parent\child\assets\image.png
C:\app\parent\child\assets\image.svg
```

To get any files where the directory name contains the word `child` at an arbitrary depth, and the file extensions are not `.md`, `.text`, or `.mtext`:

```
Matcher matcher = new();
matcher.AddInclude("/**/*child/**/*");
matcher.AddExcludePatterns(
    new[]
    {
        "**/*.md", "**/*.text", "**/*.mtext"
    });
foreach (string file in matcher.GetResultsInFullPath("parent"))
{
    Console.WriteLine(file);
}
```

Running the application would output results similar to the following:

```
C:\app\parent\child\index.js
C:\app\parent\child\assets\image.png
C:\app\parent\child\assets\image.svg
C:\app\parent\child\grandchild\style.css
```

See also

- [Runtime libraries overview](#)
- [File and Stream I/O](#)

Primitives: The extensions library for .NET

9/20/2022 • 7 minutes to read • [Edit Online](#)

In this article, you'll learn about the [Microsoft.Extensions.Primitives](#) library. The primitives in this article are *not* to be confused with .NET primitive types from the BCL, or that of the C# language. Instead, the types within the primitives library serve as building blocks for some of the peripheral .NET NuGet packages, such as:

- [Microsoft.Extensions.Configuration](#)
- [Microsoft.Extensions.Configuration.FileExtensions](#)
- [Microsoft.Extensions.FileProviders.Composite](#)
- [Microsoft.Extensions.FileProviders.Physical](#)
- [Microsoft.Extensions.Logging.EventSource](#)
- [Microsoft.Extensions.Options](#)
- [System.Text.Json](#)

Change notifications

Propagating notifications when a change occurs is a fundamental concept in programming. The observed state of an object more often than not can change. When change occurs, implementations of the [Microsoft.Extensions.Primitives.IChangeToken](#) interface can be used to notify interested parties of said change. The implementations available are as follows:

- [CancellationChangeToken](#)
- [CompositeChangeToken](#)

As a developer, you're also free to implement your own type. The [IChangeToken](#) interface defines a few properties:

- [IChangeToken.HasChanged](#): Gets a value that indicates if a change has occurred.
- [IChangeToken.ActiveChangeCallbacks](#): Indicates if the token will proactively raise callbacks. If `false`, the token consumer must poll `HasChanged` to detect changes.

Instance-based functionality

Consider the following example usage of the [CancellationChangeToken](#):

```

CancellationTokenSource cancellationTokenSource = new();
CancellationChangeToken cancellationChangeToken = new(cancellationTokenSource.Token);

Console.WriteLine($"HasChanged: {cancellationChangeToken.HasChanged}");

Action<object?> callback = _ => Console.WriteLine("The callback was invoked.");

using (IDisposable subscription =
    cancellationChangeToken.RegisterChangeCallback(callback, null))
{
    cancellationTokenSource.Cancel();
}

Console.WriteLine($"HasChanged: {cancellationChangeToken.HasChanged}\n");

// Outputs:
//     HasChanged: False
//     The callback was invoked.
//     HasChanged: True

```

In the preceding example, a [CancellationTokenSource](#) is instantiated and its [Token](#) is passed to the [CancellationChangeToken](#) constructor. The initial state of `HasChanged` is written to the console. An `Action<object?> callback` is created that writes when the callback is invoked to the console. The token's [RegisterChangeCallback\(Action<Object>, Object\)](#) method is called, given the `callback`. Within the `using` statement, the `cancellationTokenSource` is cancelled. This triggers the callback, and the state of `HasChanged` is again written to the console.

When you need to take action from multiple sources of change, use the [CompositeChangeToken](#). This implementation aggregates one or more change tokens and fires each registered callback exactly one time regardless of the number of times a change is triggered. Consider the following example:

```

CancellationTokenSource firstCancellationTokenSource = new();
CancellationChangeToken firstCancellationChangeToken = new(firstCancellationTokenSource.Token);

CancellationTokenSource secondCancellationTokenSource = new();
CancellationChangeToken secondCancellationChangeToken = new(secondCancellationTokenSource.Token);

CancellationTokenSource thirdCancellationTokenSource = new();
CancellationChangeToken thirdCancellationChangeToken = new(thirdCancellationTokenSource.Token);

var compositeChangeToken =
    new CompositeChangeToken(
        new IChangeToken[]
    {
        firstCancellationChangeToken,
        secondCancellationChangeToken,
        thirdCancellationChangeToken
    });
}

Action<object?> callback = state => Console.WriteLine($"The {state} callback was invoked.");

// 1st, 2nd, 3rd, and 4th.
compositeChangeToken.RegisterChangeCallback(callback, "1st");
compositeChangeToken.RegisterChangeCallback(callback, "2nd");
compositeChangeToken.RegisterChangeCallback(callback, "3rd");
compositeChangeToken.RegisterChangeCallback(callback, "4th");

// It doesn't matter which cancellation source triggers the change.
// If more than one trigger the change, each callback is only fired once.
Random random = new();
int index = random.Next(3);
CancellationTokenSource[] sources = new[]
{
    firstCancellationTokenSource,
    secondCancellationTokenSource,
    thirdCancellationTokenSource
};
sources[index].Cancel();

Console.WriteLine();

// Outputs:
//      The 4th callback was invoked.
//      The 3rd callback was invoked.
//      The 2nd callback was invoked.
//      The 1st callback was invoked.

```

In the preceding C# code, three `CancellationTokenSource` objects instances are created and paired with corresponding `CancellationChangeToken` instances. The composite token is instantiated by passing an array of the tokens to the `CompositeChangeToken` constructor. The `Action<object?> callback` is created, but this time the `state` object is used and written to console as a formatted message. The callback is registered four times, each with a slightly different state object argument. The code uses a pseudo-random number generator to pick one of the change token sources (doesn't matter which one) and call its `Cancel()` method. This triggers the change, invoking each registered callback exactly once.

Alternative `static` approach

As an alternative to calling `RegisterChangeCallback`, you could use the `Microsoft.Extensions.Primitives.ChangeToken` static class. Consider the following consumption pattern:

```

CancellationTokenSource cancellationTokenSource = new();
CancellationChangeToken cancellationChangeToken = new(cancellationTokenSource.Token);

Func<IChangeToken> producer = () =>
{
    // The producer factory should always return a new change token.
    // If the token's already fired, get a new token.
    if (cancellationTokenSource.IsCancellationRequested)
    {
        cancellationTokenSource = new();
        cancellationChangeToken = new(cancellationTokenSource.Token);
    }

    return cancellationChangeToken;
};

Action consumer = () => Console.WriteLine("The callback was invoked.");

using (ChangeToken.OnChange(producer, consumer))
{
    cancellationTokenSource.Cancel();
}

// Outputs:
//     The callback was invoked.

```

Much like previous examples, you'll need an implementation of `IChangeToken` that is produced by the `changeTokenProducer`. The producer is defined as a `Func<IChangeToken>` and it's expected that this will return a new token every invocation. The `consumer` is either an `Action` when not using `state`, or an `Action<TState>` where the generic type `TState` flows through the change notification.

String tokenizers, segments, and values

Interacting with strings is commonplace in application development. Various representations of strings are parsed, split, or iterated over. The primitives library offers a few choice types that help to make interacting with strings more optimized and efficient. Consider the following types:

- **`StringSegment`**: An optimized representation of a substring.
- **`StringTokenizer`**: Tokenizes a `string` into `StringSegment` instances.
- **`StringValues`**: Represents `null`, zero, one, or many strings in an efficient way.

The `StringSegment` type

In this section, you'll learn about an optimized representation of a substring known as the **`StringSegment`** `struct` type. Consider the following C# code example showing some of the `StringSegment` properties and the `AsSpan` method:

```

var segment =
    new StringSegment(
        "This a string, within a single segment representation.",
        14, 25);

Console.WriteLine($"Buffer: \'{segment.Buffer}\'");
Console.WriteLine($"Offset: {segment.Offset}");
Console.WriteLine($"Length: {segment.Length}");
Console.WriteLine($"Value: \'{segment.Value}\'");

Console.Write("Span: \"");
foreach (char @char in segment.AsSpan())
{
    Console.Write(@char);
}
Console.WriteLine("\n");

// Outputs:
//     Buffer: "This a string, within a single segment representation."
//     Offset: 14
//     Length: 25
//     Value: " within a single segment "
//     " within a single segment "

```

The preceding code instantiates the `StringSegment` given a `string` value, an `offset`, and a `length`. The `StringSegment.Buffer` is the original string argument, and the `StringSegment.Value` is the substring based on the `StringSegment.Offset` and `StringSegment.Length` values.

The `StringSegment` struct provides [many methods](#) for interacting with the segment.

The `StringTokenizer` type

The `StringTokenizer` object is a struct type that tokenizes a `string` into `StringSegment` instances. The tokenization of large strings usually involves splitting the string apart and iterating over it. With that said, `String.Split` probably comes to mind. These APIs are similar, but in general, `StringTokenizer` provides better performance. First, consider the following example:

```

var tokenizer =
    new StringTokenizer(
        s_nineHundredAutoGeneratedParagraphsOfLoremIpsum,
        new[] { ' ' });

foreach (StringSegment segment in tokenizer)
{
    // Interact with segment
}

```

In the preceding code, an instance of the `StringTokenizer` type is created given 900 auto-generated paragraphs of Lorem Ipsum text and an array with a single value of a white-space character `' '`. Each value within the tokenizer is represented as a `StringSegment`. The code iterates the segments, allowing the consumer to interact with each `segment`.

Benchmark comparing `StringTokenizer` to `string.Split`

With the various ways of slicing and dicing strings, it feels appropriate to compare two methods with a benchmark. Using the [BenchmarkDotNet](#) NuGet package, consider the following two benchmark methods:

1. Using `StringTokenizer`:

```

StringBuilder buffer = new();

var tokenizer =
    new StringTokenizer(
        s_nineHundredAutoGeneratedParagraphsOfLoremIpsum,
        new[] { ' ', '.' });

foreach (StringSegment segment in tokenizer)
{
    buffer.Append(segment.Value);
}

```

2. Using `String.Split`:

```

StringBuilder buffer = new();

string[] tokenizer =
    s_nineHundredAutoGeneratedParagraphsOfLoremIpsum.Split(
        new[] { ' ', '.' });

foreach (string segment in tokenizer)
{
    buffer.Append(segment);
}

```

Both methods look similar on the API surface area, and they're both capable of splitting a large string into chunks. The benchmark results below show that the `StringTokenizer` approach is nearly three times faster, but *results may vary*. As with all performance considerations, you should evaluate your specific use case.

METHOD	MEAN	ERROR	STANDARD DEVIATION	MEDIAN	RATIO	RATIO STANDARD DEVIATION
Tokenizer	6.306 ms	0.1481 ms	0.4179 ms	6.175 ms	0.37	0.04
Split	16.966 ms	0.6164 ms	1.8079 ms	16.862 ms	1.00	0.00

Legend

- Mean: Arithmetic mean of all measurements
- Error: Half of 99.9% confidence interval
- Standard deviation: Standard deviation of all measurements
- Median: Value separating the higher half of all measurements (50th percentile)
- Ratio: Mean of the ratio distribution (Current/Baseline)
- Ratio standard deviation: Standard deviation of the ratio distribution (Current/Baseline)
- 1 ms: 1 Millisecond (0.001 sec)

For more information on benchmarking with .NET, see [BenchmarkDotNet](#).

The `StringValues` type

The `StringValues` object is a `struct` type that represents `null`, zero, one, or many strings in an efficient way.

The `StringValues` type can be constructed with either of the following syntaxes: `string?` or `string?[]?`. Using the text from the previous example, consider the following C# code:

```
StringValues values =
    new(s_nineHundredAutoGeneratedParagraphsOfLoremIpsum.Split(
        new[] { '\n' }));

Console.WriteLine($"Count = {values.Count:#,##}");

foreach (string? value in values)
{
    // Interact with the value
}
// Outputs:
//      Count = 1,799
```

The preceding code instantiates a `StringValues` object given an array of string values. The `StringValues.Count` is written to the console.

The `StringValues` type is an implementation of the following collection types:

- `IList<string>`
- `ICollection<string>`
- `IEnumerable<string>`
- `IEnumerable`
- `IReadOnlyList<string>`
- `IReadOnlyCollection<string>`

As such, it can be iterated over and each `value` can be interacted with as needed.

See also

- [Options pattern in .NET](#)
- [Configuration in .NET](#)
- [Logging providers in .NET](#)

Globalize and localize .NET applications

9/20/2022 • 2 minutes to read • [Edit Online](#)

Developing a world-ready application, including an application that can be localized into one or more languages, involves three steps: globalization, localizability review, and localization.

Globalization

This step involves designing and coding an application that is culture-neutral and language-neutral, and that supports localized user interfaces and regional data for all users. It involves making design and programming decisions that are not based on culture-specific assumptions. While a globalized application is not localized, it nevertheless is designed and written so that it can be subsequently localized into one or more languages with relative ease.

Localizability review

This step involves reviewing an application's code and design to ensure that it can be localized easily and to identify potential roadblocks for localization, and verifying that the application's executable code is separated from its resources. If the globalization stage was effective, the localizability review will confirm the design and coding choices made during globalization. The localizability stage may also identify any remaining issues so that an application's source code doesn't have to be modified during the localization stage.

Localization

This step involves customizing an application for specific cultures or regions. If the globalization and localizability steps have been performed correctly, localization consists primarily of translating the user interface.

Following these three steps provides two advantages:

- It frees you from having to retrofit an application that is designed to support a single culture, such as U.S. English, to support additional cultures.
- It results in localized applications that are more stable and have fewer bugs.

.NET provides extensive support for the development of world-ready and localized applications. In particular, many type members in the .NET class library aid globalization by returning values that reflect the conventions of either the current user's culture or a specified culture. Also, .NET supports satellite assemblies, which facilitate the process of localizing an application.

In this section

Globalization

Discusses the first stage of creating a world-ready application, which involves designing and coding an application that is culture-neutral and language-neutral.

.NET globalization and ICU

Describes how .NET globalization uses [International Components for Unicode \(ICU\)](#).

Localizability review

Discusses the second stage of creating a localized application, which involves identifying potential roadblocks to localization.

Localization

Discusses the final stage of creating a localized application, which involves customizing an application's user interface for specific regions or cultures.

[Culture-insensitive string operations](#)

Describes how to use .NET methods and classes that are culture-sensitive by default to obtain culture-insensitive results.

[Best practices for developing world-ready applications](#)

Describes the best practices to follow for globalization, localization, and developing world-ready ASP.NET applications.

Reference

- [System.Globalization](#) namespace

Contains classes that define culture-related information, including the language, the country/region, the calendars in use, the format patterns for dates, currency, and numbers, and the sort order for strings.

- [System.Resources](#) namespace

Provides classes for creating, manipulating, and using resources.

- [System.Text](#) namespace

Contains classes representing ASCII, ANSI, Unicode, and other character encodings.

- [Resgen.exe \(Resource File Generator\)](#)

Describes how to use Resgen.exe to convert .txt files and XML-based resource format (.resx) files to common language runtime binary .resources files.

- [Winres.exe \(Windows Forms Resource Editor\)](#)

Describes how to use Winres.exe to localize Windows Forms forms.

Globalization

9/20/2022 • 50 minutes to read • [Edit Online](#)

Globalization involves designing and developing a world-ready app that supports localized interfaces and regional data for users in multiple cultures. Before beginning the design phase, you should determine which cultures your app will support. Although an app targets a single culture or region as its default, you can design and write it so that it can easily be extended to users in other cultures or regions.

As developers, we all have assumptions about user interfaces and data that are formed by our cultures. For example, for an English-speaking developer in the United States, serializing date and time data as a string in the format `MM/dd/yyyy hh:mm:ss` seems perfectly reasonable. However, deserializing that string on a system in a different culture is likely to throw a [FormatException](#) exception or produce inaccurate data. Globalization enables us to identify such culture-specific assumptions and ensure that they do not affect our app's design or code.

This article discusses some of the major issues you should consider and the best practices you can follow when handling strings, date and time values, and numeric values in a globalized app.

Strings

The handling of characters and strings is a central focus of globalization, because each culture or region may use different characters and character sets and sort them differently. This section provides recommendations for using strings in globalized apps.

Use Unicode internally

By default, .NET uses Unicode strings. A Unicode string consists of zero, one, or more [Char](#) objects, each of which represents a UTF-16 code unit. There is a Unicode representation for almost every character in every character set in use throughout the world.

Many applications and operating systems, including the Windows operating system, can also use code pages to represent character sets. Code pages typically contain the standard ASCII values from 0x00 through 0x7F and map other characters to the remaining values from 0x80 through 0xFF. The interpretation of values from 0x80 through 0xFF depends on the specific code page. Because of this, you should avoid using code pages in a globalized app if possible.

The following example illustrates the dangers of interpreting code page data when the default code page on a system is different from the code page on which the data was saved. (To simulate this scenario, the example explicitly specifies different code pages.) First, the example defines an array that consists of the uppercase characters of the Greek alphabet. It encodes them into a byte array by using code page 737 (also known as MS-DOS Greek) and saves the byte array to a file. If the file is retrieved and its byte array is decoded by using code page 737, the original characters are restored. However, if the file is retrieved and its byte array is decoded by using code page 1252 (or Windows-1252, which represents characters in the Latin alphabet), the original characters are lost.

```

using System;
using System.IO;
using System.Text;

public class Example
{
    public static void Main()
    {
        // Represent Greek uppercase characters in code page 737.
        char[] greekChars =
        {
            'Α', 'Β', 'Γ', 'Δ', 'Ε', 'Ζ', 'Η', 'Θ',
            'Ι', 'Κ', 'Λ', 'Μ', 'Ν', 'Ξ', 'Ο', 'Π',
            'Ρ', 'Σ', 'Τ', 'Υ', 'Φ', 'Χ', 'Ψ', 'Ω'
        };

        Encoding.RegisterProvider(CodePagesEncodingProvider.Instance);

        Encoding cp737 = Encoding.GetEncoding(737);
        int nBytes = cp737.GetByteCount(greekChars);
        byte[] bytes737 = new byte[nBytes];
        bytes737 = cp737.GetBytes(greekChars);
        // Write the bytes to a file.
        FileStream fs = new FileStream(@"..\CodePageBytes.dat", FileMode.Create);
        fs.Write(bytes737, 0, bytes737.Length);
        fs.Close();

        // Retrieve the byte data from the file.
        fs = new FileStream(@"..\CodePageBytes.dat", FileMode.Open);
        byte[] bytes1 = new byte[fs.Length];
        fs.Read(bytes1, 0, (int)fs.Length);
        fs.Close();

        // Restore the data on a system whose code page is 737.
        string data = cp737.GetString(bytes1);
        Console.WriteLine(data);
        Console.WriteLine();

        // Restore the data on a system whose code page is 1252.
        Encoding cp1252 = Encoding.GetEncoding(1252);
        data = cp1252.GetString(bytes1);
        Console.WriteLine(data);
    }
}

// The example displays the following output:
//      ΑΒΓΔΕΖΗΘΙΚΛΜΝΞΟΠΡΣΤΥΦΧΨΩ
//      €,ƒ„†‡^‰Š<ŒŽ‘’”•—

```

```

Imports System.IO
Imports System.Text

Module Example
    Public Sub Main()
        ' Represent Greek uppercase characters in code page 737.
        Dim greekChars() As Char = {"Α"c, "Β"c, "Γ"c, "Δ"c, "Ε"c, "Ζ"c, "Η"c, "Θ"c,
                                    "Ι"c, "Κ"c, "Λ"c, "Μ"c, "Ν"c, "Ξ"c, "Ο"c, "Π"c,
                                    "Ρ"c, "Σ"c, "Τ"c, "Υ"c, "Φ"c, "Χ"c, "Ψ"c, "Ω"c}

        Encoding.RegisterProvider(CodePagesEncodingProvider.Instance)

        Dim cp737 As Encoding = Encoding.GetEncoding(737)
        Dim nBytes As Integer = CInt(cp737.GetByteCount(greekChars))
        Dim bytes737(nBytes - 1) As Byte
        bytes737 = cp737.GetBytes(greekChars)
        ' Write the bytes to a file.
        Dim fs As New FileStream(".\CodePageBytes.dat", FileMode.Create)
        fs.Write(bytes737, 0, bytes737.Length)
        fs.Close()

        ' Retrieve the byte data from the file.
        fs = New FileStream(".\CodePageBytes.dat", FileMode.Open)
        Dim bytes1(CInt(fs.Length - 1)) As Byte
        fs.Read(bytes1, 0, CInt(fs.Length))
        fs.Close()

        ' Restore the data on a system whose code page is 737.
        Dim data As String = cp737.GetString(bytes1)
        Console.WriteLine(data)
        Console.WriteLine()

        ' Restore the data on a system whose code page is 1252.
        Dim cp1252 As Encoding = Encoding.GetEncoding(1252)
        data = cp1252.GetString(bytes1)
        Console.WriteLine(data)
    End Sub
End Module

' The example displays the following output:
' ΑΒΓΔΕΖΗΟΙΚΛΜΝΞΟΠΡΣΤΥΦΧΨΩ
' €.f,...†‡~‰$<ŒŽ‘”•—

```

The use of Unicode ensures that the same code units always map to the same characters, and that the same characters always map to the same byte arrays.

Use resource files

Even if you are developing an app that targets a single culture or region, you should use resource files to store strings and other resources that are displayed in the user interface. You should never add them directly to your code. Using resource files has a number of advantages:

- All the strings are in a single location. You don't have to search throughout your source code to identify strings to modify for a specific language or culture.
 - There is no need to duplicate strings. Developers who don't use resource files often define the same string in multiple source code files. This duplication increases the probability that one or more instances will be overlooked when a string is modified.
 - You can include non-string resources, such as images or binary data, in the resource file instead of storing them in a separate standalone file, so they can be retrieved easily.

Using resource files has particular advantages if you are creating a localized app. When you deploy resources in satellite assemblies, the common language runtime automatically selects a culture-appropriate resource based on the user's current UI culture as defined by the [CultureInfo.CurrentCulture](#) property. As long as you provide

an appropriate culture-specific resource and correctly instantiate a `ResourceManager` object or use a strongly typed resource class, the runtime handles the details of retrieving the appropriate resources.

For more information about creating resource files, see [Creating resource files](#). For information about creating and deploying satellite assemblies, see [Create satellite assemblies](#) and [Package and Deploy resources](#).

Search and compare strings

Whenever possible, you should handle strings as entire strings instead of handling them as a series of individual characters. This is especially important when you sort or search for substrings, to prevent problems associated with parsing combined characters.

TIP

You can use the `StringInfo` class to work with the text elements rather than the individual characters in a string.

In string searches and comparisons, a common mistake is to treat the string as a collection of characters, each of which is represented by a `Char` object. In fact, a single character may be formed by one, two, or more `Char` objects. Such characters are found most frequently in strings from cultures whose alphabets consist of characters outside the Unicode Basic Latin character range (U+0021 through U+007E). The following example tries to find the index of the LATIN CAPITAL LETTER A WITH GRAVE character (U+00C0) in a string. However, this character can be represented in two different ways: as a single code unit (U+00C0) or as a composite character (two code units: U+0041 and U+0300). In this case, the character is represented in the string instance by two `Char` objects, U+0041 and U+0300. The example code calls the `String.IndexOf(Char)` and `String.IndexOf(String)` overloads to find the position of this character in the string instance, but these return different results. The first method call has a `Char` argument; it performs an ordinal comparison and therefore cannot find a match. The second call has a `String` argument; it performs a culture-sensitive comparison and therefore finds a match.

```
using System;
using System.Globalization;
using System.Threading;

public class Example17
{
    public static void Main17()
    {
        Thread.CurrentThread.CurrentCulture = CultureInfo.CreateSpecificCulture("pl-PL");
        string composite = "\u0041\u0300";
        Console.WriteLine("Comparing using Char: {0}", composite.IndexOf('\u00C0'));
        Console.WriteLine("Comparing using String: {0}", composite.IndexOf("\u00C0"));
    }
}
// The example displays the following output:
//      Comparing using Char: -1
//      Comparing using String: 0
```

```

Imports System.Globalization
Imports System.Threading

Module Example17
    Public Sub Main17()
        Thread.CurrentThread.CurrentCulture = CultureInfo.CreateSpecificCulture("pl-PL")
        Dim composite As String = ChrW(&H41) + ChrW(&H300)
        Console.WriteLine("Comparing using Char: {0}", composite.IndexOf(ChrW(&HC0)))
        Console.WriteLine("Comparing using String: {0}", composite.IndexOf(ChrW(&HC0).ToString()))
    End Sub
End Module
' The example displays the following output:
'     Comparing using Char: -1
'     Comparing using String: 0

```

You can avoid some of the ambiguity of this example (calls to two similar overloads of a method returning different results) by calling an overload that includes a [StringComparison](#) parameter, such as the [String.IndexOf\(String, StringComparison\)](#) or [String.LastIndexOf\(String, StringComparison\)](#) method.

However, searches are not always culture-sensitive. If the purpose of the search is to make a security decision or to allow or disallow access to some resource, the comparison should be ordinal, as discussed in the next section.

Test strings for equality

If you want to test two strings for equality rather than determine how they compare in the sort order, use the [String.Equals](#) method instead of a string comparison method such as [String.Compare](#) or [CompareInfo.Compare](#).

Comparisons for equality are typically performed to access some resource conditionally. For example, you might perform a comparison for equality to verify a password or to confirm that a file exists. Such non-linguistic comparisons should always be ordinal rather than culture-sensitive. In general, you should call the instance [String.Equals\(String, StringComparison\)](#) method or the static [String.Equals\(String, String, StringComparison\)](#) method with a value of [StringComparison.OrdinalIgnoreCase](#) for strings such as passwords, and a value of [StringComparison.OrdinalIgnoreCaseIgnoreCase](#) for strings such as file names or URLs.

Comparisons for equality sometimes involve searches or substring comparisons rather than calls to the [String.Equals](#) method. In some cases, you may use a substring search to determine whether that substring equals another string. If the purpose of this comparison is non-linguistic, the search should also be ordinal rather than culture-sensitive.

The following example illustrates the danger of a culture-sensitive search on non-linguistic data. The [AccessesFileSystem](#) method is designed to prohibit file system access for URIs that begin with the substring "FILE". To do this, it performs a culture-sensitive, case-insensitive comparison of the beginning of the URI with the string "FILE". Because a URI that accesses the file system can begin with either "FILE:" or "file:", the implicit assumption is that "i" (U+0069) is always the lowercase equivalent of "I" (U+0049). However, in Turkish and Azerbaijani, the uppercase version of "i" is "İ" (U+0130). Because of this discrepancy, the culture-sensitive comparison allows file system access when it should be prohibited.

```

using System;
using System.Globalization;
using System.Threading;

public class Example10
{
    public static void Main10()
    {
        Thread.CurrentThread.CurrentCulture = CultureInfo.CreateSpecificCulture("tr-TR");
        string uri = @"file:\\c:\\users\\username\\Documents\\bio.txt";
        if (!AccessesFileSystem(uri))
            // Permit access to resource specified by URI
            Console.WriteLine("Access is allowed.");
        else
            // Prohibit access.
            Console.WriteLine("Access is not allowed.");
    }

    private static bool AccessesFileSystem(string uri)
    {
        return uri.StartsWith("FILE", true, CultureInfo.CurrentCulture);
    }
}

// The example displays the following output:
//      Access is allowed.

```

```

Imports System.Globalization
Imports System.Threading

Module Example10
    Public Sub Main10()
        Thread.CurrentThread.CurrentCulture = CultureInfo.CreateSpecificCulture("tr-TR")
        Dim uri As String = "file:\\c:\\users\\username\\Documents\\bio.txt"
        If Not AccessesFileSystem(uri) Then
            ' Permit access to resource specified by URI
            Console.WriteLine("Access is allowed.")
        Else
            ' Prohibit access.
            Console.WriteLine("Access is not allowed.")
        End If
    End Sub

    Private Function AccessesFileSystem(uri As String) As Boolean
        Return uri.StartsWith("FILE", True, CultureInfo.CurrentCulture)
    End Function
End Module

' The example displays the following output:
'      Access is allowed.

```

You can avoid this problem by performing an ordinal comparison that ignores case, as the following example shows.

```

using System;
using System.Globalization;
using System.Threading;

public class Example11
{
    public static void Main11()
    {
        Thread.CurrentThread.CurrentCulture = CultureInfo.CreateSpecificCulture("tr-TR");
        string uri = @"file:\\c:\\users\\username\\Documents\\bio.txt";
        if (!AccessesFileSystem(uri))
            // Permit access to resource specified by URI
            Console.WriteLine("Access is allowed.");
        else
            // Prohibit access.
            Console.WriteLine("Access is not allowed.");
    }

    private static bool AccessesFileSystem(string uri)
    {
        return uri.StartsWith("FILE", StringComparison.OrdinalIgnoreCase);
    }
}

// The example displays the following output:
//      Access is not allowed.

```

```

Imports System.Globalization
Imports System.Threading

Module Example11
    Public Sub Main11()
        Thread.CurrentThread.CurrentCulture = CultureInfo.CreateSpecificCulture("tr-TR")
        Dim uri As String = "file:\\c:\\users\\username\\Documents\\bio.txt"
        If Not AccessesFileSystem(uri) Then
            ' Permit access to resource specified by URI
            Console.WriteLine("Access is allowed.")
        Else
            ' Prohibit access.
            Console.WriteLine("Access is not allowed.")
        End If
    End Sub

    Private Function AccessesFileSystem(uri As String) As Boolean
        Return uri.StartsWith("FILE", StringComparison.OrdinalIgnoreCase)
    End Function
End Module

' The example displays the following output:
'      Access is not allowed.

```

Order and sort strings

Typically, ordered strings that are to be displayed in the user interface should be sorted based on culture. For the most part, such string comparisons are handled implicitly by .NET when you call a method that sorts strings, such as [Array.Sort](#) or [List<T>.Sort](#). By default, strings are sorted by using the sorting conventions of the current culture. The following example illustrates the difference when an array of strings is sorted by using the conventions of the English (United States) culture and the Swedish (Sweden) culture.

```
using System;
using System.Globalization;
using System.Threading;

public class Example18
{
    public static void Main18()
    {
        string[] values = { "able", "ångström", "apple", "Æble",
                            "Windows", "Visual Studio" };
        // Change thread to en-US.
        Thread.CurrentThread.CurrentCulture = CultureInfo.CreateSpecificCulture("en-US");
        // Sort the array and copy it to a new array to preserve the order.
        Array.Sort(values);
        string[] enValues = (String[]) values.Clone();

        // Change culture to Swedish (Sweden).
        Thread.CurrentThread.CurrentCulture = CultureInfo.CreateSpecificCulture("sv-SE");
        Array.Sort(values);
        string[] svValues = (String[]) values.Clone();

        // Compare the sorted arrays.
        Console.WriteLine("{0,-8} {1,-15} {2,-15}\n", "Position", "en-US", "sv-SE");
        for (int ctr = 0; ctr <= values.GetUpperBound(0); ctr++)
            Console.WriteLine("{0,-8} {1,-15} {2,-15}", ctr, enValues[ctr], svValues[ctr]);
    }
}

// The example displays the following output:
//      Position en-US          sv-SE
//
//      0      able      able
//      1      Æble      Æble
//      2      ångström    apple
//      3      apple      Windows
//      4      Visual Studio  Visual Studio
//      5      Windows     ångström
```

```

Imports System.Globalization
Imports System.Threading

Module Example18
    Public Sub Main18()
        Dim values() As String = {"able", "ångström", "apple",
                                "Æble", "Windows", "Visual Studio"}
        ' Change thread to en-US.
        Thread.CurrentThread.CurrentCulture = CultureInfo.CreateSpecificCulture("en-US")
        ' Sort the array and copy it to a new array to preserve the order.
        Array.Sort(values)
        Dim enValues() As String = CType(values.Clone(), String())

        ' Change culture to Swedish (Sweden).
        Thread.CurrentThread.CurrentCulture = New CultureInfo("sv-SE")
        Array.Sort(values)
        Dim svValues() As String = CType(values.Clone(), String())

        ' Compare the sorted arrays.
        Console.WriteLine("{0,-8} {1,-15} {2,-15}", "Position", "en-US", "sv-SE")
        Console.WriteLine()
        For ctr As Integer = 0 To values.GetUpperBound(0)
            Console.WriteLine("{0,-8} {1,-15} {2,-15}", ctr, enValues(ctr), svValues(ctr))
        Next
    End Sub
End Module
' The example displays the following output:
'   Position en-US          sv-SE
'
'   0      able           able
'   1      Æble           Æble
'   2      ångström       apple
'   3      apple          Windows
'   4      Visual Studio  Visual Studio
'   5      Windows         ångström

```

Culture-sensitive string comparison is defined by the [CompareInfo](#) object, which is returned by each culture's [CultureInfo.CompareInfo](#) property. Culture-sensitive string comparisons that use the [String.Compare](#) method overloads also use the [CompareInfo](#) object.

.NET uses tables to perform culture-sensitive sorts on string data. The content of these tables, which contain data on sort weights and string normalization, is determined by the version of the Unicode standard implemented by a particular version of .NET. The following table lists the versions of Unicode implemented by the specified versions of .NET. This list of supported Unicode versions applies to character comparison and sorting only; it does not apply to classification of Unicode characters by category. For more information, see the "Strings and The Unicode Standard" section in the [String](#) article.

.NET FRAMEWORK VERSION	OPERATING SYSTEM	UNICODE VERSION
.NET Framework 2.0	All operating systems	Unicode 4.1
.NET Framework 3.0	All operating systems	Unicode 4.1
.NET Framework 3.5	All operating systems	Unicode 4.1
.NET Framework 4	All operating systems	Unicode 5.0
.NET Framework 4.5 and later on Windows 7	Unicode 5.0	

.NET FRAMEWORK VERSION	OPERATING SYSTEM	UNICODE VERSION
.NET Framework 4.5 and later on Windows 8 and later operating systems		Unicode 6.3.0
.NET Core and .NET 5+		Depends on the version of the Unicode Standard supported by the underlying operating system.

Starting with .NET Framework 4.5 and in all versions of .NET Core and .NET 5+, string comparison and sorting depends on the operating system. .NET Framework 4.5 and later running on Windows 7 retrieves data from its own tables that implement Unicode 5.0. .NET Framework 4.5 and later running on Windows 8 and later retrieves data from operating system tables that implement Unicode 6.3. On .NET Core and .NET 5+, the supported version of Unicode depends on the underlying operating system. If you serialize culture-sensitive sorted data, you can use the [SortVersion](#) class to determine when your serialized data needs to be sorted so that it is consistent with .NET and the operating system's sort order. For an example, see the [SortVersion](#) class topic.

If your app performs extensive culture-specific sorts of string data, you can work with the [SortKey](#) class to compare strings. A sort key reflects the culture-specific sort weights, including the alphabetic, case, and diacritic weights of a particular string. Because comparisons using sort keys are binary, they are faster than comparisons that use a [CompareInfo](#) object either implicitly or explicitly. You create a culture-specific sort key for a particular string by passing the string to the [CompareInfo.GetSortKey](#) method.

The following example is similar to the previous example. However, instead of calling the [Array.Sort\(Array\)](#) method, which implicitly calls the [CompareInfo.Compare](#) method, it defines an [System.Collections.Generic.IComparer<T>](#) implementation that compares sort keys, which it instantiates and passes to the [Array.Sort<T>\(T\[\], IComparer<T>\)](#) method.

```

using System;
using System.Collections.Generic;
using System.Globalization;
using System.Threading;

public class SortKeyComparer : IComparer<String>
{
    public int Compare(string str1, string str2)
    {
        SortKey sk1, sk2;
        sk1 = CultureInfo.CurrentCulture.CompareInfo.GetSortKey(str1);
        sk2 = CultureInfo.CurrentCulture.CompareInfo.GetSortKey(str2);
        return SortKey.Compare(sk1, sk2);
    }
}

public class Example19
{
    public static void Main19()
    {
        string[] values = { "able", "ångström", "apple", "Æble",
                            "Windows", "Visual Studio" };
        SortKeyComparer comparer = new SortKeyComparer();

        // Change thread to en-US.
        Thread.CurrentThread.CurrentCulture = CultureInfo.CreateSpecificCulture("en-US");
        // Sort the array and copy it to a new array to preserve the order.
        Array.Sort(values, comparer);
        string[] enValues = (String[]) values.Clone();

        // Change culture to Swedish (Sweden).
        Thread.CurrentThread.CurrentCulture = CultureInfo.CreateSpecificCulture("sv-SE");
        Array.Sort(values, comparer);
        string[] svValues = (String[]) values.Clone();

        // Compare the sorted arrays.
        Console.WriteLine("{0,-8} {1,-15} {2,-15}\n", "Position", "en-US", "sv-SE");
        for (int ctr = 0; ctr <= values.GetUpperBound(0); ctr++)
            Console.WriteLine("{0,-8} {1,-15} {2,-15}", ctr, enValues[ctr], svValues[ctr]);
    }
}

// The example displays the following output:
//      Position en-US          sv-SE
//
//      0      able           able
//      1      Æble           Æble
//      2      ångström       apple
//      3      apple          Windows
//      4      Visual Studio  Visual Studio
//      5      Windows         ångström

```

```

Imports System.Collections.Generic
Imports System.Globalization
Imports System.Threading

Public Class SortKeyComparer : Implements IComparer(Of String)
    Public Function Compare(str1 As String, str2 As String) As Integer _
        Implements IComparer(Of String).Compare
        Dim sk1, sk2 As SortKey
        sk1 = CultureInfo.CurrentCulture.CompareInfo.GetSortKey(str1)
        sk2 = CultureInfo.CurrentCulture.CompareInfo.GetSortKey(str2)
        Return SortKey.Compare(sk1, sk2)
    End Function
End Class

Module Example19
    Public Sub Main19()
        Dim values() As String = {"able", "ångström", "apple",
                                  "Æble", "Windows", "Visual Studio"}
        Dim comparer As New SortKeyComparer()

        ' Change thread to en-US.
        Thread.CurrentThread.CurrentCulture = CultureInfo.CreateSpecificCulture("en-US")
        ' Sort the array and copy it to a new array to preserve the order.
        Array.Sort(values, comparer)
        Dim enValues() As String = CType(values.Clone(), String())

        ' Change culture to Swedish (Sweden).
        Thread.CurrentThread.CurrentCulture = New CultureInfo("sv-SE")
        Array.Sort(values, comparer)
        Dim svValues() As String = CType(values.Clone(), String())

        ' Compare the sorted arrays.
        Console.WriteLine("{0,-8} {1,-15} {2,-15}", "Position", "en-US", "sv-SE")
        Console.WriteLine()
        For ctr As Integer = 0 To values.GetUpperBound(0)
            Console.WriteLine("{0,-8} {1,-15} {2,-15}", ctr, enValues(ctr), svValues(ctr))
        Next
    End Sub
End Module

```

' The example displays the following output:

	Position	en-US	sv-SE
'			
'	0	able	able
'	1	Æble	Æble
'	2	ångström	apple
'	3	apple	Windows
'	4	Visual Studio	Visual Studio
'	5	Windows	ångström

Avoid string concatenation

If at all possible, avoid using composite strings that are built at run time from concatenated phrases. Composite strings are difficult to localize, because they often assume a grammatical order in the app's original language that does not apply to other localized languages.

Handle dates and times

How you handle date and time values depends on whether they are displayed in the user interface or persisted. This section examines both usages. It also discusses how you can handle time zone differences and arithmetic operations when working with dates and times.

Display dates and times

Typically, when dates and times are displayed in the user interface, you should use the formatting conventions of

the user's culture, which is defined by the `CultureInfo.CurrentCulture` property and by the `DateTimeFormatInfo` object returned by the `CultureInfo.CurrentCulture.DateTimeFormat` property. The formatting conventions of the current culture are automatically used when you format a date by using any of these methods:

- The parameterless `DateTime.ToString()` method
- The `DateTime.ToString(String)` method, which includes a format string
- The parameterless `DateTimeOffset.ToString()` method
- The `DateTimeOffset.ToString(String)`, which includes a format string
- The [composite formatting](#) feature, when it is used with dates

The following example displays sunrise and sunset data twice for October 11, 2012. It first sets the current culture to Croatian (Croatia), and then to English (Great Britain). In each case, the dates and times are displayed in the format that is appropriate for that culture.

```
using System;
using System.Globalization;
using System.Threading;

public class Example3
{
    static DateTime[] dates = { new DateTime(2012, 10, 11, 7, 06, 0),
                               new DateTime(2012, 10, 11, 18, 19, 0) };

    public static void Main3()
    {
        Thread.CurrentThread.CurrentCulture = CultureInfo.CreateSpecificCulture("hr-HR");
        ShowDayInfo();
        Console.WriteLine();
        Thread.CurrentThread.CurrentCulture = CultureInfo.CreateSpecificCulture("en-GB");
        ShowDayInfo();
    }

    private static void ShowDayInfo()
    {
        Console.WriteLine("Date: {0:D}", dates[0]);
        Console.WriteLine("Sunrise: {0:T}", dates[0]);
        Console.WriteLine("Sunset: {0:T}", dates[1]);
    }
}

// The example displays the following output:
//      Date: 11. listopada 2012.
//      Sunrise: 7:06:00
//      Sunset: 18:19:00
//
//      Date: 11 October 2012
//      Sunrise: 07:06:00
//      Sunset: 18:19:00
```

```

Imports System.Globalization
Imports System.Threading

Module Example3
    Dim dates() As Date = {New Date(2012, 10, 11, 7, 6, 0),
                           New Date(2012, 10, 11, 18, 19, 0)}

    Public Sub Main3()
        Thread.CurrentThread.CurrentCulture = CultureInfo.CreateSpecificCulture("hr-HR")
        ShowDayInfo()
        Console.WriteLine()
        Thread.CurrentThread.CurrentCulture = CultureInfo.CreateSpecificCulture("en-GB")
        ShowDayInfo()
    End Sub

    Private Sub ShowDayInfo()
        Console.WriteLine("Date: {0:D}", dates(0))
        Console.WriteLine("    Sunrise: {0:T}", dates(0))
        Console.WriteLine("    Sunset: {0:T}", dates(1))
    End Sub
End Module
' The example displays the following output:
'     Date: 11. listopada 2012.
'         Sunrise: 7:06:00
'         Sunset: 18:19:00
'
'     Date: 11 October 2012
'         Sunrise: 07:06:00
'         Sunset: 18:19:00

```

Persist dates and times

You should never persist date and time data in a format that can vary by culture. This is a common programming error that results in either corrupted data or a run-time exception. The following example serializes two dates, January 9, 2013 and August 18, 2013, as strings by using the formatting conventions of the English (United States) culture. When the data is retrieved and parsed by using the conventions of the English (United States) culture, it is successfully restored. However, when it is retrieved and parsed by using the conventions of the English (United Kingdom) culture, the first date is wrongly interpreted as September 1, and the second fails to parse because the Gregorian calendar does not have an eighteenth month.

```

using System;
using System.IO;
using System.Globalization;
using System.Threading;

public class Example4
{
    public static void Main4()
    {
        // Persist two dates as strings.
        Thread.CurrentThread.CurrentCulture = CultureInfo.CreateSpecificCulture("en-US");
        DateTime[] dates = { new DateTime(2013, 1, 9),
                             new DateTime(2013, 8, 18) };
        StreamWriter sw = new StreamWriter("dateData.dat");
        sw.WriteLine("{0:d}|{1:d}", dates[0], dates[1]);
        sw.Close();

        // Read the persisted data.
        StreamReader sr = new StreamReader("dateData.dat");
        string dateData = sr.ReadToEnd();
        sr.Close();
        string[] dateStrings = dateData.Split('|');

        // Restore and display the data using the conventions of the en-US culture.
        Console.WriteLine("Current Culture: {0}",
                          Thread.CurrentThread.CurrentCulture.DisplayName);
        foreach (var dateStr in dateStrings) {
            DateTime restoredDate;
            if (DateTime.TryParse(dateStr, out restoredDate))
                Console.WriteLine("The date is {0:D}", restoredDate);
            else
                Console.WriteLine("ERROR: Unable to parse {0}", dateStr);
        }
        Console.WriteLine();

        // Restore and display the data using the conventions of the en-GB culture.
        Thread.CurrentThread.CurrentCulture = CultureInfo.CreateSpecificCulture("en-GB");
        Console.WriteLine("Current Culture: {0}",
                          Thread.CurrentThread.CurrentCulture.DisplayName);
        foreach (var dateStr in dateStrings) {
            DateTime restoredDate;
            if (DateTime.TryParse(dateStr, out restoredDate))
                Console.WriteLine("The date is {0:D}", restoredDate);
            else
                Console.WriteLine("ERROR: Unable to parse {0}", dateStr);
        }
    }
}

// The example displays the following output:
//      Current Culture: English (United States)
//      The date is Wednesday, January 09, 2013
//      The date is Sunday, August 18, 2013
//
//      Current Culture: English (United Kingdom)
//      The date is 01 September 2013
//      ERROR: Unable to parse 8/18/2013

```

```

Imports System.Globalization
Imports System.IO
Imports System.Threading

Module Example4
    Public Sub Main4()
        ' Persist two dates as strings.
        Thread.CurrentThread.CurrentCulture = CultureInfo.CreateSpecificCulture("en-US")
        Dim dates() As DateTime = {New DateTime(2013, 1, 9),
                                   New DateTime(2013, 8, 18)}
        Dim sw As New StreamWriter("dateData.dat")
        sw.WriteLine("{0:d}|{1:d}", dates(0), dates(1))
        sw.Close()

        ' Read the persisted data.
        Dim sr As New StreamReader("dateData.dat")
        Dim dateData As String = sr.ReadToEnd()
        sr.Close()
        Dim dateStrings() As String = dateData.Split("|"c)

        ' Restore and display the data using the conventions of the en-US culture.
        Console.WriteLine("Current Culture: {0}",
                          Thread.CurrentThread.CurrentCulture.DisplayName)
        For Each dateStr In dateStrings
            Dim restoredDate As Date
            If Date.TryParse(dateStr, restoredDate) Then
                Console.WriteLine("The date is {0:D}", restoredDate)
            Else
                Console.WriteLine("ERROR: Unable to parse {0}", dateStr)
            End If
        Next
        Console.WriteLine()

        ' Restore and display the data using the conventions of the en-GB culture.
        Thread.CurrentThread.CurrentCulture = CultureInfo.CreateSpecificCulture("en-GB")
        Console.WriteLine("Current Culture: {0}",
                          Thread.CurrentThread.CurrentCulture.DisplayName)
        For Each dateStr In dateStrings
            Dim restoredDate As Date
            If Date.TryParse(dateStr, restoredDate) Then
                Console.WriteLine("The date is {0:D}", restoredDate)
            Else
                Console.WriteLine("ERROR: Unable to parse {0}", dateStr)
            End If
        Next
    End Sub
End Module
' The example displays the following output:
'   Current Culture: English (United States)
'   The date is Wednesday, January 09, 2013
'   The date is Sunday, August 18, 2013
'
'   Current Culture: English (United Kingdom)
'   The date is 01 September 2013
'   ERROR: Unable to parse 8/18/2013

```

You can avoid this problem in any of three ways:

- Serialize the date and time in binary format rather than as a string.
- Save and parse the string representation of the date and time by using a custom format string that is the same regardless of the user's culture.
- Save the string by using the formatting conventions of the invariant culture.

The following example illustrates the last approach. It uses the formatting conventions of the invariant culture

returned by the static [CultureInfo.InvariantCulture](#) property.

```
using System;
using System.IO;
using System.Globalization;
using System.Threading;

public class Example5
{
    public static void Main5()
    {
        // Persist two dates as strings.
        Thread.CurrentThread.CurrentCulture = CultureInfo.CreateSpecificCulture("en-US");
        DateTime[] dates = { new DateTime(2013, 1, 9),
                            new DateTime(2013, 8, 18) };
        StreamWriter sw = new StreamWriter("dateData.dat");
        sw.WriteLine(String.Format(CultureInfo.InvariantCulture,
                                  "{0:d}|{1:d}", dates[0], dates[1]));
        sw.Close();

        // Read the persisted data.
        StreamReader sr = new StreamReader("dateData.dat");
        string dateData = sr.ReadToEnd();
        sr.Close();
        string[] dateStrings = dateData.Split('|');

        // Restore and display the data using the conventions of the en-US culture.
        Console.WriteLine("Current Culture: {0}",
                          Thread.CurrentThread.CurrentCulture.DisplayName);
        foreach (var dateStr in dateStrings) {
            DateTime restoredDate;
            if (DateTime.TryParse(dateStr, CultureInfo.InvariantCulture,
                                 DateTimeStyles.None, out restoredDate))
                Console.WriteLine("The date is {0:D}", restoredDate);
            else
                Console.WriteLine("ERROR: Unable to parse {0}", dateStr);
        }
        Console.WriteLine();

        // Restore and display the data using the conventions of the en-GB culture.
        Thread.CurrentThread.CurrentCulture = CultureInfo.CreateSpecificCulture("en-GB");
        Console.WriteLine("Current Culture: {0}",
                          Thread.CurrentThread.CurrentCulture.DisplayName);
        foreach (var dateStr in dateStrings) {
            DateTime restoredDate;
            if (DateTime.TryParse(dateStr, CultureInfo.InvariantCulture,
                                 DateTimeStyles.None, out restoredDate))
                Console.WriteLine("The date is {0:D}", restoredDate);
            else
                Console.WriteLine("ERROR: Unable to parse {0}", dateStr);
        }
    }
}

// The example displays the following output:
//      Current Culture: English (United States)
//      The date is Wednesday, January 09, 2013
//      The date is Sunday, August 18, 2013
//
//      Current Culture: English (United Kingdom)
//      The date is 09 January 2013
//      The date is 18 August 2013
```

```

Imports System.Globalization
Imports System.IO
Imports System.Threading

Module Example5
    Public Sub Main5()
        ' Persist two dates as strings.
        Thread.CurrentThread.CurrentCulture = CultureInfo.CreateSpecificCulture("en-US")
        Dim dates() As DateTime = {New DateTime(2013, 1, 9),
                                   New DateTime(2013, 8, 18)}
        Dim sw As New StreamWriter("dateData.dat")
        sw.WriteLine(String.Format(CultureInfo.InvariantCulture,
                                  "{0:d}|{1:d}", dates(0), dates(1)))
        sw.Close()

        ' Read the persisted data.
        Dim sr As New StreamReader("dateData.dat")
        Dim dateData As String = sr.ReadToEnd()
        sr.Close()
        Dim dateStrings() As String = dateData.Split("|c")

        ' Restore and display the data using the conventions of the en-US culture.
        Console.WriteLine("Current Culture: {0}",
                          Thread.CurrentThread.CurrentCulture.DisplayName)
        For Each dateStr In dateStrings
            Dim restoredDate As Date
            If Date.TryParse(dateStr, CultureInfo.InvariantCulture,
                            DateTimeStyles.None, restoredDate) Then
                Console.WriteLine("The date is {0:D}", restoredDate)
            Else
                Console.WriteLine("ERROR: Unable to parse {0}", dateStr)
            End If
        Next
        Console.WriteLine()

        ' Restore and display the data using the conventions of the en-GB culture.
        Thread.CurrentThread.CurrentCulture = CultureInfo.CreateSpecificCulture("en-GB")
        Console.WriteLine("Current Culture: {0}",
                          Thread.CurrentThread.CurrentCulture.DisplayName)
        For Each dateStr In dateStrings
            Dim restoredDate As Date
            If Date.TryParse(dateStr, CultureInfo.InvariantCulture,
                            DateTimeStyles.None, restoredDate) Then
                Console.WriteLine("The date is {0:D}", restoredDate)
            Else
                Console.WriteLine("ERROR: Unable to parse {0}", dateStr)
            End If
        Next
    End Sub
End Module
' The example displays the following output:
'     Current Culture: English (United States)
'     The date is Wednesday, January 09, 2013
'     The date is Sunday, August 18, 2013
'
'     Current Culture: English (United Kingdom)
'     The date is 09 January 2013
'     The date is 18 August 2013

```

Serialization and time zone awareness

A date and time value can have multiple interpretations, ranging from a general time ("The stores open on January 2, 2013, at 9:00 A.M.") to a specific moment in time ("Date of birth: January 2, 2013 6:32:00 A.M."). When a time value represents a specific moment in time and you restore it from a serialized value, you should ensure that it represents the same moment in time regardless of the user's geographical location or time zone.

The following example illustrates this problem. It saves a single local date and time value as a string in three **standard formats** ("G" for general date long time, "s" for sortable date/time, and "o" for round-trip date/time) as well as in binary format.

```
using System;
using System.IO;
using System.Runtime.Serialization.Formatters.Binary;

public class Example6
{
    public static void Main6()
    {
        BinaryFormatter formatter = new BinaryFormatter();

        DateTime dateOriginal = new DateTime(2013, 3, 30, 18, 0, 0);
        dateOriginal = DateTime.SpecifyKind(dateOriginal, DateTimeKind.Local);

        // Serialize a date.
        if (! File.Exists("DateInfo.dat")) {
            StreamWriter sw = new StreamWriter("DateInfo.dat");
            sw.Write("{0:G}|{0:s}|{0:o}", dateOriginal);
            sw.Close();
            Console.WriteLine("Serialized dates to DateInfo.dat");
        }

        // Serialize the data as a binary value.
        if (! File.Exists("DateInfo.bin")) {
            FileStream fsIn = new FileStream("DateInfo.bin", FileMode.Create);
            formatter.Serialize(fsIn, dateOriginal);
            fsIn.Close();
            Console.WriteLine("Serialized date to DateInfo.bin");
        }

        Console.WriteLine();

        // Restore the date from string values.
        StreamReader sr = new StreamReader("DateInfo.dat");
        string datesToSplit = sr.ReadToEnd();
        string[] dateStrings = datesToSplit.Split('|');
        foreach (var dateStr in dateStrings) {
            DateTime newDate = DateTime.Parse(dateStr);
            Console.WriteLine("'{0}' --> {1} {2}",
                dateStr, newDate, newDate.Kind);
        }

        Console.WriteLine();

        // Restore the date from binary data.
        FileStream fsOut = new FileStream("DateInfo.bin", FileMode.Open);
        DateTime restoredDate = (DateTime) formatter.Deserialize(fsOut);
        Console.WriteLine("{0} {1}", restoredDate, restoredDate.Kind);
    }
}
```

```

Imports System.IO
Imports System.Runtime.Serialization.Formatters.Binary

Module Example6
    Public Sub Main6()
        Dim formatter As New BinaryFormatter()

        ' Serialize a date.
        Dim dateOriginal As Date = #03/30/2013 6:00PM#
        dateOriginal = DateTime.SpecifyKind(dateOriginal, DateTimeKind.Local)
        ' Serialize the date in string form.
        If Not File.Exists("DateInfo.dat") Then
            Dim sw As New StreamWriter("DateInfo.dat")
            sw.WriteLine("{0:G}|{0:s}|{0:o}", dateOriginal)
            sw.Close()
            Console.WriteLine("Serialized dates to DateInfo.dat")
        End If
        ' Serialize the date as a binary value.
        If Not File.Exists("DateInfo.bin") Then
            Dim fsIn As New FileStream("DateInfo.bin", FileMode.Create)
            formatter.Serialize(fsIn, dateOriginal)
            fsIn.Close()
            Console.WriteLine("Serialized date to DateInfo.bin")
        End If
        Console.WriteLine()

        ' Restore the date from string values.
        Dim sr As New StreamReader("DateInfo.dat")
        Dim datesToSplit As String = sr.ReadToEnd()
        Dim dateStrings() As String = datesToSplit.Split("|"c)
        For Each dateStr In dateStrings
            Dim newDate As DateTime = DateTime.Parse(dateStr)
            Console.WriteLine("'"{0}' --> {1} {2}",
                dateStr, newDate, newDate.Kind)
        Next
        Console.WriteLine()

        ' Restore the date from binary data.
        Dim fsOut As New FileStream("DateInfo.bin", FileMode.Open)
        Dim restoredDate As Date = DirectCast(formatter.Deserialize(fsOut), DateTime)
        Console.WriteLine("{0} {1}", restoredDate, restoredDate.Kind)
    End Sub
End Module

```

When the data is restored on a system in the same time zone as the system on which it was serialized, the deserialized date and time values accurately reflect the original value, as the output shows:

```

'3/30/2013 6:00:00 PM' --> 3/30/2013 6:00:00 PM Unspecified
'2013-03-30T18:00:00' --> 3/30/2013 6:00:00 PM Unspecified
'2013-03-30T18:00:00.0000000-07:00' --> 3/30/2013 6:00:00 PM Local

3/30/2013 6:00:00 PM Local

```

However, if you restore the data on a system in a different time zone, only the date and time value that was formatted with the "o" (round-trip) standard format string preserves time zone information and therefore represents the same instant in time. Here's the output when the date and time data is restored on a system in the Romance Standard Time zone:

```
'3/30/2013 6:00:00 PM' --> 3/30/2013 6:00:00 PM Unspecified  
'2013-03-30T18:00:00' --> 3/30/2013 6:00:00 PM Unspecified  
'2013-03-30T18:00:00.000000-07:00' --> 3/31/2013 3:00:00 AM Local  
  
3/30/2013 6:00:00 PM Local
```

To accurately reflect a date and time value that represents a single moment of time regardless of the time zone of the system on which the data is deserialized, you can do any of the following:

- Save the value as a string by using the "o" (round-trip) standard format string. Then deserialize it on the target system.
- Convert it to UTC and save it as a string by using the "r" (RFC1123) standard format string. Then deserialize it on the target system and convert it to local time.
- Convert it to UTC and save it as a string by using the "u" (universal sortable) standard format string. Then deserialize it on the target system and convert it to local time.
- Convert it to UTC and save it in binary format. Then deserialize it on the target system and convert it to local time.

The following example illustrates each technique.

```

using System;
using System.IO;
using System.Runtime.Serialization.Formatters.Binary;

public class Example9
{
    public static void Main9()
    {
        BinaryFormatter formatter = new BinaryFormatter();

        // Serialize a date.
        DateTime dateOriginal = new DateTime(2013, 3, 30, 18, 0, 0);
        dateOriginal = DateTime.SpecifyKind(dateOriginal, DateTimeKind.Local);

        // Serialize the date in string form.
        if (! File.Exists("DateInfo2.dat")) {
            StreamWriter sw = new StreamWriter("DateInfo2.dat");
            sw.Write("{0:o}|{1:r}|{1:u}", dateOriginal,
                    dateOriginal.ToUniversalTime());
            sw.Close();
            Console.WriteLine("Serialized dates to DateInfo.dat");
        }

        // Serialize the date as a binary value.
        if (! File.Exists("DateInfo2.bin")) {
            FileStream fsIn = new FileStream("DateInfo2.bin", FileMode.Create);
            formatter.Serialize(fsIn, dateOriginal.ToUniversalTime());
            fsIn.Close();
            Console.WriteLine("Serialized date to DateInfo.bin");
        }

        Console.WriteLine();

        // Restore the date from string values.
        StreamReader sr = new StreamReader("DateInfo2.dat");
        string datesToSplit = sr.ReadToEnd();
        string[] dateStrings = datesToSplit.Split('|');
        for (int ctr = 0; ctr < dateStrings.Length; ctr++) {
            DateTime newDate = DateTime.Parse(dateStrings[ctr]);
            if (ctr == 1) {
                Console.WriteLine("{0} --> {1} {2}",
                    dateStrings[ctr], newDate, newDate.Kind);
            }
            else {
                DateTime newLocalDate = newDateToLocalTime();
                Console.WriteLine("{0} --> {1} {2}",
                    dateStrings[ctr], newLocalDate, newLocalDate.Kind);
            }
        }

        Console.WriteLine();

        // Restore the date from binary data.
        FileStream fsOut = new FileStream("DateInfo2.bin", FileMode.Open);
        DateTime restoredDate = (DateTime) formatter.Deserialize(fsOut);
        restoredDate = restoredDateToLocalTime();
        Console.WriteLine("{0} {1}", restoredDate, restoredDate.Kind);
    }
}

```

```

Imports System.IO
Imports System.Runtime.Serialization.Formatters.Binary

Module Example9
    Public Sub Main9()
        Dim formatter As New BinaryFormatter()

        ' Serialize a date.
        Dim dateOriginal As Date = #03/30/2013 6:00PM#
        dateOriginal = DateTime.SpecifyKind(dateOriginal, DateTimeKind.Local)

        ' Serialize the date in string form.
        If Not File.Exists("DateInfo2.dat") Then
            Dim sw As New StreamWriter("DateInfo2.dat")
            sw.Write("{0:o}|{1:r}|{1:u}", dateOriginal,
                    dateOriginal.ToUniversalTime())
            sw.Close()
            Console.WriteLine("Serialized dates to DateInfo.dat")
        End If

        ' Serialize the date as a binary value.
        If Not File.Exists("DateInfo2.bin") Then
            Dim fsIn As New FileStream("DateInfo2.bin", FileMode.Create)
            formatter.Serialize(fsIn, dateOriginal.ToUniversalTime())
            fsIn.Close()
            Console.WriteLine("Serialized date to DateInfo.bin")
        End If
        Console.WriteLine()

        ' Restore the date from string values.
        Dim sr As New StreamReader("DateInfo2.dat")
        Dim datesToSplit As String = sr.ReadToEnd()
        Dim dateStrings() As String = datesToSplit.Split("|"c)
        For ctr As Integer = 0 To dateStrings.Length - 1
            Dim newDate As DateTime = DateTime.Parse(dateStrings(ctr))
            If ctr = 1 Then
                Console.WriteLine("'{0}' --> {1} {2}",
                    dateStrings(ctr), newDate, newDate.Kind)
            Else
                Dim newLocalDate As DateTime = newDateToLocalTime()
                Console.WriteLine("'{0}' --> {1} {2}",
                    dateStrings(ctr), newLocalDate, newLocalDate.Kind)
            End If
        Next
        Console.WriteLine()

        ' Restore the date from binary data.
        Dim fsOut As New FileStream("DateInfo2.bin", FileMode.Open)
        Dim restoredDate As Date = DirectCast(formatter.Deserialize(fsOut), DateTime)
        restoredDate = restoredDateToLocalTime()
        Console.WriteLine("{0} {1}", restoredDate, restoredDate.Kind)
    End Sub
End Module

```

When the data is serialized on a system in the Pacific Standard Time zone and deserialized on a system in the Romance Standard Time zone, the example displays the following output:

```

'2013-03-30T18:00:00.000000-07:00' --> 3/31/2013 3:00:00 AM Local
'Sun, 31 Mar 2013 01:00:00 GMT' --> 3/31/2013 3:00:00 AM Local
'2013-03-31 01:00:00Z' --> 3/31/2013 3:00:00 AM Local

3/31/2013 3:00:00 AM Local

```

For more information, see [Convert times between time zones](#).

Perform date and time arithmetic

Both the [DateTime](#) and [DateTimeOffset](#) types support arithmetic operations. You can calculate the difference between two date values, or you can add or subtract particular time intervals to or from a date value. However, arithmetic operations on date and time values do not take time zones and time zone adjustment rules into account. Because of this, date and time arithmetic on values that represent moments in time can return inaccurate results.

For example, the transition from Pacific Standard Time to Pacific Daylight Time occurs on the second Sunday of March, which is March 10 for the year 2013. As the following example shows, if you calculate the date and time that is 48 hours after March 9, 2013 at 10:30 A.M. on a system in the Pacific Standard Time zone, the result, March 11, 2013 at 10:30 A.M., does not take the intervening time adjustment into account.

```
using System;

public class Example7
{
    public static void Main7()
    {
        DateTime date1 = DateTime.SpecifyKind(new DateTime(2013, 3, 9, 10, 30, 0),
                                              DateTimeKind.Local);
        TimeSpan interval = new TimeSpan(48, 0, 0);
        DateTime date2 = date1 + interval;
        Console.WriteLine("{0:g} + {1:N1} hours = {2:g}",
                          date1, interval.TotalHours, date2);
    }
}
// The example displays the following output:
//      3/9/2013 10:30 AM + 48.0 hours = 3/11/2013 10:30 AM
```

```
Module Example7
    Public Sub Main7()
        Dim date1 As Date = DateTime.SpecifyKind(#3/9/2013 10:30AM|,
                                              DateTimeKind.Local)
        Dim interval As New TimeSpan(48, 0, 0)
        Dim date2 As Date = date1 + interval
        Console.WriteLine("{0:g} + {1:N1} hours = {2:g}",
                          date1, interval.TotalHours, date2)
    End Sub
End Module
' The example displays the following output:
'      3/9/2013 10:30 AM + 48.0 hours = 3/11/2013 10:30 AM
```

To ensure that an arithmetic operation on date and time values produces accurate results, follow these steps:

1. Convert the time in the source time zone to UTC.
2. Perform the arithmetic operation.
3. If the result is a date and time value, convert it from UTC to the time in the source time zone.

The following example is similar to the previous example, except that it follows these three steps to correctly add 48 hours to March 9, 2013 at 10:30 A.M.

```

using System;

public class Example8
{
    public static void Main8()
    {
        TimeZoneInfo pst = TimeZoneInfo.FindSystemTimeZoneById("Pacific Standard Time");
        DateTime date1 = DateTime.SpecifyKind(new DateTime(2013, 3, 9, 10, 30, 0),
                                              DateTimeKind.Local);
        DateTime utc1 = date1.ToUniversalTime();
        TimeSpan interval = new TimeSpan(48, 0, 0);
        DateTime utc2 = utc1 + interval;
        DateTime date2 = TimeZoneInfo.ConvertTimeFromUtc(utc2, pst);
        Console.WriteLine("{0:g} + {1:N1} hours = {2:g}",
                          date1, interval.TotalHours, date2);
    }
}

// The example displays the following output:
//      3/9/2013 10:30 AM + 48.0 hours = 3/11/2013 11:30 AM

```

```

Module Example8
    Public Sub Main8()
        Dim pst As TimeZoneInfo = TimeZoneInfo.FindSystemTimeZoneById("Pacific Standard Time")
        Dim date1 As Date = DateTime.SpecifyKind(#3/9/2013 10:30AM|,
                                              DateTimeKind.Local)
        Dim utc1 As Date = date1.ToUniversalTime()
        Dim interval As New TimeSpan(48, 0, 0)
        Dim utc2 As Date = utc1 + interval
        Dim date2 As Date = TimeZoneInfo.ConvertTimeFromUtc(utc2, pst)
        Console.WriteLine("{0:g} + {1:N1} hours = {2:g}",
                          date1, interval.TotalHours, date2)
    End Sub
End Module
' The example displays the following output:
'      3/9/2013 10:30 AM + 48.0 hours = 3/11/2013 11:30 AM

```

For more information, see [Perform arithmetic operations with dates and times](#).

Use culture-sensitive names for date elements

Your app may need to display the name of the month or the day of the week. To do this, code such as the following is common.

```

using System;

public class Example12
{
    public static void Main12()
    {
        DateTime midYear = new DateTime(2013, 7, 1);
        Console.WriteLine("{0:d} is a {1}.", midYear, GetDayName(midYear));
    }

    private static string GetDayName(DateTime date)
    {
        return date.DayOfWeek.ToString("G");
    }
}

// The example displays the following output:
//      7/1/2013 is a Monday.

```

```
Module Example12
    Public Sub Main12()
        Dim midYear As Date = #07/01/2013#
        Console.WriteLine("{0:d} is a {1}.", midYear, GetDayName(midYear))
    End Sub

    Private Function GetDayName(dat As Date) As String
        Return dat.DayOfWeek.ToString("G")
    End Function
End Module
' The example displays the following output:
' 7/1/2013 is a Monday.
```

However, this code always returns the names of the days of the week in English. Code that extracts the name of the month is often even more inflexible. It frequently assumes a twelve-month calendar with names of months in a specific language.

By using [custom date and time format strings](#) or the properties of the [DateTimeFormatInfo](#) object, it is easy to extract strings that reflect the names of days of the week or months in the user's culture, as the following example illustrates. It changes the current culture to French (France) and displays the name of the day of the week and the name of the month for July 1, 2013.

```

using System;
using System.Globalization;
using System.Threading;

public class Example13
{
    public static void Main13()
    {
        // Set the current culture to French (France).
        CultureInfo.CurrentCulture = CultureInfo.CreateSpecificCulture("fr-FR");

        DateTime midYear = new DateTime(2013, 7, 1);
        Console.WriteLine("{0:d} is a {1}.", midYear, DateUtilities.GetDayName(midYear));
        Console.WriteLine("{0:d} is a {1}.", midYear, DateUtilities.GetDayName((int)midYear.DayOfWeek));
        Console.WriteLine("{0:d} is in {1}.", midYear, DateUtilities.GetMonthName(midYear));
        Console.WriteLine("{0:d} is in {1}.", midYear, DateUtilities.GetMonthName(midYear.Month));
    }
}

public static class DateUtilities
{
    public static string GetDayName(int dayOfWeek)
    {
        if (dayOfWeek < 0 | dayOfWeek > DateTimeFormatInfo.CurrentInfo.DayNames.Length)
            return String.Empty;
        else
            return DateTimeFormatInfo.CurrentInfo.DayNames[dayOfWeek];
    }

    public static string GetDayName(DateTime date)
    {
        return date.ToString("dddd");
    }

    public static string GetMonthName(int month)
    {
        if (month < 1 | month > DateTimeFormatInfo.CurrentInfo.MonthNames.Length - 1)
            return String.Empty;
        else
            return DateTimeFormatInfo.CurrentInfo.MonthNames[month - 1];
    }

    public static string GetMonthName(DateTime date)
    {
        return date.ToString("MMMM");
    }
}

// The example displays the following output:
//      01/07/2013 is a lundi.
//      01/07/2013 is a lundi.
//      01/07/2013 is in juillet.
//      01/07/2013 is in juillet.

```

```

Imports System.Globalization
Imports System.Threading

Module Example13
    Public Sub Main13()
        ' Set the current culture to French (France).
        CultureInfo.CurrentCulture = CultureInfo.CreateSpecificCulture("fr-FR")

        Dim midYear As Date = #07/01/2013#
        Console.WriteLine("{0:d} is a {1}.", midYear, DateUtilities.GetDayName(midYear))
        Console.WriteLine("{0:d} is a {1}.", midYear, DateUtilities.GetDayName(midYear.DayOfWeek))
        Console.WriteLine("{0:d} is in {1}.", midYear, DateUtilities.GetMonthName(midYear))
        Console.WriteLine("{0:d} is in {1}.", midYear, DateUtilities.GetMonthName(midYear.Month))

    End Sub
End Module

Public Class DateUtilities
    Public Shared Function GetDayName(dayOfWeek As Integer) As String
        If dayOfWeek < 0 Or dayOfWeek > DateTimeFormatInfo.CurrentInfo.DayNames.Length Then
            Return String.Empty
        Else
            Return DateTimeFormatInfo.CurrentInfo.DayNames(dayOfWeek)
        End If
    End Function

    Public Shared Function GetDayName(dat As Date) As String
        Return dat.ToString("dddd")
    End Function

    Public Shared Function GetMonthName(month As Integer) As String
        If month < 1 Or month > DateTimeFormatInfo.CurrentInfo.MonthNames.Length - 1 Then
            Return String.Empty
        Else
            Return DateTimeFormatInfo.CurrentInfo.MonthNames(month - 1)
        End If
    End Function

    Public Shared Function GetMonthName(dat As Date) As String
        Return dat.ToString("MMMM")
    End Function
End Class
' The example displays the following output:
'   01/07/2013 is a lundi.
'   01/07/2013 is a lundi.
'   01/07/2013 is in juillet.
'   01/07/2013 is in juillet.

```

Numeric values

The handling of numbers depends on whether they are displayed in the user interface or persisted. This section examines both usages.

NOTE

In parsing and formatting operations, .NET recognizes only the Basic Latin characters 0 through 9 (U+0030 through U+0039) as numeric digits.

Display numeric values

Typically, when numbers are displayed in the user interface, you should use the formatting conventions of the user's culture, which is defined by the [CultureInfo.CurrentCulture](#) property and by the [NumberFormatInfo](#) object returned by the [CultureInfo.CurrentCulture.NumberFormat](#) property. The formatting conventions of the current

culture are automatically used when you format a date by using any of the following methods:

- The parameterless `ToString` method of any numeric type
- The `ToString(String)` method of any numeric type, which includes a format string as an argument
- The [composite formatting](#) feature, when it is used with numeric values

The following example displays the average temperature per month in Paris, France. It first sets the current culture to French (France) before displaying the data, and then sets it to English (United States). In each case, the month names and temperatures are displayed in the format that is appropriate for that culture. Note that the two cultures use different decimal separators in the temperature value. Also note that the example uses the "MMMM" custom date and time format string to display the full month name, and that it allocates the appropriate amount of space for the month name in the result string by determining the length of the longest month name in the `DateTimeFormatInfo.MonthNames` array.

```
using System;
using System.Globalization;
using System.Threading;

public class Example14
{
    public static void Main14()
    {
        DateTime dateForMonth = new DateTime(2013, 1, 1);
        double[] temperatures = { 3.4, 3.5, 7.6, 10.4, 14.5, 17.2,
                                  19.9, 18.2, 15.9, 11.3, 6.9, 5.3 };

        Thread.CurrentThread.CurrentCulture = CultureInfo.CreateSpecificCulture("fr-FR");
        Console.WriteLine("Current Culture: {0}", CultureInfo.CurrentCulture.DisplayName);
        // Build the format string dynamically so we allocate enough space for the month name.
        string fmtString = "{0,-" + GetLongestMonthNameLength().ToString() + ":MMMM} {1,4}";
        for (int ctr = 0; ctr < temperatures.Length; ctr++)
            Console.WriteLine(fmtString,
                              dateForMonth.AddMonths(ctr),
                              temperatures[ctr]);

        Console.WriteLine();

        Thread.CurrentThread.CurrentCulture = CultureInfo.CreateSpecificCulture("en-US");
        Console.WriteLine("Current Culture: {0}", CultureInfo.CurrentCulture.DisplayName);
        fmtString = "{0,-" + GetLongestMonthNameLength().ToString() + ":MMMM} {1,4}";
        for (int ctr = 0; ctr < temperatures.Length; ctr++)
            Console.WriteLine(fmtString,
                              dateForMonth.AddMonths(ctr),
                              temperatures[ctr]);
    }

    private static int GetLongestMonthNameLength()
    {
        int length = 0;
        foreach (var nameOfMonth in DateTimeFormatInfo.CurrentInfo.MonthNames)
            if (nameOfMonth.Length > length) length = nameOfMonth.Length;

        return length;
    }
}

// The example displays the following output:
// Current Culture: French (France)
//      janvier      3,4
//      février      3,5
//      mars          7,6
//      avril         10,4
//      mai           14,5
//      juin          17,2
//      juillet      19,9
```

```
//                ٢٧,٣
//        août      18,2
//        septembre 15,9
//        octobre    11,3
//        novembre   6,9
//        décembre   5,3
//
//        Current Culture: English (United States)
//        January     3.4
//        February   3.5
//        March      7.6
//        April      10.4
//        May       14.5
//        June      17.2
//        July      19.9
//        August    18.2
//        September 15.9
//        October   11.3
//        November  6.9
//        December  5.3
```

```

Imports System.Globalization
Imports System.Threading

Module Example14
    Public Sub Main14()
        Dim dateForMonth As Date = #1/1/2013#
        Dim temperatures() As Double = {3.4, 3.5, 7.6, 10.4, 14.5, 17.2,
                                         19.9, 18.2, 15.9, 11.3, 6.9, 5.3}

        Thread.CurrentThread.CurrentCulture = CultureInfo.CreateSpecificCulture("fr-FR")
        Console.WriteLine("Current Culture: {0}", CultureInfo.CurrentCulture.DisplayName)
        Dim fmtString As String = "{0,-" + GetLongestMonthNameLength().ToString() + ":MMMM}" {1,4}
        For ctr = 0 To temperatures.Length - 1
            Console.WriteLine(fmtString,
                               dateForMonth.AddMonths(ctr),
                               temperatures(ctr))
        Next
        Console.WriteLine()

        Thread.CurrentThread.CurrentCulture = CultureInfo.CreateSpecificCulture("en-US")
        Console.WriteLine("Current Culture: {0}", CultureInfo.CurrentCulture.DisplayName)
        ' Build the format string dynamically so we allocate enough space for the month name.
        fmtString = "{0,-" + GetLongestMonthNameLength().ToString() + ":MMMM}" {1,4}
        For ctr = 0 To temperatures.Length - 1
            Console.WriteLine(fmtString,
                               dateForMonth.AddMonths(ctr),
                               temperatures(ctr))
        Next
    End Sub

    Private Function GetLongestMonthNameLength() As Integer
        Dim length As Integer
        For Each nameOfMonth In DateTimeFormatInfo.CurrentInfo.MonthNames
            If nameOfMonth.Length > length Then length = nameOfMonth.Length
        Next
        Return length
    End Function
End Module

' The example displays the following output:
' Current Culture: French (France)
' janvier      3,4
' février      3,5
' mars         7,6
' avril         10,4
' mai          14,5
' juin          17,2
' juillet       19,9
' août          18,2
' septembre     15,9
' octobre       11,3
' novembre      6,9
' décembre      5,3
'

' Current Culture: English (United States)
' January      3.4
' February     3.5
' March        7.6
' April         10.4
' May          14.5
' June          17.2
' July          19.9
' August        18.2
' September     15.9
' October       11.3
' November      6.9
' December      5.3

```

Persist numeric values

You should never persist numeric data in a culture-specific format. This is a common programming error that results in either corrupted data or a run-time exception. The following example generates ten random floating-point numbers, and then serializes them as strings by using the formatting conventions of the English (United States) culture. When the data is retrieved and parsed by using the conventions of the English (United States) culture, it is successfully restored. However, when it is retrieved and parsed by using the conventions of the French (France) culture, none of the numbers can be parsed because the cultures use different decimal separators.

```
using System;
using System.Globalization;
using System.IO;
using System.Threading;

public class Example15
{
    public static void Main15()
    {
        // Create ten random doubles.
        Thread.CurrentThread.CurrentCulture = CultureInfo.CreateSpecificCulture("en-US");
        double[] numbers = GetRandomNumbers(10);
        DisplayRandomNumbers(numbers);

        // Persist the numbers as strings.
        StreamWriter sw = new StreamWriter("randoms.dat");
        for (int ctr = 0; ctr < numbers.Length; ctr++)
            sw.Write("{0:R}{1}", numbers[ctr], ctr < numbers.Length - 1 ? "|" : "");

        sw.Close();

        // Read the persisted data.
        StreamReader sr = new StreamReader("randoms.dat");
        string numericData = sr.ReadToEnd();
        sr.Close();
        string[] numberStrings = numericData.Split('|');

        // Restore and display the data using the conventions of the en-US culture.
        Console.WriteLine("Current Culture: {0}",
                          Thread.CurrentThread.CurrentCulture.DisplayName);
        foreach (var numberStr in numberStrings) {
            double restoredNumber;
            if (Double.TryParse(numberStr, out restoredNumber))
                Console.WriteLine(restoredNumber.ToString("R"));
            else
                Console.WriteLine("ERROR: Unable to parse '{0}'", numberStr);
        }
        Console.WriteLine();

        // Restore and display the data using the conventions of the fr-FR culture.
        Thread.CurrentThread.CurrentCulture = CultureInfo.CreateSpecificCulture("fr-FR");
        Console.WriteLine("Current Culture: {0}",
                          Thread.CurrentThread.CurrentCulture.DisplayName);
        foreach (var numberStr in numberStrings) {
            double restoredNumber;
            if (Double.TryParse(numberStr, out restoredNumber))
                Console.WriteLine(restoredNumber.ToString("R"));
            else
                Console.WriteLine("ERROR: Unable to parse '{0}'", numberStr);
        }
    }

    private static double[] GetRandomNumbers(int n)
    {
        Random rnd = new Random();
        double[] numbers = new double[n];
        for (int i = 0; i < n; i++)
            numbers[i] = rnd.NextDouble();
        return numbers;
    }
}
```

```

        for (int ctr = 0; ctr < n; ctr++)
            numbers[ctr] = rnd.NextDouble() * 1000;
        return numbers;
    }

    private static void DisplayRandomNumbers(double[] numbers)
    {
        for (int ctr = 0; ctr < numbers.Length; ctr++)
            Console.WriteLine(numbers[ctr].ToString("R"));
        Console.WriteLine();
    }
}

// The example displays output like the following:
//      487.0313743534644
//      674.12000879371533
//      498.72077885024288
//      42.3034229512808
//      970.57311049223563
//      531.33717716268131
//      587.82905693530529
//      562.25210175023039
//      600.7711019370571
//      299.46113717717174
//
//      Current Culture: English (United States)
//      487.0313743534644
//      674.12000879371533
//      498.72077885024288
//      42.3034229512808
//      970.57311049223563
//      531.33717716268131
//      587.82905693530529
//      562.25210175023039
//      600.7711019370571
//      299.46113717717174
//
//      Current Culture: French (France)
//      ERROR: Unable to parse '487.0313743534644'
//      ERROR: Unable to parse '674.12000879371533'
//      ERROR: Unable to parse '498.72077885024288'
//      ERROR: Unable to parse '42.3034229512808'
//      ERROR: Unable to parse '970.57311049223563'
//      ERROR: Unable to parse '531.33717716268131'
//      ERROR: Unable to parse '587.82905693530529'
//      ERROR: Unable to parse '562.25210175023039'
//      ERROR: Unable to parse '600.7711019370571'
//      ERROR: Unable to parse '299.46113717717174'

```

```

Imports System.Globalization
Imports System.IO
Imports System.Threading

Module Example15
    Public Sub Main15()
        ' Create ten random doubles.
        Thread.CurrentThread.CurrentCulture = CultureInfo.CreateSpecificCulture("en-US")
        Dim numbers() As Double = GetRandomNumbers(10)
        DisplayRandomNumbers(numbers)

        ' Persist the numbers as strings.
        Dim sw As New StreamWriter("randoms.dat")
        For ctr As Integer = 0 To numbers.Length - 1
            sw.Write("{0:R}{1}", numbers(ctr), If(ctr < numbers.Length - 1, "|", ""))
        Next
        sw.Close()

        ' Read the persisted data.
        Dim sr As New StreamReader("randoms.dat")

```

```

Dim sr As New StreamReader("randoms.dat")
Dim numericData As String = sr.ReadToEnd()
sr.Close()
Dim numberStrings() As String = numericData.Split("|"c)

' Restore and display the data using the conventions of the en-US culture.
Console.WriteLine("Current Culture: {0}",
                  Thread.CurrentThread.CurrentCulture.DisplayName)
For Each numberStr In numberStrings
    Dim restoredNumber As Double
    If Double.TryParse(numberStr, restoredNumber) Then
        Console.WriteLine(restoredNumber.ToString("R"))
    Else
        Console.WriteLine("ERROR: Unable to parse '{0}'", numberStr)
    End If
Next
Console.WriteLine()

' Restore and display the data using the conventions of the fr-FR culture.
Thread.CurrentThread.CurrentCulture = CultureInfo.CreateSpecificCulture("fr-FR")
Console.WriteLine("Current Culture: {0}",
                  Thread.CurrentThread.CurrentCulture.DisplayName)
For Each numberStr In numberStrings
    Dim restoredNumber As Double
    If Double.TryParse(numberStr, restoredNumber) Then
        Console.WriteLine(restoredNumber.ToString("R"))
    Else
        Console.WriteLine("ERROR: Unable to parse '{0}'", numberStr)
    End If
Next
End Sub

Private Function GetRandomNumbers(n As Integer) As Double()
    Dim rnd As New Random()
    Dim numbers(n - 1) As Double
    For ctr As Integer = 0 To n - 1
        numbers(ctr) = rnd.NextDouble * 1000
    Next
    Return numbers
End Function

Private Sub DisplayRandomNumbers(numbers As Double())
    For ctr As Integer = 0 To numbers.Length - 1
        Console.WriteLine(numbers(ctr).ToString("R"))
    Next
    Console.WriteLine()
End Sub
End Module
' The example displays output like the following:
' 487.0313743534644
' 674.12000879371533
' 498.72077885024288
' 42.3034229512808
' 970.57311049223563
' 531.33717716268131
' 587.82905693530529
' 562.25210175023039
' 600.7711019370571
' 299.46113717717174
'
' Current Culture: English (United States)
487.0313743534644
674.12000879371533
498.72077885024288
42.3034229512808
970.57311049223563
531.33717716268131
587.82905693530529
562.25210175023039
-----
```

```
600.7711019370571
299.46113717717174

Current Culture: French (France)
ERROR: Unable to parse '487.0313743534644'
ERROR: Unable to parse '674.12000879371533'
ERROR: Unable to parse '498.72077885024288'
ERROR: Unable to parse '42.3034229512808'
ERROR: Unable to parse '970.57311049223563'
ERROR: Unable to parse '531.33717716268131'
ERROR: Unable to parse '587.82905693530529'
ERROR: Unable to parse '562.25210175023039'
ERROR: Unable to parse '600.7711019370571'
ERROR: Unable to parse '299.46113717717174'
```

To avoid this problem, you can use one of these techniques:

- Save and parse the string representation of the number by using a custom format string that is the same regardless of the user's culture.
- Save the number as a string by using the formatting conventions of the invariant culture, which is returned by the [CultureInfo.InvariantCulture](#) property.
- Serialize the number in binary instead of string format.

The following example illustrates the last approach. It serializes the array of [Double](#) values, and then deserializes and displays them by using the formatting conventions of the English (United States) and French (France) cultures.

```
using System;
using System.Globalization;
using System.IO;
using System.Runtime.Serialization.Formatters.Binary;
using System.Threading;

public class Example16
{
    public static void Main16()
    {
        // Create ten random doubles.
        Thread.CurrentCulture = CultureInfo.CreateSpecificCulture("en-US");
        double[] numbers = GetRandomNumbers(10);
        DisplayRandomNumbers(numbers);

        // Serialize the array.
        FileStream fsIn = new FileStream("randoms.dat", FileMode.Create);
        BinaryFormatter formatter = new BinaryFormatter();
        formatter.Serialize(fsIn, numbers);
        fsIn.Close();

        // Read the persisted data.
        FileStream fsOut = new FileStream("randoms.dat", FileMode.Open);
        double[] numbers1 = (Double[]) formatter.Deserialize(fsOut);
        fsOut.Close();

        // Display the data using the conventions of the en-US culture.
        Console.WriteLine("Current Culture: {0}",
            Thread.CurrentCulture.DisplayName);
        DisplayRandomNumbers(numbers1);

        // Display the data using the conventions of the fr-FR culture.
        Thread.CurrentCulture = CultureInfo.CreateSpecificCulture("fr-FR");
        Console.WriteLine("Current Culture: {0}",
            Thread.CurrentCulture.DisplayName);
        DisplayRandomNumbers(numbers1);
    }
}
```

```

private static double[] GetRandomNumbers(int n)
{
    Random rnd = new Random();
    double[] numbers = new double[n];
    for (int ctr = 0; ctr < n; ctr++)
        numbers[ctr] = rnd.NextDouble() * 1000;
    return numbers;
}

private static void DisplayRandomNumbers(double[] numbers)
{
    for (int ctr = 0; ctr < numbers.Length; ctr++)
        Console.WriteLine(numbers[ctr].ToString("R"));
    Console.WriteLine();
}
}

// The example displays output like the following:
//      932.10070623648392
//      96.868112262742642
//      857.111520067375
//      771.37727233179726
//      262.65733840999064
//      387.00796914613244
//      557.49389788019187
//      83.79498919648816
//      957.31006048494487
//      996.54487892824454
//
//      Current Culture: English (United States)
//      932.10070623648392
//      96.868112262742642
//      857.111520067375
//      771.37727233179726
//      262.65733840999064
//      387.00796914613244
//      557.49389788019187
//      83.79498919648816
//      957.31006048494487
//      996.54487892824454
//
//      Current Culture: French (France)
//      932,10070623648392
//      96,868112262742642
//      857,111520067375
//      771,37727233179726
//      262,65733840999064
//      387,00796914613244
//      557,49389788019187
//      83,79498919648816
//      957,31006048494487
//      996,54487892824454

```

```

Imports System.Globalization
Imports System.IO
Imports System.Runtime.Serialization.Formatters.Binary
Imports System.Threading

Module Example16
    Public Sub Main16()
        ' Create ten random doubles.
        Thread.CurrentThread.CurrentCulture = CultureInfo.CreateSpecificCulture("en-US")
        Dim numbers() As Double = GetRandomNumbers(10)
        DisplayRandomNumbers(numbers)

        ' Serialize the array.
        Dim fsIn As New FileStream("randoms.dat", FileMode.Create)

```

```

        Dim formatter As New BinaryFormatter()
        formatter.Serialize(fsIn, numbers)
        fsIn.Close()

        ' Read the persisted data.
        Dim fsOut As New FileStream("randoms.dat", FileMode.Open)
        Dim numbers1() As Double = DirectCast(formatter.Deserialize(fsOut), Double())
        fsOut.Close()

        ' Display the data using the conventions of the en-US culture.
        Console.WriteLine("Current Culture: {0}",
                           Thread.CurrentThread.CurrentCulture.DisplayName)
        DisplayRandomNumbers(numbers1)

        ' Display the data using the conventions of the fr-FR culture.
        Thread.CurrentThread.CurrentCulture = CultureInfo.CreateSpecificCulture("fr-FR")
        Console.WriteLine("Current Culture: {0}",
                           Thread.CurrentThread.CurrentCulture.DisplayName)
        DisplayRandomNumbers(numbers1)
    End Sub

    Private Function GetRandomNumbers(n As Integer) As Double()
        Dim rnd As New Random()
        Dim numbers(n - 1) As Double
        For ctr As Integer = 0 To n - 1
            numbers(ctr) = rnd.NextDouble * 1000
        Next
        Return numbers
    End Function

    Private Sub DisplayRandomNumbers(numbers As Double())
        For ctr As Integer = 0 To numbers.Length - 1
            Console.WriteLine(numbers(ctr).ToString("R"))
        Next
        Console.WriteLine()
    End Sub
End Module

' The example displays output like the following:
' 932.10070623648392
' 96.868112262742642
' 857.111520067375
' 771.37727233179726
' 262.65733840999064
' 387.00796914613244
' 557.49389788019187
' 83.79498919648816
' 957.31006048494487
' 996.54487892824454
'

' Current Culture: English (United States)
932.10070623648392
96.868112262742642
857.111520067375
771.37727233179726
262.65733840999064
387.00796914613244
557.49389788019187
83.79498919648816
957.31006048494487
996.54487892824454
'

' Current Culture: French (France)
932,10070623648392
96,868112262742642
857,111520067375
771,37727233179726
262,65733840999064
387,00796914613244
557,49389788019187

```

```
' 83,79498919648816
' 957,31006048494487
' 996,54487892824454
```

Serializing currency values is a special case. Because a currency value depends on the unit of currency in which it is expressed; it makes little sense to treat it as an independent numeric value. However, if you save a currency value as a formatted string that includes a currency symbol, it cannot be deserialized on a system whose default culture uses a different currency symbol, as the following example shows.

```
using System;
using System.Globalization;
using System.IO;
using System.Threading;

public class Example1
{
    public static void Main1()
    {
        // Display the currency value.
        Thread.CurrentThread.CurrentCulture = CultureInfo.CreateSpecificCulture("en-US");
        Decimal value = 16039.47m;
        Console.WriteLine("Current Culture: {0}", CultureInfo.CurrentCulture.DisplayName);
        Console.WriteLine("Currency Value: {0:C2}", value);

        // Persist the currency value as a string.
        StreamWriter sw = new StreamWriter("currency.dat");
        sw.Write(value.ToString("C2"));
        sw.Close();

        // Read the persisted data using the current culture.
        StreamReader sr = new StreamReader("currency.dat");
        string currencyData = sr.ReadToEnd();
        sr.Close();

        // Restore and display the data using the conventions of the current culture.
        Decimal restoredValue;
        if (Decimal.TryParse(currencyData, out restoredValue))
            Console.WriteLine(restoredValue.ToString("C2"));
        else
            Console.WriteLine("ERROR: Unable to parse '{0}'", currencyData);
        Console.WriteLine();

        // Restore and display the data using the conventions of the en-GB culture.
        Thread.CurrentThread.CurrentCulture = CultureInfo.CreateSpecificCulture("en-GB");
        Console.WriteLine("Current Culture: {0}",
                        Thread.CurrentThread.CurrentCulture.DisplayName);
        if (Decimal.TryParse(currencyData, NumberStyles.Currency, null, out restoredValue))
            Console.WriteLine(restoredValue.ToString("C2"));
        else
            Console.WriteLine("ERROR: Unable to parse '{0}'", currencyData);
        Console.WriteLine();
    }
}

// The example displays output like the following:
//      Current Culture: English (United States)
//      Currency Value: $16,039.47
//      ERROR: Unable to parse '$16,039.47'
//
//      Current Culture: English (United Kingdom)
//      ERROR: Unable to parse '$16,039.47'
```

```

Imports System.Globalization
Imports System.IO
Imports System.Threading

Module Example1
    Public Sub Main()
        ' Display the currency value.
        Thread.CurrentThread.CurrentCulture = CultureInfo.CreateSpecificCulture("en-US")
        Dim value As Decimal = 16039.47D
        Console.WriteLine("Current Culture: {0}", CultureInfo.CurrentCulture.DisplayName)
        Console.WriteLine("Currency Value: {0:C2}", value)

        ' Persist the currency value as a string.
        Dim sw As New StreamWriter("currency.dat")
        sw.WriteLine(value.ToString("C2"))
        sw.Close()

        ' Read the persisted data using the current culture.
        Dim sr As New StreamReader("currency.dat")
        Dim currencyData As String = sr.ReadToEnd()
        sr.Close()

        ' Restore and display the data using the conventions of the current culture.
        Dim restoredValue As Decimal
        If Decimal.TryParse(currencyData, restoredValue) Then
            Console.WriteLine(restoredValue.ToString("C2"))
        Else
            Console.WriteLine("ERROR: Unable to parse '{0}'", currencyData)
        End If
        Console.WriteLine()

        ' Restore and display the data using the conventions of the en-GB culture.
        Thread.CurrentThread.CurrentCulture = CultureInfo.CreateSpecificCulture("en-GB")
        Console.WriteLine("Current Culture: {0}",
            Thread.CurrentThread.CurrentCulture.DisplayName)
        If Decimal.TryParse(currencyData, NumberStyles.Currency, Nothing, restoredValue) Then
            Console.WriteLine(restoredValue.ToString("C2"))
        Else
            Console.WriteLine("ERROR: Unable to parse '{0}'", currencyData)
        End If
        Console.WriteLine()
    End Sub
End Module
' The example displays output like the following:
' Current Culture: English (United States)
' Currency Value: $16,039.47
' ERROR: Unable to parse '$16,039.47'
'
' Current Culture: English (United Kingdom)
' ERROR: Unable to parse '$16,039.47'

```

Instead, you should serialize the numeric value along with some cultural information, such as the name of the culture, so that the value and its currency symbol can be deserialized independently of the current culture. The following example does that by defining a `CurrencyValue` structure with two members: the `Decimal` value and the name of the culture to which the value belongs.

```

using System;
using System.Globalization;
using System.IO;
using System.Runtime.Serialization.Formatters.Binary;
using System.Threading;

public class Example2
{
    public static void Main2()
    {
        // Display the currency value.
        Thread.CurrentThread.CurrentCulture = CultureInfo.CreateSpecificCulture("en-US");
        Decimal value = 16039.47m;
        Console.WriteLine("Current Culture: {0}", CultureInfo.CurrentCulture.DisplayName);
        Console.WriteLine("Currency Value: {0:C2}", value);

        // Serialize the currency data.
        BinaryFormatter bf = new BinaryFormatter();
        FileStream fw = new FileStream("currency.dat", FileMode.Create);
        CurrencyValue data = new CurrencyValue(value, CultureInfo.CurrentCulture.Name);
        bf.Serialize(fw, data);
        fw.Close();
        Console.WriteLine();

        // Change the current culture.
        CultureInfo.CurrentCulture = CultureInfo.CreateSpecificCulture("en-GB");
        Console.WriteLine("Current Culture: {0}", CultureInfo.CurrentCulture.DisplayName);

        // Deserialize the data.
        FileStream fr = new FileStream("currency.dat", FileMode.Open);
        CurrencyValue restoredData = (CurrencyValue) bf.Deserialize(fr);
        fr.Close();

        // Display the original value.
        CultureInfo culture = CultureInfo.CreateSpecificCulture(restoredData.CultureName);
        Console.WriteLine("Currency Value: {0}", restoredData.Amount.ToString("C2", culture));
    }
}

[Serializable] internal struct CurrencyValue
{
    public CurrencyValue(Decimal amount, string name)
    {
        this.Amount = amount;
        this.CultureName = name;
    }

    public Decimal Amount;
    public string CultureName;
}

// The example displays the following output:
//      Current Culture: English (United States)
//      Currency Value: $16,039.47
//
//      Current Culture: English (United Kingdom)
//      Currency Value: $16,039.47

```

```

Imports System.Globalization
Imports System.IO
Imports System.Runtime.Serialization.Formatters.Binary
Imports System.Threading

<Serializable> Friend Structure CurrencyValue
    Public Sub New(amount As Decimal, name As String)
        Me.Amount = amount
        Me.CultureName = name
    End Sub

    Public Amount As Decimal
    Public CultureName As String
End Structure

Module Example2
    Public Sub Main2()
        ' Display the currency value.
        Thread.CurrentThread.CurrentCulture = CultureInfo.CreateSpecificCulture("en-US")
        Dim value As Decimal = 16039.47D
        Console.WriteLine("Current Culture: {0}", CultureInfo.CurrentCulture.DisplayName)
        Console.WriteLine("Currency Value: {0:C2}", value)

        ' Serialize the currency data.
        Dim bf As New BinaryFormatter()
        Dim fw As New FileStream("currency.dat", FileMode.Create)
        Dim data As New CurrencyValue(value, CultureInfo.CurrentCulture.Name)
        bf.Serialize(fw, data)
        fw.Close()
        Console.WriteLine()

        ' Change the current culture.
        CultureInfo.CurrentCulture = CultureInfo.CreateSpecificCulture("en-GB")
        Console.WriteLine("Current Culture: {0}", CultureInfo.CurrentCulture.DisplayName)

        ' Deserialize the data.
        Dim fr As New FileStream("currency.dat", FileMode.Open)
        Dim restoredData As CurrencyValue = CType(bf.Deserialize(fr), CurrencyValue)
        fr.Close()

        ' Display the original value.
        Dim culture As CultureInfo = CultureInfo.CreateSpecificCulture(restoredData.CultureName)
        Console.WriteLine("Currency Value: {0}", restoredData.Amount.ToString("C2", culture))
    End Sub
End Module

' The example displays the following output:
'   Current Culture: English (United States)
'   Currency Value: $16,039.47
'
'   Current Culture: English (United Kingdom)
'   Currency Value: $16,039.47

```

Work with culture-specific settings

In .NET, the [CultureInfo](#) class represents a particular culture or region. Some of its properties return objects that provide specific information about some aspect of a culture:

- The [CultureInfo.CompareInfo](#) property returns a [CompareInfo](#) object that contains information about how the culture compares and orders strings.
- The [CultureInfo.DateTimeFormat](#) property returns a [DateTimeFormatInfo](#) object that provides culture-specific information used in formatting date and time data.

- The [CultureInfo.NumberFormat](#) property returns a [NumberFormatInfo](#) object that provides culture-specific information used in formatting numeric data.
- The [CultureInfo.TextInfo](#) property returns a [TextInfo](#) object that provides information about the culture's writing system.

In general, do not make any assumptions about the values of specific [CultureInfo](#) properties and their related objects. Instead, you should view culture-specific data as subject to change, for these reasons:

- Individual property values are subject to change and revision over time, as data is corrected, better data becomes available, or culture-specific conventions change.
- Individual property values may vary across versions of .NET or operating system versions.
- .NET supports replacement cultures. This makes it possible to define a new custom culture that either supplements existing standard cultures or completely replaces an existing standard culture.
- On Windows systems, the user can customize culture-specific settings by using the [Region and Language](#) app in Control Panel. When you instantiate a [CultureInfo](#) object, you can determine whether it reflects these user customizations by calling the [CultureInfo\(String, Boolean\)](#) constructor. Typically, for end-user apps, you should respect user preferences so that the user is presented with data in a format that they expect.

See also

- [Globalization and localization](#)
- [Best practices for using strings](#)

.NET globalization and ICU

9/20/2022 • 7 minutes to read • [Edit Online](#)

Prior to .NET 5, the .NET globalization APIs used different underlying libraries on different platforms. On Unix, the APIs used [International Components for Unicode \(ICU\)](#), and on Windows, they used [National Language Support \(NLS\)](#). This resulted in some behavioral differences in a handful of globalization APIs when running applications on different platforms. Behavior differences were evident in these areas:

- Cultures and culture data
- String casing
- String sorting and searching
- Sort keys
- String normalization
- Internationalized Domain Names (IDN) support
- Time zone display name on Linux

Starting with .NET 5, developers have more control over which underlying library is used, enabling applications to avoid differences across platforms.

ICU on Windows

Windows 10 May 2019 Update and later versions include `icu.dll` as part of the OS, and .NET 5 and later versions use ICU by default. When running on Windows, .NET 5 and later versions try to load `icu.dll` and, if it's available, use it for the globalization implementation. If the ICU library can't be found or loaded, such as when running on older versions of Windows, .NET 5 and later versions fall back to the NLS-based implementation.

NOTE

Even when using ICU, the `CurrentCulture`, `CurrentUICulture`, and `CurrentRegion` members still use Windows operating system APIs to honor user settings.

Behavioral differences

If you upgrade your app to target .NET 5, you might see changes in your app even if you don't realize you're using globalization facilities. This section lists one of the behavioral changes you might see, but there are others too.

`String.IndexOf`

Consider the following code that calls `String.IndexOf(String)` to find the index of the null character `\0` in a string.

```
const string greeting = "He\0l\0o";
Console.WriteLine($"{greeting.IndexOf("\0")}");
Console.WriteLine($"{greeting.IndexOf("\0", StringComparison.CurrentCulture)}");
Console.WriteLine($"{greeting.IndexOf("\0", StringComparison.Ordinal)}");
```

- In .NET Core 3.1 and earlier versions on Windows, the snippet prints `3` on each of the three lines.
- In .NET 5 and later versions on Windows 19H1 and later versions, the snippet prints `0`, `0`, and `3` (for the ordinal search).

By default, `String.IndexOf(String)` performs a culture-aware linguistic search. ICU considers the null character `\0` to be a *zero-weight character*, and thus the character isn't found in the string when using a linguistic search on .NET 5 and later. However, NLS doesn't consider the null character `\0` to be a zero-weight character, and a linguistic search on .NET Core 3.1 and earlier locates the character at position 3. An ordinal search finds the character at position 3 on all .NET versions.

You can run code analysis rules [CA1307: Specify StringComparison for clarity](#) and [CA1309: Use ordinal StringComparison](#) to find call sites in your code where the string comparison isn't specified or it is not ordinal.

For more information, see [Behavior changes when comparing strings on .NET 5+](#).

Use NLS instead of ICU

Using ICU instead of NLS may result in behavioral differences with some globalization-related operations. To revert back to using NLS, a developer can opt out of the ICU implementation. Applications can enable NLS mode in any of the following ways:

- In the project file:

```
<ItemGroup>
    <RuntimeHostConfigurationOption Include="System.Globalization.UseNls" Value="true" />
</ItemGroup>
```

- In the `runtimeconfig.json` file:

```
{
  "runtimeOptions": {
    "configProperties": {
      "System.Globalization.UseNls": true
    }
  }
}
```

- By setting the environment variable `DOTNET_SYSTEM_GLOBALIZATION_USENLS` to the value `true` or `1`.

NOTE

A value set in the project or in the `runtimeconfig.json` file takes precedence over the environment variable.

For more information, see [Runtime config settings](#).

Determine if your app is using ICU

The following code snippet can help you determine if your app is running with ICU libraries (and not NLS).

```
public static bool ICUMode()
{
    SortVersion sortVersion = CultureInfo.InvariantCulture.CompareInfo.Version;
    byte[] bytes = sortVersion.SortId.ToByteArray();
    int version = bytes[3] << 24 | bytes[2] << 16 | bytes[1] << 8 | bytes[0];
    return version != 0 && version == sortVersion.FullVersion;
}
```

To determine the version of .NET, use `RuntimelInformation.FrameworkDescription`.

App-local ICU

Each release of ICU may bring with it bug fixes as well as updated Common Locale Data Repository (CLDR) data

that describes the world's languages. Moving between versions of ICU can subtly impact app behavior when it comes to globalization-related operations. To help application developers ensure consistency across all deployments, .NET 5 and later versions enable apps on both Windows and Unix to carry and use their own copy of ICU.

Applications can opt in to an app-local ICU implementation mode in any of the following ways:

- In the project file:

```
<ItemGroup>
  <RuntimeHostConfigurationOption Include="System.Globalization.AppLocalIcu" Value="<suffix>:<version> or <version>" />
</ItemGroup>
```

- In the `runtimeconfig.json` file:

```
{
  "runtimeOptions": {
    "configProperties": {
      "System.Globalization.AppLocalIcu": "<suffix>:<version> or <version>"
    }
  }
}
```

- By setting the environment variable `DOTNET_SYSTEM_GLOBALIZATION_APPLOCALICU` to the value

`<suffix>:<version>` OR `<version>`.

`<suffix>` : Optional suffix of fewer than 36 characters in length, following the public ICU packaging conventions. When building a custom ICU, you can customize it to produce the lib names and exported symbol names to contain a suffix, for example, `libicuucmyapp`, where `myapp` is the suffix.

`<version>` : A valid ICU version, for example, 67.1. This version is used to load the binaries and to get the exported symbols.

To load ICU when the app-local switch is set, .NET uses the `NativeLibrary.TryLoad` method, which probes multiple paths. The method first tries to find the library in the `NATIVE_DLL_SEARCH_DIRECTORIES` property, which is created by the dotnet host based on the `deps.json` file for the app. For more information, see [Default probing](#).

For self-contained apps, no special action is required by the user, other than making sure ICU is in the app directory (for self-contained apps, the working directory defaults to `NATIVE_DLL_SEARCH_DIRECTORIES`).

If you're consuming ICU via a NuGet package, this works in framework-dependent applications. NuGet resolves the native assets and includes them in the `deps.json` file and in the output directory for the application under the `runtimes` directory. .NET loads it from there.

For framework-dependent apps (not self contained) where ICU is consumed from a local build, you must take additional steps. The .NET SDK doesn't yet have a feature for "loose" native binaries to be incorporated into `deps.json` (see [this SDK issue](#)). Instead, you can enable this by adding additional information into the application's project file. For example:

```
<ItemGroup>
  <IcuAssemblies Include="icu\*.so*" />
  <RuntimeTargetsCopyLocalItems Include="@{IcuAssemblies}" AssetType="native" CopyLocal="true"
    DestinationSubDirectory="runtimes/linux-x64/native/" DestinationSubPath="%(FileName)%(Extension)"
    RuntimeIdentifier="linux-x64" NuGetPackageId="System.Private.Runtime.UnicodeData" />
</ItemGroup>
```

This must be done for all the ICU binaries for the supported runtimes. Also, the `NuGetPackageId` metadata in the `RuntimeTargetsCopyLocalItems` item group needs to match a NuGet package that the project actually references.

macOS behavior

macOS has a different behavior for resolving dependent dynamic libraries from the load commands specified in the `Mach-O` file than the Linux loader. In the Linux loader, .NET can try `libcudata`, `libicuuc`, and `libicui18n` (in that order) to satisfy ICU dependency graph. However, on macOS, this doesn't work. When building ICU on macOS, you, by default, get a dynamic library with these load commands in `libicuuc`. The following snippet shows an example.

```
~/ % otool -L /Users/santifdezm/repos/icu-build/icu/install/lib/libicuuc.67.1.dylib
/Users/santifdezm/repos/icu-build/icu/install/lib/libicuuc.67.1.dylib:
    libicuuc.67.dylib (compatibility version 67.0.0, current version 67.1.0)
    libcudata.67.dylib (compatibility version 67.0.0, current version 67.1.0)
    /usr/lib/libSystem.B.dylib (compatibility version 1.0.0, current version 1281.100.1)
    /usr/lib/libc++.1.dylib (compatibility version 1.0.0, current version 902.1.0)
```

These commands just reference the name of the dependent libraries for the other components of ICU. The loader performs the search following the `dlopen` conventions, which involves having these libraries in the system directories or setting the `LD_LIBRARY_PATH` env vars, or having ICU at the app-level directory. If you can't set `LD_LIBRARY_PATH` or ensure that ICU binaries are at the app-level directory, you will need to do some extra work.

There are some directives for the loader, like `@loader_path`, which tell the loader to search for that dependency in the same directory as the binary with that load command. There are two ways to achieve this:

- `install_name_tool -change`

Run the following commands:

```
install_name_tool -change "libcudata.67.dylib" "@loader_path/libcudata.67.dylib"
/path/to/libicuuc.67.1.dylib
install_name_tool -change "libcudata.67.dylib" "@loader_path/libcudata.67.dylib"
/path/to/libicui18n.67.1.dylib
install_name_tool -change "libicuuc.67.dylib" "@loader_path/libicuuc.67.dylib"
/path/to/libicui18n.67.1.dylib
```

- Patch ICU to produce the install names with `@loader_path`

Before running autoconf (`./runConfigureICU`), change [these lines](#) to:

```
LD SONAME = -Wl,-compatibility_version -Wl,$(SO_TARGET_VERSION_MAJOR) -Wl,-current_version -
Wl,$(SO_TARGET_VERSION) -install_name @loader_path/$(notdir $(MIDDLE_SO_TARGET))
```

ICU on WebAssembly

A version of ICU is available that's specifically for WebAssembly workloads. This version provides globalization compatibility with desktop profiles. To reduce the ICU data file size from 24 MB to 1.4 MB (or ~0.3 MB if compressed with Brotli), this workload has a handful of limitations.

The following APIs are not supported:

- [CultureInfo.EnglishName](#)
- [CultureInfo.NativeName](#)
- [DateTimeFormatInfo.NativeCalendarName](#)

- [RegionInfo.NativeName](#)

The following APIs are supported with limitations:

- [String.Normalize\(NormalizationForm\)](#) and [String.IsNormalized\(NormalizationForm\)](#) don't support the rarely used [FormKC](#) and [FormKD](#) forms.
- [RegionInfo.CurrencyNativeName](#) returns the same value as [RegionInfo.CurrencyEnglishName](#).

In addition, a list of supported locales can be found on the [dotnet/icu repo](#).

Localizability review

9/20/2022 • 3 minutes to read • [Edit Online](#)

The localizability review is an intermediate step in the development of a world-ready application. It verifies that a globalized application is ready for localization and identifies any code or any aspects of the user interface that require special handling. This step also helps ensure that the localization process will not introduce any functional defects into your application. When all the issues raised by the localizability review have been addressed, your application is ready for localization. If the localizability review is thorough, you should not have to modify any source code during the localization process.

The localizability review consists of the following three checks:

- [Are the globalization recommendations implemented?](#)
- [Are culture-sensitive features handled correctly?](#)
- [Have you tested your application with international data?](#)

Implement globalization recommendations

If you have designed and developed your application with localization in mind, and if you have followed the recommendations discussed in the [Globalization](#) article, the localizability review will largely be a quality assurance pass. Otherwise, during this stage you should review and implement the recommendations for [globalization](#) and fix the errors in source code that prevent localization.

Handle culture-sensitive features

.NET does not provide programmatic support in a number of areas that vary widely by culture. In most cases, you have to write custom code to handle feature areas like the following:

- Addresses
- Telephone numbers
- Paper sizes
- Units of measure used for lengths, weights, area, volume, and temperatures

Although .NET does not offer built-in support for converting between units of measure, you can use the [RegionInfo.IsMetric](#) property to determine whether a particular country or region uses the metric system, as the following example illustrates.

```

using System;
using System.Globalization;

public class Example
{
    public static void Main()
    {
        string[] cultureNames = { "en-US", "en-GB", "fr-FR",
                                  "ne-NP", "es-BO", "ig-NG" };
        foreach (var cultureName in cultureNames) {
            RegionInfo region = new RegionInfo(cultureName);
            Console.WriteLine("{0} {1} the metric system.", region.EnglishName,
                             region.IsMetric ? "uses" : "does not use");
        }
    }
}
// The example displays the following output:
//      United States does not use the metric system.
//      United Kingdom uses the metric system.
//      France uses the metric system.
//      Nepal uses the metric system.
//      Bolivia uses the metric system.
//      Nigeria uses the metric system.

```

```

Imports System.Globalization

Module Example
    Public Sub Main()
        Dim cultureNames() As String = {"en-US", "en-GB", "fr-FR",
                                         "ne-NP", "es-BO", "ig-NG"}
        For Each cultureName In cultureNames
            Dim region As New RegionInfo(cultureName)
            Console.WriteLine("{0} {1} the metric system.", region.EnglishName,
                             If(region.IsMetric, "uses", "does not use"))
        Next
    End Sub
End Module
' The example displays the following output:
'      United States does not use the metric system.
'      United Kingdom uses the metric system.
'      France uses the metric system.
'      Nepal uses the metric system.
'      Bolivia uses the metric system.
'      Nigeria uses the metric system.

```

Test your application

Before you localize your application, you should test it by using international data on international versions of the operating system. Although most of the user interface will not be localized at this point, you will be able to detect problems such as the following:

- Serialized data that does not deserialize correctly across operating system versions.
- Numeric data that does not reflect the conventions of the current culture. For example, numbers may be displayed with inaccurate group separators, decimal separators, or currency symbols.
- Date and time data that does not reflect the conventions of the current culture. For example, numbers that represent the month and day may appear in the wrong order, date separators may be incorrect, or time zone information may be incorrect.
- Resources that cannot be found because you have not identified a default culture for your application.

- Strings that are displayed in an unusual order for the specific culture.
- String comparisons or comparisons for equality that return unexpected results.

If you've followed the globalization recommendations when developing your application, handled culture-sensitive features correctly, and identified and addressed the localization issues that arose during testing, you can proceed to the next step, [Localization](#).

See also

- [Globalization and Localization](#)
- [Localization](#)
- [Globalization](#)
- [Resources in .NET apps](#)

Localization in .NET

9/20/2022 • 8 minutes to read • [Edit Online](#)

Localization is the process of translating an application's resources into localized versions for each culture that the application will support. You should proceed to the localization step only after completing the [Localizability review](#) step to verify that the globalized application is ready for localization.

An application that is ready for localization is separated into two conceptual blocks: a block that contains all user interface elements and a block that contains executable code. The user interface block contains only localizable user-interface elements such as strings, error messages, dialog boxes, menus, embedded object resources, and so on for the neutral culture. The code block contains only the application code to be used by all supported cultures. The common language runtime supports a satellite assembly resource model that separates an application's executable code from its resources. For more information about implementing this model, see [Resources in .NET](#).

For each localized version of your application, add a new satellite assembly that contains the localized user interface block translated into the appropriate language for the target culture. The code block for all cultures should remain the same. The combination of a localized version of the user interface block with the code block produces a localized version of your application.

In this article, you will learn how to use the `IStringLocalizer<T>` and `IStringLocalizerFactory` implementations. All of the example source code in this article relies on the `Microsoft.Extensions.Localization` and `Microsoft.Extensions.Hosting` NuGet packages. For more information on hosting, see [.NET Generic Host](#).

Resource files

The primary mechanism for isolating localizable strings is with **resource files**. A resource file is an XML file with the `.resx` file extension. Resource files are translated prior to the execution of the consuming application — in other words, they represent translated content at rest. A resource file name most commonly contains a locale identifier, and takes on the following form:

```
<FullTypeName><.Locale>.resx
```

Where:

- The `<FullTypeName>` represents localizable resources for a specific type.
- The optional `<.Locale>` represents the locale of the resource file contents.

Specifying locales

The locale should define the language, at a bare minimum, but it can also define the culture (dialect), and even the country. These segments are commonly delimited by the `-` character. With the added specificity of a culture, the "culture fallback" rules are applied where best matches are prioritized. The locale should map to a well-known language tag. For more information, see [CultureInfo.Name](#).

Culture fallback scenarios

Imagine that your localized app supports various Serbian locales, and has the following resource files for its `MessageService`:

FILE	DIALECT	COUNTRY CODE
<code>MessageService sr-Cyrl-RS.resx</code>	(Cyrillic, Serbia)	RS

FILE	DIALECT	COUNTRY CODE
<i>MessageService.sr-Cyrl.resx</i>	Cyrillic	
<i>MessageService.sr-Latn-BA.resx</i>	(Latin, Bosnia & Herzegovina)	BA
<i>MessageService.sr-Latn-ME.resx</i>	(Latin, Montenegro)	ME
<i>MessageService.sr-Latn-RS.resx</i>	(Latin, Serbia)	RS
<i>MessageService.sr-Latn.resx</i>	Latin	
<i>MessageService.sr.resx</i>	[†] Latin	
<i>MessageService.resx</i>		

[†] The default dialect for the language.

When your app is running with the `CultureInfo.CurrentCulture` set to a culture of `"sr-Cyrl-RS"` localization attempts to resolve files in the following order:

1. *MessageService.sr-Cyrl-RS.resx*
2. *MessageService.sr-Cyrl.resx*
3. *MessageService.sr.resx*
4. *MessageService.resx*

However, if your app was running with the `CultureInfo.CurrentCulture` set to a culture of `"sr-Latn-BA"` localization attempts to resolve files in the following order:

1. *MessageService.sr-Latn-BA.resx*
2. *MessageService.sr-Latn.resx*
3. *MessageService.sr.resx*
4. *MessageService.resx*

The "culture fallback" rule will ignore locales when there are no corresponding matches, meaning resource file number four is selected if it's unable to find a match. If the culture was set to `"fr-FR"`, localization would end up falling to the *MessageService.resx* file which can be problematic. For more information, see [The resource fallback process](#).

Resource lookup

Resource files are automatically resolved as part of a lookup routine. If your project file name is different than the root namespace of your project, the assembly name might differ. This can prevent resource lookup from being otherwise successful. To address this mismatch, use the `RootNamespaceAttribute` to provide a hint to the localization services. When provided, it is used during resource lookup.

The example project is named *example.csproj*, which creates an *example.dll* and *example.exe* — however, the `Localization.Example` namespace is used. Apply an `assembly` level attribute to correct this mismatch:

```
[assembly: RootNamespace("Localization.Example")]
```

Register localization services

To register localization services, call one of the `AddLocalization` extension methods during the configuration of

services. This will enable dependency injection (DI) of the following types:

- `Microsoft.Extensions.Localization.IStringLocalizer<T>`
- `Microsoft.Extensions.Localization.IStringLocalizerFactory`

Configure localization options

The `AddLocalization(IServiceCollection, Action<LocalizationOptions>)` overload accepts a `setupAction` parameter of type `Action<LocalizationOptions>`. This allows you to configure localization options.

```
using IHost host = Host.CreateDefaultBuilder(args)
    .ConfigureServices(services =>
{
    services.AddLocalization(options =>
    {
        options.ResourcesPath = "Resources";
    });
});

// Omitted for brevity.
```

Resource files can live anywhere in a project, but there are common practices in place that have proven to be successful. More often than not, the path of least resistance is followed. The preceding C# code:

- Creates the default host builder.
- Calls `HostBuilder.ConfigureServices` with the `IServiceCollection` overload.
- Calls `AddLocalization` to the service collection, specifying `LocalizationOptions.ResourcesPath` as `"Resources"`.

This would cause the localization services to look in the *Resources* directory for resource files.

Use `IStringLocalizer<T>` and `IStringLocalizerFactory`

After you've [registered](#) (and optionally [configured](#)) the localization services, you can use the following types with DI:

- `IStringLocalizer<T>`
- `IStringLocalizerFactory`

To create a message service that is capable of returning localized strings, consider the following `MessageService`:

```
using System.Diagnostics.CodeAnalysis;
using Microsoft.Extensions.Localization;

namespace Localization.Example;

public class MessageService
{
    private readonly IStringLocalizer<MessageService> _localizer = null!;

    public MessageService(IStringLocalizer<MessageService> localizer) =>
        _localizer = localizer;

    [return: NotNullIfNotNull("_localizer")]
    public string? GetGreetingMessage()
    {
        LocalizedString localizedString = _localizer["GreetingMessage"];
        return localizedString;
    }
}
```

In the preceding C# code:

- A `IStringLocalizer<MessageService> _localizer` field is declared.
- The constructor takes a `IStringLocalizer<MessageService>` parameter and assigns it to the `_localizer` field.
- The `GetGreetingMessage` method invokes the `IStringLocalizer.Item[String]` passing `"GreetingMessage"` as an argument.

The `IStringLocalizer` also supports parameterized string resources, consider the following

`ParameterizedMessageService` :

```
using System.Diagnostics.CodeAnalysis;
using Microsoft.Extensions.Localization;

namespace Localization.Example;

public class ParameterizedMessageService
{
    private readonly IStringLocalizer _localizer = null!;

    public ParameterizedMessageService(IStringLocalizerFactory factory) =>
        _localizer = factory.Create(typeof(ParameterizedMessageService));

    [return: NotNullIfNotNull("_localizer")]
    public string? GetFormattedMessage(DateTime dateTime, double dinnerPrice)
    {
        LocalizedString localizedString = _localizer["DinnerPriceFormat", dateTime, dinnerPrice];
        return localizedString;
    }
}
```

In the preceding C# code:

- A `IStringLocalizer _localizer` field is declared.
- The constructor takes an `IStringLocalizerFactory` parameter, which is used to create an `IStringLocalizer` from the `ParameterizedMessageService` type, and assigns it to the `_localizer` field.
- The `GetFormattedMessage` method invokes `IStringLocalizer.Item[String, Object[]]`, passing `"DinnerPriceFormat"`, a `dateTime` object, and `dinnerPrice` as arguments.

IMPORTANT

The `IStringLocalizerFactory` isn't required. Instead, it is preferred for consuming services to require the `IStringLocalizer<T>`.

Both `IStringLocalizer.Item[]` indexers return a `LocalizedString`, which have [implicit conversions](#) to `string?`.

Put it all together

To exemplify an app using both message services, along with localization and resource files, consider the following `Program.cs` file:

```

using System.Globalization;
using Localization.Example;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Localization;
using Microsoft.Extensions.Logging;
using static System.Console;
using static System.Text.Encoding;

[assembly: RootNamespace("Localization.Example")]

OutputEncoding = Unicode;

if (args is { Length: 1 })
{
    CultureInfo.CurrentCulture =
        CultureInfo.CurrentUICulture =
            CultureInfo.GetCultureInfo(args[0]);
}

using IHost host = Host.CreateDefaultBuilder(args)
    .ConfigureServices(services =>
    {
        services.AddLocalization();
        services.AddTransient<MessageService>();
        services.AddTransient<ParameterizedMessageService>();
    })
    .ConfigureLogging(options => options.SetMinimumLevel(LogLevel.Warning))
    .Build();

IServiceProvider services = host.Services;

ILogger logger =
    services.GetRequiredService<ILoggerFactory>()
        .CreateLogger("Localization.Example");

MessageService messageService =
    services.GetRequiredService<MessageService>();
logger.LogWarning(
    messageService.GetGreetingMessage());

ParameterizedMessageService parameterizedMessageService =
    services.GetRequiredService<ParameterizedMessageService>();
logger.LogWarning(
    parameterizedMessageService.GetFormattedMessage(
        DateTime.Today.AddDays(-3), 37.63));

await host.RunAsync();

```

In the preceding C# code:

- The [RootNamespaceAttribute](#) sets `"Localization.Example"` as the root namespace.
- The [Console.OutputEncoding](#) is assigned to [Encoding.Unicode](#).
- When a single argument is passed to `args`, the [CultureInfo.CurrentCulture](#) and [CultureInfo.CurrentUICulture](#) are assigned the result of [CultureInfo.GetCultureInfo\(String\)](#) given the `arg[0]`.
- The [Host](#) is created with [defaults](#).
- The localization services, `MessageService`, and `ParameterizedMessageService` are registered to the `IServiceCollection` for DI.
- To remove noise, logging is configured to ignore any log level lower than a warning.
- The `MessageService` is resolved from the `IServiceProvider` instance and its resulting message is logged.
- The `ParameterizedMessageService` is resolved from the `IServiceProvider` instance and its resulting formatted message is logged.

Each of the `*MessageService` classes defines a set of `.resx` files, each with a single entry. Here is the example content for the `MessageService` resource files, starting with `MessageService.resx`.

```
<?xml version="1.0" encoding="utf-8"?>
<root>
  <data name="GreetingMessage" xml:space="preserve">
    <value>Hi friends, the ".NET" developer community is excited to see you here!</value>
  </data>
</root>
```

`MessageService.sr-Cyrl-RS.resx`:

```
<?xml version="1.0" encoding="utf-8"?>
<root>
  <data name="GreetingMessage" xml:space="preserve">
    <value>Здраво пријатељи, ".NET" девелопер заједница је узбуђена што вас види овде!</value>
  </data>
</root>
```

`MessageService.sr-Latn.resx`:

```
<?xml version="1.0" encoding="utf-8"?>
<root>
  <data name="GreetingMessage" xml:space="preserve">
    <value>Zdravo prijatelji, ".NET" developer zajednica je uzbudena sto vas vidi ovde!</value>
  </data>
</root>
```

Here is the example content for the `ParameterizedMessageService` resource files, starting with `ParameterizedMessageService.resx`.

```
<?xml version="1.0" encoding="utf-8"?>
<root>
  <data name="DinnerPriceFormat" xml:space="preserve">
    <value>On {0:D} my dinner cost {1:C}.</value>
  </data>
</root>
```

`ParameterizedMessageService.sr-Cyrl-RS.resx`:

```
<?xml version="1.0" encoding="utf-8"?>
<root>
  <data name="DinnerPriceFormat" xml:space="preserve">
    <value>У {0:D} моја вечера је коштала {1:C}.</value>
  </data>
</root>
```

`ParameterizedMessageService.sr-Latn.resx`:

```
<?xml version="1.0" encoding="utf-8"?>
<root>
  <data name="DinnerPriceFormat" xml:space="preserve">
    <value>U {0:D} moja vecera je kostala {1:C}.</value>
  </data>
</root>
```

TIP

All of the resource file XML comments, schema, and `<resheader>` elements are intentionally omitted for brevity.

Example runs

The following example runs show the various localized outputs, given targeted locales.

Consider "sr-Latn" :

```
dotnet run --project .\example\example.csproj sr-Latn

warn: Localization.Example[0]
      Zdravo prijatelji, ".NET" developer zajednica je uzbudena što vas vidi ovde!
warn: Localization.Example[0]
      U utorak, 03. avgust 2021. moja večera je koštala 37,63 ₠.
```

When omitting an argument to the [.NET CLI to run](#) the project, the default system culture is used — in this case

"en-US" :

```
dotnet run --project .\example\example.csproj

warn: Localization.Example[0]
      Hi friends, the ".NET" developer community is excited to see you here!
warn: Localization.Example[0]
      On Tuesday, August 3, 2021 my dinner cost $37.63.
```

When passing "sr-Cryl-RS" , the correct corresponding resource files are found and the localization applied:

```
dotnet run --project .\example\example.csproj sr-Cryl-RS

warn: Localization.Example[0]
      Здраво пријатељи, ".NET" девелопер заједница је узбуђена што вас види овде!
warn: Localization.Example[0]
      У уторак, 03. август 2021. моја вечера је коштала 38 RSD.
```

The sample application does not provide resource files for "fr-CA" , but when called with that culture, the non-localized resource files are used.

WARNING

Since the culture is found but the correct resource files are not, when formatting is applied you end up with partial localization:

```
dotnet run --project .\example\example.csproj fr-CA

warn: Localization.Example[0]
      Hi friends, the ".NET" developer community is excited to see you here!
warn: Localization.Example[0]
      On mardi 3 août 2021 my dinner cost 37,63 $.
```

See also

- [Globalizing and localizing .NET applications](#)
- [Package and deploy resources in .NET apps](#)

- [Microsoft.Extensions.Localization](#)

- [Dependency injection in .NET](#)

- [Logging in .NET](#)

- [ASP.NET Core localization](#)

Perform culture-insensitive string operations

9/20/2022 • 2 minutes to read • [Edit Online](#)

Culture-sensitive string operations are advantageous if you're creating applications designed to display results to users on a per-culture basis. By default, culture-sensitive methods obtain the culture to use from the [CurrentCulture](#) property for the current thread.

Sometimes, culture-sensitive string operations are not the desired behavior. Using culture-sensitive operations when results should be independent of culture can cause application code to fail on cultures with custom case mappings and sorting rules. For an example, see the [String Comparisons that Use the Current Culture](#) section in [Best Practices for Using Strings](#).

Whether string operations should be culture-sensitive or culture-insensitive depends on how your application uses the results. String operations that display results to the user should typically be culture-sensitive. For example, if an application displays a sorted list of localized strings in a list box, the application should perform a culture-sensitive sort.

Results of string operations that are used internally should typically be culture-insensitive. In general, if the application is working with file names, persistence formats, or symbolic information that is not displayed to the user, results of string operations should not vary by culture. For example, if an application compares a string to determine whether it is a recognized XML tag, the comparison should not be culture-sensitive. In addition, if a security decision is based on the result of a string comparison or case change operation, the operation should be culture-insensitive to ensure that the result is not affected by the value of [CurrentCulture](#).

Most .NET methods that *by default* perform culture-sensitive string operations also provide an overload that allows you to guarantee culture-insensitive results. These overloads that take a [CultureInfo](#) argument allow you to eliminate cultural variations in case mappings and sorting rules. For culture-insensitive string operations, specify the culture as [CultureInfo.InvariantCulture](#).

In this section

The articles in this section demonstrate how to perform culture-insensitive string operations using .NET methods that are culture-sensitive by default.

[Performing culture-insensitive string comparisons](#)

Describes how to use the [String.Compare](#) and [String.CompareTo](#) methods to perform culture-insensitive string comparisons.

[Performing culture-insensitive case changes](#)

Describes how to use the [String.ToUpper](#), [String.ToLower](#), [Char.ToUpper](#), and [Char.ToLower](#) methods to perform culture-insensitive case changes.

[Performing culture-insensitive string operations in collections](#)

Describes how to use the [CaseInsensitiveComparer](#), [CaseInsensitiveHashCodeProvider](#) class, [SortedList](#), [ArrayList.Sort](#) and [CollectionsUtil.CreateCaseInsensitiveHashtable](#) to perform culture-insensitive operations in collections.

[Performing culture-insensitive string operations in arrays](#)

Describes how to use the [Array.Sort](#) and [Array.BinarySearch](#) methods to perform culture-insensitive operations in arrays.

See also

- [Sorting Weight Tables](#) (for .NET on Windows systems)
- [Default Unicode Collation Element Table](#) (for .NET Core on Linux and macOS)

Perform culture-insensitive string comparisons

9/20/2022 • 2 minutes to read • [Edit Online](#)

By default, the [String.Compare](#) method performs culture-sensitive and case-sensitive comparisons. This method also includes several overloads that provide a `culture` parameter that lets you specify the culture to use, and a `comparisonType` parameter that lets you specify the comparison rules to use. Calling these methods instead of the default overload removes any ambiguity about the rules used in a particular method call, and makes it clear whether a particular comparison is culture-sensitive or culture-insensitive.

NOTE

Both overloads of the [String.CompareTo](#) method perform culture-sensitive and case-sensitive comparisons; you cannot use this method to perform culture-insensitive comparisons. For code clarity, we recommend that you use the [String.Compare](#) method instead.

For culture-sensitive operations, specify the [StringComparison.CurrentCulture](#) or [StringComparison.CurrentCultureIgnoreCase](#) enumeration value as the `comparisonType` parameter. If you want to perform a culture-sensitive comparison using a designated culture other than the current culture, specify the [CultureInfo](#) object that represents that culture as the `culture` parameter.

The culture-insensitive string comparisons supported by the [String.Compare](#) method are either linguistic (based on the sorting conventions of the invariant culture) or non-linguistic (based on the ordinal value of the characters in the string). Most culture-insensitive string comparisons are non-linguistic. For these comparisons, specify the [StringComparison.Ordinal](#) or [StringComparison.OrdinalIgnoreCase](#) enumeration value as the `comparisonType` parameter. For example, if a security decision (such as a user name or password comparison) is based on the result of a string comparison, the operation should be culture-insensitive and non-linguistic to ensure that the result is not affected by the conventions of a particular culture or language.

Use culture-insensitive linguistic string comparison if you want to handle linguistically relevant strings from multiple cultures in a consistent way. For example, if your application displays words that use multiple character sets in a list box, you might want to display words in the same order regardless of the current culture. For culture-insensitive linguistic comparisons, .NET defines an invariant culture that is based on the linguistic conventions of English. To perform a culture-insensitive linguistic comparison, specify [StringComparison.InvariantCulture](#) or [StringComparison.InvariantCultureIgnoreCase](#) as the `comparisonType` parameter.

The following example performs two culture-insensitive, non-linguistic string comparisons. The first is case-sensitive, but the second is not.

```

using System;

public class CompareSample
{
    public static void Main()
    {
        string string1 = "file";
        string string2 = "FILE";
        int compareResult = 0;

        compareResult = String.Compare(string1, string2,
                                       StringComparison.OrdinalIgnoreCase);
        Console.WriteLine("{0} comparison of '{1}' and '{2}': {3}",
                          StringComparison.OrdinalIgnoreCase, string1, string2,
                          compareResult);

        compareResult = String.Compare(string1, string2,
                                       StringComparison.OrdinalIgnoreCaseIgnoreCase);
        Console.WriteLine("{0} comparison of '{1}' and '{2}': {3}",
                          StringComparison.OrdinalIgnoreCaseIgnoreCase, string1, string2,
                          compareResult);
    }
}

// The example displays the following output:
//      Ordinal comparison of 'file' and 'FILE': 32
//      OrdinalIgnoreCase comparison of 'file' and 'FILE': 0

```

```

Public Class CompareSample
    Public Shared Sub Main()
        Dim string1 As String = "file"
        Dim string2 As String = "FILE"
        Dim compareResult As Integer

        compareResult = String.Compare(string1, string2, _
                                       StringComparison.OrdinalIgnoreCase)
        Console.WriteLine("{0} comparison of '{1}' and '{2}': {3}",
                          StringComparison.OrdinalIgnoreCase, string1, string2,
                          compareResult)

        compareResult = String.Compare(string1, string2,
                                       StringComparison.OrdinalIgnoreCaseIgnoreCase)
        Console.WriteLine("{0} comparison of '{1}' and '{2}': {3}",
                          StringComparison.OrdinalIgnoreCaseIgnoreCase, string1, string2,
                          compareResult)
    End Sub
End Class
'The example displays the following output:
'      Ordinal comparison of 'file' and 'FILE': 32
'      OrdinalIgnoreCase comparison of 'file' and 'FILE': 0

```

You can download the [Sorting Weight Tables](#), a set of text files that contain information on the character weights used in sorting and comparison operations for Windows operating systems, and the [Default Unicode Collation Element Table](#), the sort weight table for Linux and macOS.

See also

- [String.Compare](#)
- [String.CompareTo](#)
- [Perform culture-insensitive string operations](#)
- [Best practices for using strings](#)

Perform culture-insensitive case changes

9/20/2022 • 2 minutes to read • [Edit Online](#)

The `String.ToUpper`, `String.ToLower`, `Char.ToUpper`, and `Char.ToLower` methods provide overloads that do not accept any parameters. By default, these overloads without parameters perform case changes based on the value of the `CultureInfo.CurrentCulture`. This produces case-sensitive results that can vary by culture. To make it clear whether you want case changes to be culture-sensitive or culture-insensitive, you should use the overloads of these methods that require you to explicitly specify a `culture` parameter. For culture-sensitive case changes, specify `CultureInfo.CurrentCulture` for the `culture` parameter. For culture-insensitive case changes, specify `CultureInfo.InvariantCulture` for the `culture` parameter.

Often, strings are converted to a standard case to enable easier lookup later. When strings are used in this way, you should specify `CultureInfo.InvariantCulture` for the `culture` parameter, because the value of `Thread.CurrentCulture` can potentially change between the time that the case is changed and the time that the lookup occurs.

If a security decision is based on a case change operation, the operation should be culture-insensitive to ensure that the result is not affected by the value of `CultureInfo.CurrentCulture`. See the "String Comparisons that Use the Current Culture" section of the [Best Practices for Using Strings](#) article for an example that demonstrates how culture-sensitive string operations can produce inconsistent results.

Using the `String.ToUpper` and `String.ToLower` Methods

For code clarity, it is recommended that you always use overloads of the `String.ToUpper` and `String.ToLower` methods that allow you to specify a `culture` parameter explicitly. For example, the following code performs an identifier lookup. The `key.ToLower` operation is culture-sensitive by default, but this behavior is not clear from reading the code.

Example

```
Shared Function LookupKey(key As String) As Object
    Return internalHashtable(key.ToLower())
End Function
```

```
static object LookupKey(string key)
{
    return internalHashtable[key.ToLower()];
}
```

If you want the `key.ToLower` operation to be culture-insensitive, you should change the preceding example as follows to explicitly use the `CultureInfo.InvariantCulture` when changing the case.

```
Shared Function LookupKey(key As String) As Object
    Return internalHashtable(key.ToLower(CultureInfo.InvariantCulture))
End Function
```

```
static object LookupKey(string key)
{
    return internalHashtable[key.ToLower(CultureInfo.InvariantCulture)];
}
```

Using the Char.ToUpper and Char.ToLower Methods

Although the `Char.ToUpper` and `Char.ToLower` methods have the same characteristics as the `String.ToUpper` and `String.ToLower` methods, the only cultures that are affected are Turkish (Turkey) and Azerbaijani (Latin, Azerbaijan). These are the only two cultures with single-character casing differences. For more details about this unique case mapping, see the "Casing" section in the [String](#) class topic. For code clarity and to ensure consistent results, it is recommended that you always use the overloads of these methods that allow you to explicitly specify a `culture` parameter.

See also

- [String.ToUpper](#)
- [String.ToLower](#)
- [Char.ToUpper](#)
- [Char.ToLower](#)
- [Change case in .NET](#)
- [Perform culture-insensitive string operations](#)

Perform culture-insensitive string operations in collections

9/20/2022 • 3 minutes to read • [Edit Online](#)

There are classes and members in the [System.Collections](#) namespace that provide culture-sensitive behavior by default. The parameterless constructors for the [CaseInsensitiveComparer](#) and [CaseInsensitiveHashCodeProvider](#) classes initialize a new instance using the [Thread.CurrentCulture](#) property. All overloads of the [CollectionsUtil.CreateCaseInsensitiveHashtable](#) method create a new instance of the [Hashtable](#) class using the [Thread.CurrentCulture](#) property by default. Overloads of the [ArrayList.Sort](#) method perform culture-sensitive sorts by default using [Thread.CurrentCulture](#). Sorting and lookup in a [SortedList](#) can be affected by [Thread.CurrentCulture](#) when strings are used as the keys. Follow the usage recommendations provided in this section to obtain culture-insensitive results from these classes and methods in the [Collections](#) namespace.

NOTE

Passing [CultureInfo.InvariantCulture](#) to a comparison method does perform a culture-insensitive comparison. However, it does not cause a non-linguistic comparison, for example, for file paths, registry keys, and environment variables. Neither does it support security decisions based on the comparison result. For a non-linguistic comparison or support for result-based security decisions, the application should use a comparison method that accepts a [StringComparison](#) value. The application should then pass [StringComparison](#).

Use the [CaseInsensitiveComparer](#) and [CaseInsensitiveHashCodeProvider](#) classes

The parameterless constructors for [CaseInsensitiveHashCodeProvider](#) and [CaseInsensitiveComparer](#) initialize a new instance of the class using the [Thread.CurrentCulture](#), resulting in culture-sensitive behavior. The following code example demonstrates the constructor for a [Hashtable](#) that is culture-sensitive because it uses the parameterless constructors for [CaseInsensitiveHashCodeProvider](#) and [CaseInsensitiveComparer](#).

```
internalHashtable = New Hashtable(CaseInsensitiveHashCodeProvider.Default, CaseInsensitiveComparer.Default)
```

```
internalHashtable = new Hashtable(CaseInsensitiveHashCodeProvider.Default, CaseInsensitiveComparer.Default);
```

If you want to create a culture-insensitive [Hashtable](#) using the [CaseInsensitiveComparer](#) and [CaseInsensitiveHashCodeProvider](#) classes, initialize new instances of these classes using the constructors that accept a [culture](#) parameter. For the [culture](#) parameter, specify [CultureInfo.InvariantCulture](#). The following code example demonstrates the constructor for a culture-insensitive [Hashtable](#).

```
internalHashtable = New Hashtable(New  
CaseInsensitiveHashCodeProvider(CultureInfo.InvariantCulture),  
New CaseInsensitiveComparer(CultureInfo.InvariantCulture))
```

```
internalHashtable = new Hashtable(new CaseInsensitiveHashCodeProvider
    (CultureInfo.InvariantCulture),
    new CaseInsensitiveComparer(CultureInfo.InvariantCulture));
```

Use the `CollectionsUtil.CreateCaseInsensitiveHashTable` method

The `CollectionsUtil.CreateCaseInsensitiveHashTable` method is a useful shortcut for creating a new instance of the `Hashtable` class that ignores the case of strings. However, all overloads of the `CollectionsUtil.CreateCaseInsensitiveHashTable` method are culture-sensitive because they use the `Thread.CurrentCulture` property. You cannot create a culture-insensitive `Hashtable` using this method. To create a culture-insensitive `Hashtable`, use the `Hashtable` constructor that accepts a `culture` parameter. For the `culture` parameter, specify `CultureInfo.InvariantCulture`. The following code example demonstrates the constructor for a culture-insensitive `Hashtable`.

```
internalHashtable = New Hashtable(New
    CaseInsensitiveHashCodeProvider(CultureInfo.InvariantCulture),
    New CaseInsensitiveComparer(CultureInfo.InvariantCulture))
```

```
internalHashtable = new Hashtable(new CaseInsensitiveHashCodeProvider
    (CultureInfo.InvariantCulture),
    new CaseInsensitiveComparer(CultureInfo.InvariantCulture));
```

Use the `SortedList` class

A `SortedList` represents a collection of key-and-value pairs that are sorted by the keys and are accessible by key and by index. When you use a `SortedList` where strings are the keys, the sorting and lookup can be affected by the `Thread.CurrentCulture` property. To obtain culture-insensitive behavior from a `SortedList`, create a `SortedList` using one of the constructors that accepts a `comparer` parameter. The `comparer` parameter specifies the `IComparer` implementation to use when comparing keys. For the parameter, specify a custom comparer class that uses `CultureInfo.InvariantCulture` to compare keys. The following example illustrates a custom culture-insensitive comparer class that you can specify as the `comparer` parameter to a `SortedList` constructor.

```

Imports System.Collections
Imports System.Globalization

Friend Class InvariantComparer
    Implements IComparer
    Private m_compareInfo As CompareInfo
    Friend Shared [Default] As New InvariantComparer()

    Friend Sub New()
        m_compareInfo = CultureInfo.InvariantCulture.CompareInfo
    End Sub

    Public Function Compare(a As Object, b As Object) As Integer _
        Implements IComparer.Compare
        Dim sa As String = CType(a, String)
        Dim sb As String = CType(b, String)
        If Not (sa Is Nothing) And Not (sb Is Nothing) Then
            Return m_compareInfo.Compare(sa, sb)
        Else
            Return Comparer.Default.Compare(a, b)
        End If
    End Function
End Class

```

```

using System;
using System.Collections;
using System.Globalization;

internal class InvariantComparer : IComparer
{
    private CompareInfo _compareInfo;
    internal static readonly InvariantComparer Default = new
        InvariantComparer();

    internal InvariantComparer()
    {
        _compareInfo = CultureInfo.InvariantCulture.CompareInfo;
    }

    public int Compare(Object a, Object b)
    {
        if (a is string sa && b is string sb)
            return _compareInfo.Compare(sa, sb);
        else
            return Comparer.Default.Compare(a,b);
    }
}

```

In general, if you use a `SortedList` on strings without specifying a custom invariant comparer, a change to `Thread.CurrentCulture` after the list has been populated can invalidate the list.

Use the `ArrayList.Sort` method

Overloads of the `ArrayList.Sort` method perform culture-sensitive sorts by default using the `Thread.CurrentCulture` property. Results can vary by culture due to different sort orders. To eliminate culture-sensitive behavior, use the overloads of this method that accept an `IComparer` implementation. For the `comparer` parameter, specify a custom invariant comparer class that uses `CultureInfo.InvariantCulture`. An example of a custom invariant comparer class is provided in the [Using the SortedList Class](#) topic.

See also

- [CaseInsensitiveComparer](#)
- [CaseInsensitiveHashCodeProvider](#)
- [ArrayList.Sort](#)
- [SortedList](#)
- [Hashtable](#)
- [IComparer](#)
- [Perform culture-insensitive string operations](#)
- [CollectionsUtil.CreateCaseInsensitiveHashtable](#)

Perform culture-insensitive string operations in arrays

9/20/2022 • 2 minutes to read • [Edit Online](#)

Overloads of the [Array.Sort](#) and [Array.BinarySearch](#) methods perform culture-sensitive sorts by default using the [Thread.CurrentCulture](#) property. Culture-sensitive results returned by these methods can vary by culture due to differences in sort orders. To eliminate culture-sensitive behavior, use one of the overloads of this method that accepts a `comparer` parameter. The `comparer` parameter specifies the [IComparer](#) implementation to use when comparing elements in the array. For the parameter, specify a custom invariant comparer class that uses [CultureInfo.InvariantCulture](#). An example of a custom invariant comparer class is provided in the "Using the [SortedList Class](#)" subtopic of the [Perform culture-insensitive string operations in collections](#) topic.

NOTE

Passing [CultureInfo.InvariantCulture](#) to a comparison method does perform a culture-insensitive comparison. However, it does not cause a non-linguistic comparison, for example, for file paths, registry keys, and environment variables. Neither does it support security decisions based on the comparison result. For a non-linguistic comparison or support for result-based security decisions, the application should use a comparison method that accepts a [StringComparison](#) value. The application should then pass [Ordinal](#).

See also

- [Array.Sort](#)
- [Array.BinarySearch](#)
- [IComparer](#)
- [Perform culture-insensitive string operations](#)

Best practices for developing world-ready applications

9/20/2022 • 4 minutes to read • [Edit Online](#)

This section describes the best practices to follow when developing world-ready applications.

Globalization best practices

1. Make your application Unicode internally.
2. Use the culture-aware classes provided by the [System.Globalization](#) namespace to manipulate and format data.
 - For sorting, use the [SortKey](#) class and the [CompareInfo](#) class.
 - For string comparisons, use the [CompareInfo](#) class.
 - For date and time formatting, use the [DateTimeFormatInfo](#) class.
 - For numeric formatting, use the [NumberFormatInfo](#) class.
 - For Gregorian and non-Gregorian calendars, use the [Calendar](#) class or one of the specific calendar implementations.
3. Use the culture property settings provided by the [System.Globalization.CultureInfo](#) class in the appropriate situations. Use the [CultureInfo.CurrentCulture](#) property for formatting tasks, such as date and time or numeric formatting. Use the [CultureInfo.CurrentUICulture](#) property to retrieve resources. Note that the `CurrentCulture` and `CurrentUICulture` properties can be set per thread.
4. Enable your application to read and write data to and from a variety of encodings by using the encoding classes in the [System.Text](#) namespace. Do not assume ASCII data. Assume that international characters will be supplied anywhere a user can enter text. For example, the application should accept international characters in server names, directories, file names, user names, and URLs.
5. When using the [UTF8Encoding](#) class, for security reasons, use the error detection feature offered by this class. To turn on the error detection feature, create an instance of the class using the constructor that takes a `throwOnInvalidBytes` parameter and set the value of this parameter to `true`.
6. Whenever possible, handle strings as entire strings instead of as a series of individual characters. This is especially important when sorting or searching for substrings. This will prevent problems associated with parsing combined characters. You can also work with units of text rather than single characters by using the [System.Globalization.StringInfo](#) class.
7. Display text using the classes provided by the [System.Drawing](#) namespace.
8. For consistency across operating systems, do not allow user settings to override [CultureInfo](#). Use the `CultureInfo` constructor that accepts a `useUserOverride` parameter and set it to `false`.
9. Test your application functionality on international operating system versions, using international data.
10. If a security decision is based on the result of a string comparison or case change operation, use a culture-insensitive string operation. This practice ensures that the result is not affected by the value of `CultureInfo.CurrentCulture`. See the "String Comparisons that Use the Current Culture" section of [Best Practices for Using Strings](#) for an example that demonstrates how culture-sensitive string comparisons

can produce inconsistent results.

Localization best practices

1. Move all localizable resources to separate resource-only DLLs. Localizable resources include user interface elements, such as strings, error messages, dialog boxes, menus, and embedded object resources.
2. Do not hardcode strings or user interface resources.
3. Do not put non-localizable resources into the resource-only DLLs. This causes confusion for translators.
4. Do not use composite strings that are built at run time from concatenated phrases. Composite strings are difficult to localize because they often assume an English grammatical order that does not apply to all languages.
5. Avoid ambiguous constructs such as "Empty Folder" where the strings can be translated differently depending on the grammatical roles of the string components. For example, "empty" can be either a verb or an adjective, which can lead to different translations in languages such as Italian or French.
6. Avoid using images and icons that contain text in your application. They are expensive to localize.
7. Allow plenty of room for the length of strings to expand in the user interface. In some languages, phrases can require 50-75 percent more space than they need in other languages.
8. Use the [System.Resources.ResourceManager](#) class to retrieve resources based on culture.
9. Use [Visual Studio](#) to create Windows Forms dialog boxes so they can be localized using the [Windows Forms Resource Editor \(Winres.exe\)](#). Do not code Windows Forms dialog boxes by hand.
10. Arrange for professional localization (translation).
11. For a complete description of creating and localizing resources, see [Resources in .NET apps](#).

Globalization best practices for ASP.NET applications

1. Explicitly set the [CurrentUICulture](#) and [CurrentCulture](#) properties in your application. Do not rely on defaults.
2. Note that ASP.NET applications are managed applications and therefore can use the same classes as other managed applications for retrieving, displaying, and manipulating information based on culture.
3. Be aware that you can specify the following three types of encodings in ASP.NET:
 - `requestEncoding` specifies the encoding received from the client's browser.
 - `responseEncoding` specifies the encoding to send to the client browser. In most situations, this encoding should be the same as that specified for `requestEncoding`.
 - `fileEncoding` specifies the default encoding for `.aspx`, `.asmx`, and `.asax` file parsing.
4. Specify the values for the `requestEncoding`, `responseEncoding`, `fileEncoding`, `culture`, and `uiCulture` attributes in the following three places in an ASP.NET application:
 - In the globalization section of a `Web.config` file. This file is external to the ASP.NET application. For more information, see [`<globalization>` element](#).
 - In a page directive. Note that, when an application is in a page, the file has already been read. Therefore, it is too late to specify `fileEncoding` and `requestEncoding`. Only `uiCulture`, `Culture`, and `responseEncoding` can be specified in a page directive.

- Programmatically in application code. This setting can vary per request. As with a page directive, by the time the application's code is reached it is too late to specify fileEncoding and requestEncoding. Only uiCulture, Culture, and responseEncoding can be specified in application code.
5. Note that the uiCulture value can be set to the browser accept language.

See also

- [Globalization and Localization](#)
- [Resources in .NET apps](#)

Resources in .NET apps

9/20/2022 • 3 minutes to read • [Edit Online](#)

Nearly every production-quality app has to use resources. A resource is any non-executable data that is logically deployed with an app. A resource might be displayed in an app as error messages or as part of the user interface. Resources can contain data in a number of forms, including strings, images, and persisted objects. (To write persisted objects to a resource file, the objects must be serializable.) Storing your data in a resource file enables you to change the data without recompiling your entire app. It also enables you to store data in a single location, and eliminates the need to rely on hard-coded data that is stored in multiple locations.

.NET provides comprehensive support for the creation and [localization](#) of resources. In addition, .NET supports a simple model for packaging and deploying localized resources.

Create and localize resources

In a non-localized app, you can use resource files as a repository for app data, particularly for strings that might otherwise be hard-coded in multiple locations in source code. Most commonly, you create resources as either text (.txt) or XML (.resx) files, and use [Resgen.exe \(Resource File Generator\)](#) to compile them into binary .resources files. These files can then be embedded in the app's executable file by a language compiler. For more information about creating resources, see [Create resource files](#).

You can also localize your app's resources for specific cultures. This enables you to build localized (translated) versions of your apps. When you develop an app that uses localized resources, you designate a culture that serves as the neutral or fallback culture whose resources are used if no suitable resources are available. Typically, the resources of the neutral culture are stored in the app's executable. The remaining resources for individual localized cultures are stored in standalone satellite assemblies. For more information, see [Create satellite assemblies](#).

Package and deploy resources

You deploy localized app resources in [satellite assemblies](#). A satellite assembly contains the resources of a single culture; it does not contain any app code. In the satellite assembly deployment model, you create an app with one default assembly (which is typically the main assembly) and one satellite assembly for each culture that the app supports. Because the satellite assemblies are not part of the main assembly, you can easily replace or update resources corresponding to a specific culture without replacing the app's main assembly.

Carefully determine which resources will make up your app's default resource assembly. Because it is a part of the main assembly, any changes to it will require you to replace the main assembly. If you do not provide a default resource, an exception will be thrown when the [resource fallback process](#) attempts to find it. In a well-designed app, using resources should never throw an exception.

For more information, see the [Packaging and Deploying Resources](#) article.

Retrieve resources

At run time, an app loads the appropriate localized resources on a per-thread basis, based on the culture specified by the [CultureInfo.CurrentCulture](#) property. This property value is derived as follows:

- By directly assigning a [CultureInfo](#) object that represents the localized culture to the [Thread.CurrentCulture](#) property.
- If a culture is not explicitly assigned, by retrieving the default thread UI culture from the

`CultureInfo.DefaultThreadCurrentUICulture` property.

- If a default thread UI culture is not explicitly assigned, by retrieving the culture for the current user on the local computer. .NET implementations running on Windows do this by calling the Windows `GetUserDefaultUILanguage` function.

For more information about how the current UI culture is set, see the [CultureInfo](#) and [CultureInfo.CurrentUICulture](#) reference pages.

You can then retrieve resources for the current UI culture or for a specific culture by using the [System.Resources.ResourceManager](#) class. Although the [ResourceManager](#) class is most commonly used for retrieving resources, the [System.Resources](#) namespace contains additional types that you can use to retrieve resources. These include:

- The [ResourceReader](#) class, which enables you to enumerate resources embedded in an assembly or stored in a standalone binary .resources file. It is useful when you don't know the precise names of the resources that are available at run time.
- The [ResXResourceReader](#) class, which enables you to retrieve resources from an XML (.resx) file.
- The [ResourceSet](#) class, which enables you to retrieve the resources of a specific culture without observing fallback rules. The resources can be stored in an assembly or a standalone binary .resources file. You can also develop an [IResourceReader](#) implementation that enables you to use the [ResourceSet](#) class to retrieve resources from some other source.
- The [ResXResourceSet](#) class, which enables you to retrieve all the items in an XML resource file into memory.

See also

- [CultureInfo](#)
- [CultureInfo.CurrentUICulture](#)
- [Create resource files](#)
- [Package and deploy resources](#)
- [Create satellite assemblies](#)
- [Retrieve resources](#)
- [Localization in .NET](#)

Create resource files for .NET apps

9/20/2022 • 12 minutes to read • [Edit Online](#)

You can include resources, such as strings, images, or object data, in resources files to make them easily available to your application. The .NET Framework offers five ways to create resources files:

- Create a text file that contains string resources. You can use [Resource File Generator \(*resgen.exe*\)](#) to convert the text file into a binary resource (.resources) file. You can then embed the binary resource file in an application executable or an application library by using a language compiler, or you can embed it in a satellite assembly by using [Assembly Linker \(Al.exe\)](#). For more information, see the [Resources in text files](#) section.
- Create an XML resource (.resx) file that contains string, image, or object data. You can use [Resource File Generator \(*resgen.exe*\)](#) to convert the .resx file into a binary resource (.resources) file. You can then embed the binary resource file in an application executable or an application library by using a language compiler, or you can embed it in a satellite assembly by using [Assembly Linker \(Al.exe\)](#). For more information, see the [Resources in .resx Files](#) section.
- Create an XML resource (.resx) file programmatically by using types in the [System.Resources](#) namespace. You can create a .resx file, enumerate its resources, and retrieve specific resources by name. For more information, see [Working with .resx Files Programmatically](#).
- Create a binary resource (.resources) file programmatically. You can then embed the file in an application executable or an application library by using a language compiler, or you can embed it in a satellite assembly by using [Assembly Linker \(Al.exe\)](#). For more information, see the [Resources in .resources Files](#) section.
- Use [Visual Studio](#) to create a resource file and include it in your project. Visual Studio provides a resource editor that lets you add, delete, and modify resources. At compile time, the resource file is automatically converted to a binary .resources file and embedded in an application assembly or satellite assembly. For more information, see the [Resource files in Visual Studio](#) section.

Resources in text files

You can use text (.txt or .restext) files to store string resources only. For non-string resources, use .resx files or create them programmatically. Text files that contain string resources have the following format:

```

# This is an optional comment.
name = value

; This is another optional comment.
name = value

; The following supports conditional compilation if X is defined.
#ifndef X
name1=value1
name2=value2
#endif

# The following supports conditional compilation if Y is undefined.
#if !Y
name1=value1
name2=value2
#endif

```

The resource file format of .txt and .restext files is identical. The .restext file extension merely serves to make text files immediately identifiable as text-based resource files.

String resources appear as *name/value* pairs, where *name* is a string that identifies the resource, and *value* is the resource string that is returned when you pass *name* to a resource retrieval method such as [ResourceManager.GetString](#). *name* and *value* must be separated by an equal sign (=). For example:

```

FileMenuName=File
EditMenuName>Edit
ViewMenuName=View
HelpMenuName=Help

```

Caution

Do not use resource files to store passwords, security-sensitive information, or private data.

Empty strings (that is, a resource whose value is [String.Empty](#)) are permitted in text files. For example:

```
EmptyString=
```

Starting with .NET Framework 4.5 and in all versions of .NET Core, text files support conditional compilation with the `#ifdef symbol...` `#endif` and `#if ! symbol...` `#endif` constructs. You can then use the `/define` switch with [Resource File Generator \(resgen.exe\)](#) to define symbols. Each resource requires its own `#ifdef symbol...` `#endif` or `#if ! symbol...` `#endif` construct. If you use an `#ifdef` statement and *symbol* is defined, the associated resource is included in the .resources file; otherwise, it is not included. If you use an `#if !` statement and *symbol* is not defined, the associated resource is included in the .resources file; otherwise, it is not included.

Comments are optional in text files and are preceded either by a semicolon (;) or by a pound sign (#) at the beginning of a line. Lines that contain comments can be placed anywhere in the file. Comments are not included in a compiled .resources file that is created by using [Resource File Generator \(resgen.exe\)](#).

Any blank lines in the text files are considered to be white space and are ignored.

The following example defines two string resources named `OKButton` and `CancelButton`.

```

#define resources for buttons in the user interface.
OKButton=OK
CancelButton=Cancel

```

If the text file contains duplicate occurrences of *name*, [Resource File Generator \(resgen.exe\)](#) displays a warning

and ignores the second name.

value cannot contain new line characters, but you can use C language-style escape characters such as `\n` to represent a new line and `\t` to represent a tab. You can also include a backslash character if it is escaped (for example, "`\\`"). In addition, an empty string is permitted.

Save resources in text file format by using UTF-8 encoding or UTF-16 encoding in either little-endian or big-endian byte order. However, [Resource File Generator \(*resgen.exe*\)](#), which converts a .txt file to a .resources file, treats files as UTF-8 by default. If you want Resgen.exe to recognize a file that was encoded using UTF-16, you must include a Unicode byte order mark (U+FEFF) at the beginning of the file.

To embed a resource file in text format into a .NET assembly, you must convert the file to a binary resource (.resources) file by using [Resource File Generator \(*resgen.exe*\)](#). You can then embed the .resources file in a .NET assembly by using a language compiler or embed it in a satellite assembly by using [Assembly Linker \(Al.exe\)](#).

The following example uses a resource file in text format named GreetingResources.txt for a simple "Hello World" console application. The text file defines two strings, `prompt` and `greeting`, that prompt the user to enter their name and display a greeting.

```
# GreetingResources.txt
# A resource file in text format for a "Hello World" application.
#
# Initial prompt to the user.
prompt=Enter your name:
# Format string to display the result.
greeting=Hello, {0}!
```

The text file is converted to a .resources file by using the following command:

```
resgen GreetingResources.txt
```

The following example shows the source code for a console application that uses the .resources file to display messages to the user.

```
using System;
using System.Reflection;
using System.Resources;

public class Example
{
    public static void Main()
    {
        ResourceManager rm = new ResourceManager("GreetingResources",
                                                typeof(Example).Assembly);
        Console.WriteLine(rm.GetString("prompt"));
        string name = Console.ReadLine();
        Console.WriteLine(rm.GetString("greeting"), name);
    }
}
// The example displays output like the following:
//      Enter your name: Wilberforce
//      Hello, Wilberforce!
```

```

Imports System.Reflection
Imports System.Resources

Module Example
    Public Sub Main()
        Dim rm As New ResourceManager("GreetingResources",
                                      GetType(Example).Assembly())
        Console.WriteLine(rm.GetString("prompt"))
        Dim name As String = Console.ReadLine()
        Console.WriteLine(rm.GetString("greeting"), name)
    End Sub
End Module
' The example displays output like the following:
'   Enter your name: Wilberforce
'   Hello, Wilberforce!

```

If you are using Visual Basic, and the source code file is named Greeting.vb, the following command creates an executable file that includes the embedded .resources file:

```
vbc greeting.vb -resource:GreetingResources.resources
```

If you are using C#, and the source code file is named Greeting.cs, the following command creates an executable file that includes the embedded .resources file:

```
csc greeting.cs -resource:GreetingResources.resources
```

Resources in .resx files

Unlike text files, which can only store string resources, XML resource (.resx) files can store strings, binary data such as images, icons, and audio clips, and programmatic objects. A .resx file contains a standard header, which describes the format of the resource entries and specifies the versioning information for the XML that is used to parse the data. The resource file data follows the XML header. Each data item consists of a name/value pair that is contained in a `data` tag. Its `name` attribute defines the resource name, and the nested `value` tag contains the resource value. For string data, the `value` tag contains the string.

For example, the following `data` tag defines a string resource named `prompt` whose value is "Enter your name".

```

<data name="prompt" xml:space="preserve">
  <value>Enter your name:</value>
</data>

```

WARNING

Do not use resource files to store passwords, security-sensitive information, or private data.

For resource objects, the `data` tag includes a `type` attribute that indicates the data type of the resource. For objects that consist of binary data, the `data` tag also includes a `mimetype` attribute, which indicates the `base64` type of the binary data.

NOTE

All .resx files use a binary serialization formatter to generate and parse the binary data for a specified type. As a result, a .resx file can become invalid if the binary serialization format for an object changes in an incompatible way.

The following example shows a portion of a .resx file that includes an [Int32](#) resource and a bitmap image.

```
<data name="i1" type="System.Int32, mscorel">
  <value>20</value>
</data>

<data name="flag" type="System.Drawing.Bitmap, System.Drawing,
  Version=1.0.5000.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
  mimetype="application/x-microsoft.net.object.bytearray.base64">
  <value>
    AAEAAAD/////AQAAAAAAAAMAgAAADtTeX...
  </value>
</data>
```

IMPORTANT

Because .resx files must consist of well-formed XML in a predefined format, we do not recommend working with .resx files manually, particularly when the .resx files contain resources other than strings. Instead, [Visual Studio](#) provides a transparent interface for creating and manipulating .resx files. For more information, see the [Resource files in Visual Studio](#) section. You can also create and manipulate .resx files programmatically. For more information, see [Work with .resx files programmatically](#).

Resources in .resources files

You can use the [System.Resources.ResourceWriter](#) class to programmatically create a binary resource (.resources) file directly from code. You can also use [Resource File Generator \(resgen.exe\)](#) to create a .resources file from a text file or a .resx file. The .resources file can contain binary data (byte arrays) and object data in addition to string data. Programmatically creating a .resources file requires the following steps:

1. Create a [ResourceWriter](#) object with a unique file name. You can do this by specifying either a file name or a file stream to a [ResourceWriter](#) class constructor.
2. Call one of the overloads of the [ResourceWriter.AddResource](#) method for each named resource to add to the file. The resource can be a string, an object, or a collection of binary data (a byte array).
3. Call the [ResourceWriter.Close](#) method to write the resources to the file and to close the [ResourceWriter](#) object.

NOTE

Do not use resource files to store passwords, security-sensitive information, or private data.

The following example programmatically creates a .resources file named CarResources.resources that stores six strings, an icon, and two application-defined objects (two [Automobile](#) objects). The [Automobile](#) class, which is defined and instantiated in the example, is tagged with the [SerializableAttribute](#) attribute, which allows it to be persisted by the binary serialization formatter.

```
using System;
using System.Drawing;
```

```
using System.Resources;

[Serializable()] public class Automobile
{
    private string carMake;
    private string carModel;
    private int carYear;
    private int carDoors;
    private int carCylinders;

    public Automobile(string make, string model, int year) :
        this(make, model, year, 0, 0)
    { }

    public Automobile(string make, string model, int year,
                      int doors, int cylinders)
    {
        this.carMake = make;
        this.carModel = model;
        this.carYear = year;
        this.carDoors = doors;
        this.carCylinders = cylinders;
    }

    public string Make {
        get { return this.carMake; }
    }

    public string Model {
        get { return this.carModel; }
    }

    public int Year {
        get { return this.carYear; }
    }

    public int Doors {
        get {
            return this.carDoors; }
    }

    public int Cylinders {
        get {
            return this.carCylinders; }
    }
}

public class Example
{
    public static void Main()
    {
        // Instantiate an Automobile object.
        Automobile car1 = new Automobile("Ford", "Model N", 1906, 0, 4);
        Automobile car2 = new Automobile("Ford", "Model T", 1909, 2, 4);
        // Define a resource file named CarResources.resx.
        using (ResourceWriter rw = new ResourceWriter(@"\CarResources.resources"))
        {
            rw.AddResource("Title", "Classic American Cars");
            rw.AddResource("HeaderString1", "Make");
            rw.AddResource("HeaderString2", "Model");
            rw.AddResource("HeaderString3", "Year");
            rw.AddResource("HeaderString4", "Doors");
            rw.AddResource("HeaderString5", "Cylinders");
            rw.AddResource("Information", SystemIcons.Information);
            rw.AddResource("EarlyAuto1", car1);
            rw.AddResource("EarlyAuto2", car2);
        }
    }
}
```

```

Imports System.Drawing
Imports System.Resources

<Serializable()> Public Class Automobile
    Private carMake As String
    Private carModel As String
    Private carYear As Integer
    Private carDoors AS Integer
    Private carCylinders As Integer

    Public Sub New(make As String, model As String, year As Integer)
        Me.New(make, model, year, 0, 0)
    End Sub

    Public Sub New(make As String, model As String, year As Integer,
                  doors As Integer, cylinders As Integer)
        Me.carMake = make
        Me.carModel = model
        Me.carYear = year
        Me.carDoors = doors
        Me.carCylinders = cylinders
    End Sub

    Public ReadOnly Property Make As String
        Get
            Return Me.carMake
        End Get
    End Property

    Public ReadOnly Property Model As String
        Get
            Return Me.carModel
        End Get
    End Property

    Public ReadOnly Property Year As Integer
        Get
            Return Me.carYear
        End Get
    End Property

    Public ReadOnly Property Doors As Integer
        Get
            Return Me.carDoors
        End Get
    End Property

    Public ReadOnly Property Cylinders As Integer
        Get
            Return Me.carCylinders
        End Get
    End Property
End Class

Module Example
    Public Sub Main()
        ' Instantiate an Automobile object.
        Dim car1 As New Automobile("Ford", "Model N", 1906, 0, 4)
        Dim car2 As New Automobile("Ford", "Model T", 1909, 2, 4)
        ' Define a resource file named CarResources.resx.
        Using rw As New ResourceWriter(".\CarResources.resources")
            rw.AddResource("Title", "Classic American Cars")
            rw.AddResource("HeaderString1", "Make")
            rw.AddResource("HeaderString2", "Model")
            rw.AddResource("HeaderString3", "Year")
            rw.AddResource("HeaderString4", "Doors")
        End Using
    End Sub
End Module

```

```
rw.AddResource("HeaderString5", "Cylinders")
rw.AddResource("Information", SystemIcons.Information)
rw.AddResource("EarlyAuto1", car1)
rw.AddResource("EarlyAuto2", car2)
End Using
End Sub
End Module
```

After you create the .resources file, you can embed it in a run-time executable or library by including the language compiler's `/resource` switch, or embed it in a satellite assembly by using [Assembly Linker \(Al.exe\)](#).

Resource files in Visual Studio

When you add a resource file to your [Visual Studio](#) project, Visual Studio creates a .resx file in the project directory. Visual Studio provides resource editors that enable you to add strings, images, and binary objects. Because the editors are designed to handle static data only, they cannot be used to store programmatic objects; you must write object data to either a .resx file or to a .resources file programmatically. For more information, see [Work with .resx files programmatically](#) and the [Resources in .resources files](#) section.

If you're adding localized resources, give them the same root file name as the main resource file. You should also designate their culture in the file name. For example, if you add a resource file named *Resources.resx*, you might also create resource files named *Resources.en-US.resx* and *Resources.fr-FR.resx* to hold localized resources for the English (United States) and French (France) cultures, respectively. You should also designate your application's default culture. This is the culture whose resources are used if no localized resources for a particular culture can be found.

To specify the default culture, in **Solution Explorer** in Visual Studio:

- Open the project properties, right-click the project and select **Properties** (or Alt + Enter when project is selected).
- Select the **Package** tab.
- In the **General** area, select the appropriate language/culture from the **Assembly neutral language** control.
- Save your changes.

At compile time, Visual Studio first converts the .resx files in a project to binary resource (.resources) files and stores them in a subdirectory of the project's *obj* directory. Visual Studio embeds any resource files that do not contain localized resources in the main assembly that is generated by the project. If any resource files contain localized resources, Visual Studio embeds them in separate satellite assemblies for each localized culture. It then stores each satellite assembly in a directory whose name corresponds to the localized culture. For example, localized English (United States) resources are stored in a satellite assembly in the en-US subdirectory.

See also

- [System.Resources](#)
- [Resources in .NET Apps](#)
- [Package and deploy resources](#)

Work with .resx files programmatically

9/20/2022 • 10 minutes to read • [Edit Online](#)

NOTE

This article applies to .NET Framework. For information that applies to .NET 5+ (including .NET Core), see [Resources in .resx files](#).

Because XML resource (.resx) files must consist of well-defined XML, including a header that must follow a specific schema followed by data in name/value pairs, you may find that creating these files manually is error-prone. As an alternative, you can create .resx files programmatically by using types and members in the .NET Class Library. You can also use the .NET Class Library to retrieve resources that are stored in .resx files. This article explains how you can use the types and members in the [System.Resources](#) namespace to work with .resx files.

This article discusses working with XML (.resx) files that contain resources. For information on working with binary resource files that have been embedded in assemblies, see [ResourceManager](#).

WARNING

There are also ways to work with .resx files other than programmatically. When you add a resource file to a [Visual Studio](#) project, Visual Studio provides an interface for creating and maintaining a .resx file, and automatically converts the .resx file to a .resources file at compile time. You can also use a text editor to manipulate a .resx file directly. However, to avoid corrupting the file, be careful not to modify any binary information that is stored in the file.

Create a .resx file

You can use the [System.Resources.ResXResourceWriter](#) class to create a .resx file programmatically, by following these steps:

1. Instantiate a [ResXResourceWriter](#) object by calling the [ResXResourceWriter\(String\)](#) method and supplying the name of the .resx file. The file name must include the .resx extension. If you instantiate the [ResXResourceWriter](#) object in a `using` block, you do not explicitly have to call the [ResXResourceWriter.Close](#) method in step 3.
2. Call the [ResXResourceWriter.AddResource](#) method for each resource you want to add to the file. Use the overloads of this method to add string, object, and binary (byte array) data. If the resource is an object, it must be serializable.
3. Call the [ResXResourceWriter.Close](#) method to generate the resource file and to release all resources. If the [ResXResourceWriter](#) object was created within a `using` block, resources are written to the .resx file and the resources used by the [ResXResourceWriter](#) object are released at the end of the `using` block.

The resulting .resx file has the appropriate header and a `data` tag for each resource added by the [ResXResourceWriter.AddResource](#) method.

WARNING

Do not use resource files to store passwords, security-sensitive information, or private data.

The following example creates a .resx file named CarResources.resx that stores six strings, an icon, and two application-defined objects (two `Automobile` objects). The `Automobile` class, which is defined and instantiated in the example, is tagged with the `SerializableAttribute` attribute.

```
using System;
using System.Drawing;
using System.Resources;

[Serializable()] public class Automobile
{
    private string carMake;
    private string carModel;
    private int carYear;
    private int carDoors;
    private int carCylinders;

    public Automobile(string make, string model, int year) :
        this(make, model, year, 0, 0)
    {
    }

    public Automobile(string make, string model, int year,
                      int doors, int cylinders)
    {
        this.carMake = make;
        this.carModel = model;
        this.carYear = year;
        this.carDoors = doors;
        this.carCylinders = cylinders;
    }

    public string Make {
        get { return this.carMake; }
    }

    public string Model {
        get { return this.carModel; }
    }

    public int Year {
        get { return this.carYear; }
    }

    public int Doors {
        get { return this.carDoors; }
    }

    public int Cylinders {
        get { return this.carCylinders; }
    }
}

public class Example
{
    public static void Main()
    {
        // Instantiate an Automobile object.
        Automobile car1 = new Automobile("Ford", "Model N", 1906, 0, 4);
        Automobile car2 = new Automobile("Ford", "Model T", 1909, 2, 4);
        // Define a resource file named CarResources.resx.
        using (ResXResourceWriter resx = new ResXResourceWriter(@".\CarResources.resx"))
        {
            resx.AddResource("Title", "Classic American Cars");
            resx.AddResource("HeaderString1", "Make");
            resx.AddResource("HeaderString2", "Model");
            resx.AddResource("HeaderString3", "Year");
            resx.AddResource("HeaderString4", "Doors");
            resx.AddResource("HeaderString5", "Cylinders");
        }
    }
}
```

```

        resx.AddResource("Information", SystemIcons.Information);
        resx.AddResource("EarlyAuto1", car1);
        resx.AddResource("EarlyAuto2", car2);
    }
}
}

```

```

Imports System.Drawing
Imports System.Resources

<Serializable()> Public Class Automobile
    Private carMake As String
    Private carModel As String
    Private carYear As Integer
    Private carDoors AS Integer
    Private carCylinders As Integer

    Public Sub New(make As String, model As String, year As Integer)
        Me.New(make, model, year, 0, 0)
    End Sub

    Public Sub New(make As String, model As String, year As Integer,
                  doors As Integer, cylinders As Integer)
        Me.carMake = make
        Me.carModel = model
        Me.carYear = year
        Me.carDoors = doors
        Me.carCylinders = cylinders
    End Sub

    Public ReadOnly Property Make As String
        Get
            Return Me.carMake
        End Get
    End Property

    Public ReadOnly Property Model As String
        Get
            Return Me.carModel
        End Get
    End Property

    Public ReadOnly Property Year As Integer
        Get
            Return Me.carYear
        End Get
    End Property

    Public ReadOnly Property Doors As Integer
        Get
            Return Me.carDoors
        End Get
    End Property

    Public ReadOnly Property Cylinders As Integer
        Get
            Return Me.carCylinders
        End Get
    End Property
End Class

Module Example
    Public Sub Main()
        ' Instantiate an Automobile object.
        Dim car1 As New Automobile("Ford", "Model N", 1906, 0, 4)
        Dim car2 As New Automobile("Ford", "Model T", 1909, 2, 4)
        ' Define a resource file named CarResources.resx.
    End Sub

```

```

Using resx As New ResXResourceWriter(".\CarResources.resx")
    resx.AddResource("Title", "Classic American Cars")
    resx.AddResource("HeaderString1", "Make")
    resx.AddResource("HeaderString2", "Model")
    resx.AddResource("HeaderString3", "Year")
    resx.AddResource("HeaderString4", "Doors")
    resx.AddResource("HeaderString5", "Cylinders")
    resx.AddResource("Information", SystemIcons.Information)
    resx.AddResource("EarlyAuto1", car1)
    resx.AddResource("EarlyAuto2", car2)
End Using
End Sub
End Module

```

TIP

You can also use [Visual Studio](#) to create .resx files. At compile time, Visual Studio uses the [Resource File Generator \(Resgen.exe\)](#) to convert the .resx file to a binary resource (.resources) file, and also embeds it in either an application assembly or a satellite assembly.

You cannot embed a .resx file in a runtime executable or compile it into a satellite assembly. You must convert your .resx file into a binary resource (.resources) file by using the [Resource File Generator \(Resgen.exe\)](#). The resulting .resources file can then be embedded in an application assembly or a satellite assembly. For more information, see [Create resource files](#).

Enumerate resources

In some cases, you may want to retrieve all resources, instead of a specific resource, from a .resx file. To do this, you can use the [System.Resources.ResXResourceReader](#) class, which provides an enumerator for all resources in the .resx file. The [System.Resources.ResXResourceReader](#) class implements [IDictionaryEnumerator](#), which returns a [DictionaryEntry](#) object that represents a particular resource for each iteration of the loop. Its [DictionaryEntry.Key](#) property returns the resource's key, and its [DictionaryEntry.Value](#) property returns the resource's value.

The following example creates a [ResXResourceReader](#) object for the CarResources.resx file created in the previous example and iterates through the resource file. It adds the two [Automobile](#) objects that are defined in the resource file to a [System.Collections.Generic.List<T>](#) object, and it adds five of the six strings to a [SortedList](#) object. The values in the [SortedList](#) object are converted to a parameter array, which is used to display column headings to the console. The [Automobile](#) property values are also displayed to the console.

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Resources;

public class Example
{
    public static void Main()
    {
        string resxFile = @"\CarResources.resx";
        List<Automobile> autos = new List<Automobile>();
        SortedList headers = new SortedList();

        using (ResXResourceReader resxReader = new ResXResourceReader(resxFile))
        {
            foreach (DictionaryEntry entry in resxReader)
            {
                if (((string) entry.Key).StartsWith("EarlyAuto"))
                    autos.Add((Automobile) entry.Value);
                else if (((string) entry.Key).StartsWith("Header"))
                    headers.Add((string) entry.Key, (string) entry.Value);
            }
        }

        string[] headerColumns = new string[headers.Count];
        headers.GetValueList().CopyTo(headerColumns, 0);
        Console.WriteLine("{0,-8} {1,-10} {2,-4} {3,-5} {4,-9}\n",
                           headerColumns);

        foreach (var auto in autos)
            Console.WriteLine("{0,-8} {1,-10} {2,4} {3,5} {4,9}",
                            auto.Make, auto.Model, auto.Year,
                            auto.Doors, auto.Cylinders);
    }
}

// The example displays the following output:
//      Make      Model      Year    Doors    Cylinders
//
//      Ford      Model N    1906      0        4
//      Ford      Model T    1909      2        4
```

```

Imports System.Collections
Imports System.Collections.Generic
Imports System.Resources

Module Example
    Public Sub Main()
        Dim resxFile As String = ".\CarResources.resx"
        Dim autos As New List(Of Automobile)
        Dim headers As New SortedList()

        Using resxReader As New ResXResourceReader(resxFile)
            For Each entry As DictionaryEntry In resxReader
                If CType(entry.Key, String).StartsWith("EarlyAuto") Then
                    autos.Add(CType(entry.Value, Automobile))
                Else If CType(entry.Key, String).StartsWith("Header") Then
                    headers.Add(CType(entry.Key, String), CType(entry.Value, String))
                End If
            Next
        End Using
        Dim headerColumns(headers.Count - 1) As String
        headers.GetValueList().CopyTo(headerColumns, 0)
        Console.WriteLine("{0,-8} {1,-10} {2,-4} {3,-5} {4,-9}",
                           headerColumns)
        Console.WriteLine()
        For Each auto In autos
            Console.WriteLine("{0,-8} {1,-10} {2,4} {3,5} {4,9}",
                           auto.Make, auto.Model, auto.Year,
                           auto.Doors, auto.Cylinders)
        Next
    End Sub
End Module
' The example displays the following output:
'     Make      Model      Year    Doors    Cylinders
'
'     Ford      Model N    1906      0        4
'     Ford      Model T    1909      2        4

```

Retrieve a specific resource

In addition to enumerating the items in a .resx file, you can retrieve a specific resource by name by using the [System.Resources.ResXResourceSet](#) class. The [ResourceSet.GetString\(String\)](#) method retrieves the value of a named string resource. The [ResourceSet.GetObject\(String\)](#) method retrieves the value of a named object or binary data. The method returns an object that must then be cast (in C#) or converted (in Visual Basic) to an object of the appropriate type.

The following example retrieves a form's caption string and icon by their resource names. It also retrieves the application-defined `Automobile` objects used in the previous example and displays them in a [DataGridView](#) control.

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Resources;
using System.Windows.Forms;

public class CarDisplayApp : Form
{
    private const string resxFile = @"\CarResources.resx";
    Automobile[] cars;

    public static void Main()
    {
        CarDisplayApp app = new CarDisplayApp();
        Application.Run(app);
    }

    public CarDisplayApp()
    {
        // Instantiate controls.
        PictureBox pictureBox = new PictureBox();
        pictureBox.Location = new Point(10, 10);
        this.Controls.Add(pictureBox);
        DataGridView grid = new DataGridView();
        grid.Location = new Point(10, 60);
        this.Controls.Add(grid);

        // Get resources from .resx file.
        using (ResXResourceSet resxSet = new ResXResourceSet(resxFile))
        {
            // Retrieve the string resource for the title.
            this.Text = resxSet.GetString("Title");
            // Retrieve the image.
            Icon image = (Icon) resxSet.GetObject("Information", true);
            if (image != null)
                pictureBox.Image = image.ToBitmap();

            // Retrieve Automobile objects.
            List<Automobile> carList = new List<Automobile>();
            string resName = "EarlyAuto";
            Automobile auto;
            int ctr = 1;
            do {
                auto = (Automobile) resxSet.GetObject(resName + ctr.ToString());
                ctr++;
                if (auto != null)
                    carList.Add(auto);
            } while (auto != null);
            cars = carList.ToArray();
            grid.DataSource = cars;
        }
    }
}
```

```

Imports System.Collections.Generic
Imports System.Drawing
Imports System.Resources
Imports System.Windows.Forms

Public Class CarDisplayApp : Inherits Form
    Private Const resxFile As String = ".\CarResources.resx"
    Dim cars() As Automobile

    Public Shared Sub Main()
        Dim app As New CarDisplayApp()
        Application.Run(app)
    End Sub

    Public Sub New()
        ' Instantiate controls.
        Dim pictureBox As New PictureBox()
        pictureBox.Location = New Point(10, 10)
        Me.Controls.Add(pictureBox)
        Dim grid As New DataGridView()
        grid.Location = New Point(10, 60)
        Me.Controls.Add(grid)

        ' Get resources from .resx file.
        Using resxSet As New ResXResourceSet(resxFile)
            ' Retrieve the string resource for the title.
            Me.Text = resxSet.GetString("Title")
            ' Retrieve the image.
            Dim image As Icon = CType(resxSet.GetObject("Information", True), Icon)
            If image IsNot Nothing Then
                pictureBox.Image = image.ToBitmap()
            End If

            ' Retrieve Automobile objects.
            Dim carList As New List(Of Automobile)
            Dim resName As String = "EarlyAuto"
            Dim auto As Automobile
            Dim ctr As Integer = 1
            Do
                auto = CType(resxSet.GetObject(resName + ctr.ToString()), Automobile)
                ctr += 1
                If auto IsNot Nothing Then carList.Add(auto)
            Loop While auto IsNot Nothing
            cars = carList.ToArray()
            grid.DataSource = cars
        End Using
    End Sub
End Class

```

Convert .resx files to binary .resources files

Converting .resx files to embedded binary resource (.resources) files has significant advantages. Although .resx files are easy to read and maintain during application development, they are rarely included with finished applications. If they are distributed with an application, they exist as separate files apart from the application executable and its accompanying libraries. In contrast, .resources files are embedded in the application executable or its accompanying assemblies. In addition, for localized applications, relying on .resx files at run time places the responsibility for handling resource fallback on the developer. In contrast, if a set of satellite assemblies that contain embedded .resources files has been created, the common language runtime handles the resource fallback process.

To convert a .resx file to a .resources file, you use [Resource File Generator \(resgen.exe\)](#), which has the following basic syntax:

```
resgen.exe .resxFilename
```

The result is a binary resource file that has the same root file name as the .resx file and a .resources file extension. This file can then be compiled into an executable or a library at compile time. If you are using the Visual Basic compiler, use the following syntax to embed a .resources file in an application's executable:

```
vbc filename .vb -resource: .resourcesFilename
```

If you are using C#, the syntax is as follows:

```
csc filename .cs -resource: .resourcesFilename
```

The *.resources* file can also be embedded in a satellite assembly by using [Assembly Linker \(al.exe\)](#), which has the following basic syntax:

```
al resourcesFilename -out: assemblyFilename
```

See also

- [Create resource files](#)
- [Resource File Generator \(resgen.exe\)](#)
- [Assembly Linker \(al.exe\)](#)

Create satellite assemblies for .NET apps

9/20/2022 • 16 minutes to read • [Edit Online](#)

Resource files play a central role in localized applications. They enable an application to display strings, images, and other data in the user's language and culture, and provide alternate data if resources for the user's language or culture are unavailable. .NET uses a hub-and-spoke model to locate and retrieve localized resources. The hub is the main assembly that contains the non-localizable executable code and the resources for a single culture, which is called the neutral or default culture. The default culture is the fallback culture for the application; it's used when no localized resources are available. You use the [NeutralResourcesLanguageAttribute](#) attribute to designate the culture of the application's default culture. Each spoke connects to a satellite assembly that contains the resources for a single localized culture but does not contain any code. Because the satellite assemblies aren't part of the main assembly, you can easily update or replace resources that correspond to a specific culture without replacing the main assembly for the application.

NOTE

The resources of an application's default culture can also be stored in a satellite assembly. To do this, you assign the [NeutralResourcesLanguageAttribute](#) attribute a value of [UltimateResourceFallbackLocation.Satellite](#).

Satellite assembly name and location

The hub-and-spoke model requires that you place resources in specific locations so that they can be easily located and used. If you don't compile and name resources as expected, or if you don't place them in the correct locations, the common language runtime won't be able to locate them and will use the resources of the default culture instead. The .NET resource manager is represented by the [ResourceManager](#) type, and it's used to automatically access localized resources. The resource manager requires the following:

- A single satellite assembly must include all the resources for a particular culture. In other words, you should compile multiple `.txt` or `.resx` files into a single binary `.resources` file.
- There must be a separate subdirectory in the application directory for each localized culture that stores that culture's resources. The subdirectory name must be the same as the culture name. Alternately, you can store your satellite assemblies in the global assembly cache. In this case, the culture information component of the assembly's strong name must indicate its culture. For more information, see [Install satellite assemblies in the Global Assembly Cache](#).

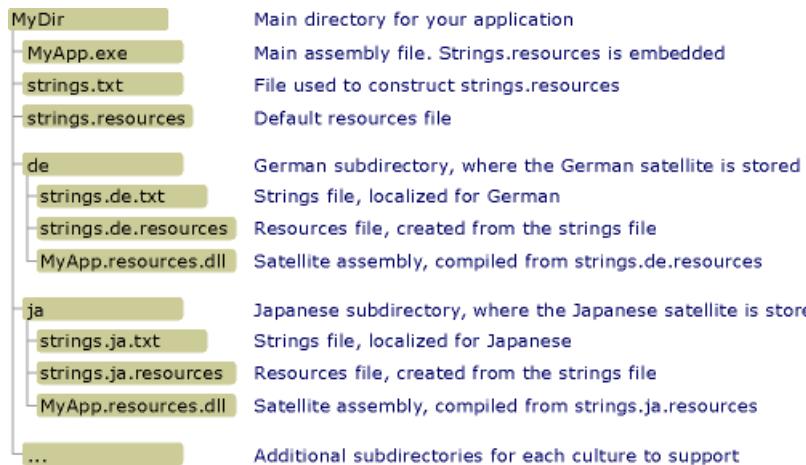
NOTE

If your application includes resources for subcultures, place each subculture in a separate subdirectory under the application directory. Do not place subcultures in subdirectories under their main culture's directory.

- The satellite assembly must have the same name as the application, and must use the file name extension `".resources.dll"`. For example, if an application is named `Example.exe`, the name of each satellite assembly should be `Example.resources.dll`. The satellite assembly name doesn't indicate the culture of its resource files. However, the satellite assembly appears in a directory that does specify the culture.
- Information about the culture of the satellite assembly must be included in the assembly's metadata. To store the culture name in the satellite assembly's metadata, you specify the `/culture` option when you use [Assembly Linker](#) to embed resources in the satellite assembly.

The following illustration shows a sample directory structure and location requirements for applications that you aren't installing in the [global assembly cache](#). The items with `.txt` and `.resources` extensions won't ship with the final application. These are the intermediate resource files used to create the final satellite resource assemblies. In this example, you could substitute `.resx` files for the `.txt` files. For more information, see [Package and deploy resources](#).

The following image shows the satellite assembly directory:



Compile satellite assemblies

You use [Resource File Generator \(`resgen.exe`\)](#) to compile text files or XML (`.resx`) files that contain resources to binary `.resources` files. You then use [Assembly Linker \(`al.exe`\)](#) to compile `.resources` files into satellite assemblies. `al.exe` creates an assembly from the `.resources` files that you specify. Satellite assemblies can contain only resources; they can't contain any executable code.

The following `al.exe` command creates a satellite assembly for the application [Example](#) from the German resources file `strings.de.resources`.

```
al -target:lib -embed:strings.de.resources -culture:de -out:Example.resources.dll
```

The following `al.exe` command also creates a satellite assembly for the application [Example](#) from the file `strings.de.resources`. The `/template` option causes the satellite assembly to inherit all assembly metadata except for its culture information from the parent assembly (`Example.dll`).

```
al -target:lib -embed:strings.de.resources -culture:de -out:Example.resources.dll -template:Example.dll
```

The following table describes the `al.exe` options used in these commands in more detail:

OPTION	DESCRIPTION
<code>-target:lib</code>	Specifies that your satellite assembly is compiled to a library (.dll) file. Because a satellite assembly doesn't contain executable code and is not an application's main assembly, you must save satellite assemblies as DLLs.
<code>-embed:strings.de.resources</code>	Specifies the name of the resource file to embed when <code>al.exe</code> compiles the assembly. You can embed multiple <code>.resources</code> files in a satellite assembly, but if you are following the hub-and-spoke model, you must compile one satellite assembly for each culture. However, you can create separate <code>.resources</code> files for strings and objects.

OPTION	DESCRIPTION
<code>-culture:de</code>	Specifies the culture of the resource to compile. The common language runtime uses this information when it searches for the resources for a specified culture. If you omit this option, <code>al.exe</code> will still compile the resource, but the runtime won't be able to find it when a user requests it.
<code>-out:Example.resources.dll</code>	Specifies the name of the output file. The name must follow the naming standard <code>baseName.resources.extension</code> , where <code>baseName</code> is the name of the main assembly and <code>extension</code> is a valid file name extension (such as <code>.dll</code>). The runtime is not able to determine the culture of a satellite assembly based on its output file name; you must use the <code>/culture</code> option to specify it.
<code>-template:Example.dll</code>	Specifies an assembly from which the satellite assembly will inherit all assembly metadata except the culture field. This option affects satellite assemblies only if you specify an assembly that has a strong name .

For a complete list of options available with `al.exe`, see [Assembly Linker \(`al.exe`\)](#).

NOTE

There may be times when you want to use the .NET Core MSBuild task to compile satellite assemblies, even though you're targeting .NET Framework. For example, you may want to use the C# compiler `deterministic` option to be able to compare assemblies from different builds. In this case, set `GenerateSatelliteAssembliesForCore` to `true` in the `.csproj` file to generate satellite assemblies using `csc.exe` instead of [Al.exe \(Assembly Linker\)](#).

```
<Project>
  <PropertyGroup>
    <GenerateSatelliteAssembliesForCore>true</GenerateSatelliteAssembliesForCore>
  </PropertyGroup>
</Project>
```

The .NET Core MSBuild task uses `csc.exe` instead of `al.exe` to generate satellite assemblies, by default. For more information, see [Make it easier to opt into "Core" satellite assembly generation](#).

Satellite assemblies example

The following is a simple "Hello world" example that displays a message box containing a localized greeting. The example includes resources for the English (United States), French (France), and Russian (Russia) cultures, and its fallback culture is English. To create the example, do the following:

1. Create a resource file named `Greeting.resx` or `Greeting.txt` to contain the resource for the default culture. Store a single string named `HelloString` whose value is "Hello world!" in this file.
2. To indicate that English (en) is the application's default culture, add the following `System.Resources.NeutralResourcesLanguageAttribute` attribute to the application's `AssemblyInfo` file or to the main source code file that will be compiled into the application's main assembly.

```
[assembly: NeutralResourcesLanguage("en")]
```

```
<Assembly: NeutralResourcesLanguage("en")>
```

3. Add support for additional cultures (`en-US` , `fr-FR` , and `ru-RU`) to the application as follows:

- To support the `en-US` or English (United States) culture, create a resource file named `Greeting.en-US.resx` or `Greeting.en-US.txt`, and store in it a single string named `HelloString` whose value is "Hi world!".
- To support the `fr-FR` or French (France) culture, create a resource file named `Greeting.fr-FR.resx` or `Greeting.fr-FR.txt`, and store in it a single string named `HelloString` whose value is "Salut tout le monde!".
- To support the `ru-RU` or Russian (Russia) culture, create a resource file named `Greeting.ru-RU.resx` or `Greeting.ru-RU.txt`, and store in it a single string named `HelloString` whose value is "Всем привет!".

4. Use `resgen.exe` to compile each text or XML resource file to a binary `.resources` file. The output is a set of files that have the same root file name as the `.resx` or `.txt` files, but a `.resources` extension. If you create the example with Visual Studio, the compilation process is handled automatically. If you aren't using Visual Studio, run the following commands to compile the `.resx` files into `.resources` files:

```
resgen Greeting.resx
resgen Greeting.en-us.resx
resgen Greeting.fr-FR.resx
resgen Greeting.ru-RU.resx
```

If your resources are in text files instead of XML files, replace the `.resx` extension with `.txt`.

5. Compile the following source code along with the resources for the default culture into the application's main assembly:

IMPORTANT

If you are using the command line rather than Visual Studio to create the example, you should modify the call to the `ResourceManager` class constructor to the following:

```
ResourceManager rm = new ResourceManager("Greeting", typeof(Example).Assembly);
```

```
using System;
using System.Globalization;
using System.Reflection;
using System.Resources;
using System.Threading;
using System.Windows.Forms;

class Example
{
    static void Main()
    {
        // Create array of supported cultures
        string[] cultures = {"en-CA", "en-US", "fr-FR", "ru-RU"};
        Random rnd = new Random();
        int cultureNdx = rnd.Next(0, cultures.Length);
        CultureInfo originalCulture = Thread.CurrentThread.CurrentCulture;

        try {
            CultureInfo newCulture = new CultureInfo(cultures[cultureNdx]);
            Thread.CurrentThread.CurrentCulture = newCulture;
            Thread.CurrentThread.CurrentUICulture = newCulture;
            ResourceManager rm = new ResourceManager("Example.Greeting",
                typeof(Example).Assembly);
            string greeting = String.Format("The current culture is {0}.\n{1}",
                Thread.CurrentThread.CurrentUICulture.Name,
                rm.GetString("HelloString"));

            MessageBox.Show(greeting);
        }
        catch (CultureNotFoundException e) {
            Console.WriteLine("Unable to instantiate culture {0}", e.InvalidCultureName);
        }
        finally {
            Thread.CurrentThread.CurrentCulture = originalCulture;
            Thread.CurrentThread.CurrentUICulture = originalCulture;
        }
    }
}
```

```

Imports System.Globalization
Imports System.Resources
Imports System.Threading

Module Module1

    Sub Main()
        ' Create array of supported cultures
        Dim cultures() As String = {"en-CA", "en-US", "fr-FR", "ru-RU"}
        Dim rnd As New Random()
        Dim cultureNdx As Integer = rnd.Next(0, cultures.Length)
        Dim originalCulture As CultureInfo = Thread.CurrentThread.CurrentCulture

        Try
            Dim newCulture As New CultureInfo(cultures(cultureNdx))
            Thread.CurrentThread.CurrentCulture = newCulture
            Thread.CurrentThread.CurrentUICulture = newCulture
            Dim greeting As String = String.Format("The current culture is {0}.{1}{2}",
                Thread.CurrentThread.CurrentCulture.Name,
                vbCrLf, My.Resources.Greetings.HelloString)

            MsgBox(greeting)
        Catch e As CultureNotFoundException
            Console.WriteLine("Unable to instantiate culture {0}", e.InvalidCultureName)
        Finally
            Thread.CurrentThread.CurrentCulture = originalCulture
            Thread.CurrentThread.CurrentUICulture = originalCulture
        End Try
    End Sub
End Module

```

If the application is named **Example** and you're compiling from the command line, the command for the C# compiler is:

```
csc Example.cs -res:Greeting.resources
```

The corresponding Visual Basic compiler command is:

```
vbc Example.vb -res:Greeting.resources
```

6. Create a subdirectory in the main application directory for each localized culture supported by the application. You should create an *en-US*, an *fr-FR*, and an *ru-RU* subdirectory. Visual Studio creates these subdirectories automatically as part of the compilation process.
7. Embed the individual culture-specific *.resources* files into satellite assemblies and save them to the appropriate directory. The command to do this for each *.resources* file is:

```
al -target:lib -embed:Greeting.culture.resources -culture:culture -out:culture\Example.resources.dll
```

where *culture* is the name of the culture whose resources the satellite assembly contains. Visual Studio handles this process automatically.

You can then run the example. It will randomly make one of the supported cultures the current culture and display a localized greeting.

Install satellite assemblies in the Global Assembly Cache

Instead of installing assemblies in a local application subdirectory, you can install them in the global assembly cache. This is particularly useful if you have class libraries and class library resource assemblies that are used by multiple applications.

Installing assemblies in the global assembly cache requires that they have strong names. Strong-named assemblies are signed with a valid public/private key pair. They contain version information that the runtime uses to determine which assembly to use to satisfy a binding request. For more information about strong names and versioning, see [Assembly versioning](#). For more information about strong names, see [Strong-named assemblies](#).

When you're developing an application, it's unlikely that you'll have access to the final public/private key pair. To install a satellite assembly in the global assembly cache and ensure that it works as expected, you can use a technique called delayed signing. When you delay sign an assembly, at build time you reserve space in the file for the strong name signature. The actual signing is delayed until later, when the final public/private key pair is available. For more information about delayed signing, see [Delay signing an assembly](#).

Obtain the public key

To delay sign an assembly, you must have access to the public key. You can either obtain the real public key from the organization in your company that will do the eventual signing, or create a public key by using the [Strong Name tool \(sn.exe\)](#).

The following *Sn.exe* command creates a test public/private key pair. The **-k** option specifies that *Sn.exe* should create a new key pair and save it in a file named *TestKeyPair.snk*.

```
sn -k TestKeyPair.snk
```

You can extract the public key from the file that contains the test key pair. The following command extracts the public key from *TestKeyPair.snk* and saves it in *PublicKey.snk*.

```
sn -p TestKeyPair.snk PublicKey.snk
```

Delay signing an Assembly

After you obtain or create the public key, you use the [Assembly Linker \(al.exe\)](#) to compile the assembly and specify delayed signing.

The following *al.exe* command creates a strong-named satellite assembly for the application StringLibrary from the *strings.ja.resources* file:

```
al -target:lib -embed:strings.ja.resources -culture:ja -out:StringLibrary.resources.dll -delay+ -keyfile:PublicKey.snk
```

The **-delay+** option specifies that the Assembly Linker should delay sign the assembly. The **-keyfile** option specifies the name of the key file that contains the public key to use to delay sign the assembly.

Re-signing an Assembly

Before you deploy your application, you must re-sign the delay signed satellite assembly with the real key pair. You can do this by using *Sn.exe*.

The following *Sn.exe* command signs *StringLibrary.resources.dll* with the key pair stored in the file *RealKeyPair.snk*. The **-R** option specifies that a previously signed or delay signed assembly is to be re-signed.

```
sn -R StringLibrary.resources.dll RealKeyPair.snk
```

Install a satellite assembly in the Global Assembly Cache

When the runtime searches for resources in the resource fallback process, it looks in the [global assembly cache](#) first. (For more information, see the "Resource Fallback Process" section of [Package and deploy resources](#).) As soon as a satellite assembly is signed with a strong name, it can be installed in the global assembly cache by using the [Global Assembly Cache tool \(gacutil.exe\)](#).

The following *Gacutil.exe* command installs *StringLibrary.resources.dll** in the global assembly cache:

```
gacutil -i:StringLibrary.resources.dll
```

The */i* option specifies that *Gacutil.exe* should install the specified assembly into the global assembly cache. After the satellite assembly is installed in the cache, the resources it contains become available to all applications that are designed to use the satellite assembly.

Resources in the Global Assembly Cache: An Example

The following example uses a method in a .NET class library to extract and return a localized greeting from a resource file. The library and its resources are registered in the global assembly cache. The example includes resources for the English (United States), French (France), Russian (Russia), and English cultures. English is the default culture; its resources are stored in the main assembly. The example initially delay-signs the library and its satellite assemblies with a public key, then re-signs them with a public/private key pair. To create the example, do the following:

1. If you aren't using Visual Studio, use the following [Strong Name Tool \(Sn.exe\)](#) command to create a public/private key pair named *ResKey.snk*.

```
sn -k ResKey.snk
```

If you're using Visual Studio, use the **Signing** tab of the project **Properties** dialog box to generate the key file.

2. Use the following [Strong Name Tool \(Sn.exe\)](#) command to create a public key file named *PublicKey.snk*.

```
sn -p ResKey.snk PublicKey.snk
```

3. Create a resource file named *Strings.resx* to contain the resource for the default culture. Store a single string named `Greeting` whose value is "How do you do?" in that file.

4. To indicate that "en" is the application's default culture, add the following

[System.Resources.NeutralResourcesLanguageAttribute](#) attribute to the application's *AssemblyInfo* file or to the main source code file that will be compiled into the application's main assembly:

```
[assembly:NeutralResourcesLanguageAttribute("en")]
```

```
<Assembly: NeutralResourcesLanguageAttribute("en")>
```

5. Add support for additional cultures (the en-US, fr-FR, and ru-RU cultures) to the application as follows:

- To support the "en-US" or English (United States) culture, create a resource file named *Strings.en-US.resx* or *Strings.en-US.txt*, and store in it a single string named `Greeting` whose value is "Hello!".
- To support the "fr-FR" or French (France) culture, create a resource file named *Strings.fr-FR.resx* or *Strings.fr-FR.txt* and store in it a single string named `Greeting` whose value is "Bon jour!".

- To support the "ru-RU" or Russian (Russia) culture, create a resource file named *Strings.ru-RU.resx* or *Strings.ru-RU.txt* and store in it a single string named `Greeting` whose value is "Привет!".
6. Use `resgen.exe` to compile each text or XML resource file to a binary .resources file. The output is a set of files that have the same root file name as the *.resx* or *.txt* files, but a *.resources* extension. If you create the example with Visual Studio, the compilation process is handled automatically. If you aren't using Visual Studio, run the following command to compile the *.resx* files into *.resources* files:

```
resgen filename
```

Where *filename* is the optional path, file name, and extension of the *.resx* or text file.

7. Compile the following source code for *StringLibrary.vb* or *StringLibrary.cs* along with the resources for the default culture into a delay signed library assembly named *StringLibrary.dll*.

IMPORTANT

If you are using the command line rather than Visual Studio to create the example, you should modify the call to the `ResourceManager` class constructor to `ResourceManager rm = new ResourceManager("Strings", typeof(Example).Assembly);`.

```
using System;
using System.Globalization;
using System.Reflection;
using System.Resources;
using System.Threading;

[assembly:NeutralResourcesLanguageAttribute("en")]

public class StringLibrary
{
    public string GetGreeting()
    {
        ResourceManager rm = new ResourceManager("Strings",
                                                Assembly.GetAssembly(typeof(StringLibrary)));
        string greeting = rm.GetString("Greeting");
        return greeting;
    }
}
```

```
Imports System.Globalization
Imports System.Reflection
Imports System.Resources
Imports System.Threading

<Assembly: NeutralResourcesLanguageAttribute("en")>

Public Class StringLibrary
    Public Function GetGreeting() As String
        Dim rm As New ResourceManager("Strings", _
                                      Assembly.GetAssembly(GetType(StringLibrary)))
        Dim greeting As String = rm.GetString("Greeting")
        Return greeting
    End Function
End Class
```

The command for the C# compiler is:

```
csc -t:library -resource:Strings.resources -delaysign+ -keyfile:publickey.snk StringLibrary.cs
```

The corresponding Visual Basic compiler command is:

```
vbc -t:library -resource:Strings.resources -delaysign+ -keyfile:publickey.snk StringLibrary.vb
```

8. Create a subdirectory in the main application directory for each localized culture supported by the application. You should create an *en-US*, an *fr-FR*, and an *ru-RU* subdirectory. Visual Studio creates these subdirectories automatically as part of the compilation process. Because all satellite assemblies have the same file name, the subdirectories are used to store individual culture-specific satellite assemblies until they're signed with a public/private key pair.
9. Embed the individual culture-specific *.resources* files into delay signed satellite assemblies and save them to the appropriate directory. The command to do this for each *.resources* file is:

```
al -target:lib -embed:Strings.culture.resources -culture:culture -  
out:culture\StringLibrary.resources.dll -delay+ -keyfile:publickey.snk
```

where *culture* is the name of a culture. In this example, the culture names are *en-US*, *fr-FR*, and *ru-RU*.

10. Re-sign *StringLibrary.dll* by using the [Strong Name tool \(*sn.exe*\)](#) as follows:

```
sn -R StringLibrary.dll RealKeyPair.snk
```

11. Re-sign the individual satellite assemblies. To do this, use the [Strong Name tool \(*sn.exe*\)](#) as follows for each satellite assembly:

```
sn -R StringLibrary.resources.dll RealKeyPair.snk
```

12. Register *StringLibrary.dll* and each of its satellite assemblies in the global assembly cache by using the following command:

```
gacutil -i filename
```

where *filename* is the name of the file to register.

13. If you're using Visual Studio, create a new **Console Application** project named `Example`, add a reference to *StringLibrary.dll* and the following source code to it, and compile.

```

using System;
using System.Globalization;
using System.Threading;

public class Example
{
    public static void Main()
    {
        string[] cultureNames = { "en-GB", "en-US", "fr-FR", "ru-RU" };
        Random rnd = new Random();
        string cultureName = cultureNames[rnd.Next(0, cultureNames.Length)];
        Thread.CurrentThread.CurrentCulture = CultureInfo.CreateSpecificCulture(cultureName);
        Console.WriteLine("The current UI culture is {0}",
            Thread.CurrentThread.CurrentCulture.Name);
        StringLibrary strLib = new StringLibrary();
        string greeting = strLib.GetGreeting();
        Console.WriteLine(greeting);
    }
}

```

```

Imports System.Globalization
Imports System.Threading

Module Example
    Public Sub Main()
        Dim cultureNames() As String = {"en-GB", "en-US", "fr-FR", "ru-RU"}
        Dim rnd As New Random()
        Dim cultureName As String = cultureNames(rnd.Next(0, cultureNames.Length))
        Thread.CurrentThread.CurrentCulture = CultureInfo.CreateSpecificCulture(cultureName)
        Console.WriteLine("The current UI culture is {0}",
            Thread.CurrentThread.CurrentCulture.Name)
        Dim strLib As New StringLibrary()
        Dim greeting As String = strLib.GetGreeting()
        Console.WriteLine(greeting)
    End Sub
End Module

```

To compile from the command line, use the following command for the C# compiler:

```
csc Example.cs -r:StringLibrary.dll
```

The command line for the Visual Basic compiler is:

```
vbc Example.vb -r:StringLibrary.dll
```

14. Run *Example.exe*.

See also

- [Package and deploy resources](#)
- [Delay signing an assembly](#)
- [al.exe \(Assembly Linker\)](#)
- [sn.exe \(Strong Name tool\)](#)
- [gacutil.exe \(Global Assembly Cache tool\)](#)
- [Resources in .NET](#)

Package and deploy resources in .NET Apps

9/20/2022 • 15 minutes to read • [Edit Online](#)

Applications rely on the .NET Framework Resource Manager, represented by the [ResourceManager](#) class, to retrieve localized resources. The Resource Manager assumes that a hub and spoke model is used to package and deploy resources. The hub is the main assembly that contains the nonlocalizable executable code and the resources for a single culture, called the neutral or default culture. The default culture is the fallback culture for the application; it is the culture whose resources are used if localized resources cannot be found. Each spoke connects to a satellite assembly that contains the resources for a single culture, but does not contain any code.

There are several advantages to this model:

- You can incrementally add resources for new cultures after you have deployed an application. Because subsequent development of culture-specific resources can require a significant amount of time, this allows you to release your main application first, and deliver culture-specific resources at a later date.
- You can update and change an application's satellite assemblies without recompiling the application.
- An application needs to load only those satellite assemblies that contain the resources needed for a particular culture. This can significantly reduce the use of system resources.

However, there are also disadvantages to this model:

- You must manage multiple sets of resources.
- The initial cost of testing an application increases, because you must test several configurations. Note that in the long term it will be easier and less expensive to test one core application with several satellites, than to test and maintain several parallel international versions.

Resource naming conventions

When you package your application's resources, you must name them using the resource naming conventions that the common language runtime expects. The runtime identifies a resource by its culture name. Each culture is given a unique name, which is usually a combination of a two-letter, lowercase culture name associated with a language and, if required, a two-letter, uppercase subculture name associated with a country or region. The subculture name follows the culture name, separated by a dash (-). Examples include ja-JP for Japanese as spoken in Japan, en-US for English as spoken in the United States, de-DE for German as spoken in Germany, or de-AT for German as spoken in Austria. See the **Language tag** column in the [list of language/region names supported by Windows](#). Culture names follow the standard defined by [BCP 47](#).

NOTE

There are some exceptions for the two-letter culture names, such as `zh-Hans` for Chinese (Simplified).

For more information, see [Create resource files](#) and [Create satellite assemblies](#).

The resource fallback process

The hub and spoke model for packaging and deploying resources uses a fallback process to locate appropriate resources. If an application requests a localized resource that is unavailable, the common language runtime searches the hierarchy of cultures for an appropriate fallback resource that most closely matches the user's application's request, and throws an exception only as a last resort. At each level of the hierarchy, if an

appropriate resource is found, the runtime uses it. If the resource is not found, the search continues at the next level.

To improve lookup performance, apply the [NeutralResourcesLanguageAttribute](#) attribute to your main assembly, and pass it the name of the neutral language that will work with your main assembly.

.NET Framework resource fallback process

The .NET Framework resource fallback process involves the following steps:

TIP

You may be able to use the `<relativeBindForResources>` configuration element to optimize the resource fallback process and the process by which the runtime probes for resource assemblies. For more information, see [Optimizing the resource fallback process](#).

1. The runtime first checks the [global assembly cache](#) for an assembly that matches the requested culture for your application.

The global assembly cache can store resource assemblies that are shared by many applications. This frees you from having to include specific sets of resources in the directory structure of every application you create. If the runtime finds a reference to the assembly, it searches the assembly for the requested resource. If it finds the entry in the assembly, it uses the requested resource. If it doesn't find the entry, it continues the search.

2. The runtime next checks the directory of the currently executing assembly for a subdirectory that matches the requested culture. If it finds the subdirectory, it searches that subdirectory for a valid satellite assembly for the requested culture. The runtime then searches the satellite assembly for the requested resource. If it finds the resource in the assembly, it uses it. If it doesn't find the resource, it continues the search.
3. The runtime next queries the Windows Installer to determine whether the satellite assembly is to be installed on demand. If so, it handles the installation, loads the assembly, and searches it for the requested resource. If it finds the resource in the assembly, it uses it. If it doesn't find the resource, it continues the search.
4. The runtime raises the [AppDomain.AssemblyResolve](#) event to indicate that it is unable to find the satellite assembly. If you choose to handle the event, your event handler can return a reference to the satellite assembly whose resources will be used for the lookup. Otherwise, the event handler returns `null` and the search continues.
5. The runtime next searches the global assembly cache again, this time for the parent assembly of the requested culture. If the parent assembly exists in the global assembly cache, the runtime searches the assembly for the requested resource.

The parent culture is defined as the appropriate fallback culture. Consider parents as fallback candidates, because providing any resource is preferable to throwing an exception. This process also allows you to reuse resources. You should include a particular resource at the parent level only if the child culture doesn't need to localize the requested resource. For example, if you supply satellite assemblies for `en` (neutral English), `en-GB` (English as spoken in the United Kingdom), and `en-US` (English as spoken in the United States), the `en` satellite would contain the common terminology, and the `en-GB` and `en-US` satellites could provide overrides for only those terms that differ.

6. The runtime next checks the directory of the currently executing assembly to see if it contains a parent directory. If a parent directory exists, the runtime searches the directory for a valid satellite assembly for the parent culture. If it finds the assembly, the runtime searches the assembly for the requested resource. If it finds the resource, it uses it. If it doesn't find the resource, it continues the search.

7. The runtime next queries the Windows Installer to determine whether the parent satellite assembly is to be installed on demand. If so, it handles the installation, loads the assembly, and searches it or the requested resource. If it finds the resource in the assembly, it uses it. If it doesn't find the resource, it continues the search.
8. The runtime raises the [AppDomain.AssemblyResolve](#) event to indicate that it is unable to find an appropriate fallback resource. If you choose to handle the event, your event handler can return a reference to the satellite assembly whose resources will be used for the lookup. Otherwise, the event handler returns `null` and the search continues.
9. The runtime next searches parent assemblies, as in the previous three steps, through many potential levels. Each culture has only one parent, which is defined by the [CultureInfo.Parent](#) property, but a parent might have its own parent. The search for parent cultures stops when a culture's [Parent](#) property returns [CultureInfo.InvariantCulture](#); for resource fallback, the invariant culture is not considered a parent culture or a culture that can have resources.
10. If the culture that was originally specified and all parents have been searched and the resource is still not found, the resource for the default (fallback) culture is used. Typically, the resources for the default culture are included in the main application assembly. However, you can specify a value of [Satellite](#) for the [Location](#) property of the [NeutralResourcesLanguageAttribute](#) attribute to indicate that the ultimate fallback location for resources is a satellite assembly, rather than the main assembly.

NOTE

The default resource is the only resource that can be compiled with the main assembly. Unless you specify a satellite assembly by using the [NeutralResourcesLanguageAttribute](#) attribute, it is the ultimate fallback (final parent). Therefore, we recommend that you always include a default set of resources in your main assembly. This helps prevent exceptions from being thrown. By including a default resource, file you provide a fallback for all resources and ensure that at least one resource is always present for the user, even if it is not culturally specific.

11. Finally, if the runtime doesn't find a resource for a default (fallback) culture, a [MissingManifestResourceException](#) or [MissingSatelliteAssemblyException](#) exception is thrown to indicate that the resource could not be found.

For example, suppose the application requests a resource localized for Spanish (Mexico) (the `es-MX` culture). The runtime first searches the global assembly cache for the assembly that matches `es-MX`, but doesn't find it. The runtime then searches the directory of the currently executing assembly for an `es-MX` directory. Failing that, the runtime searches the global assembly cache again for a parent assembly that reflects the appropriate fallback culture — in this case, `es` (Spanish). If the parent assembly is not found, the runtime searches all potential levels of parent assemblies for the `es-MX` culture until it finds a corresponding resource. If a resource isn't found, the runtime uses the resource for the default culture.

Optimize the .NET Framework resource fallback process

Under the following conditions, you can optimize the process by which the runtime searches for resources in satellite assemblies

- Satellite assemblies are deployed in the same location as the code assembly. If the code assembly is installed in the [Global Assembly Cache](#), satellite assemblies are also installed in the global assembly cache. If the code assembly is installed in a directory, satellite assemblies are installed in culture-specific folders of that directory.
- Satellite assemblies are not installed on demand.
- Application code does not handle the [AppDomain.AssemblyResolve](#) event.

You optimize the probe for satellite assemblies by including the `<relativeBindForResources>` element and setting its `enabled` attribute to `true` in the application configuration file, as shown in the following example.

```
<configuration>
  <runtime>
    <relativeBindForResources enabled="true" />
  </runtime>
</configuration>
```

The optimized probe for satellite assemblies is an opt-in feature. That is, the runtime follows the steps documented in [The resource fallback process](#) unless the `<relativeBindForResources>` element is present in the application's configuration file and its `enabled` attribute is set to `true`. If this is the case, the process of probing for a satellite assembly is modified as follows:

- The runtime uses the location of the parent code assembly to probe for the satellite assembly. If the parent assembly is installed in the global assembly cache, the runtime probes in the cache but not in the application's directory. If the parent assembly is installed in an application directory, the runtime probes in the application directory but not in the global assembly cache.
- The runtime doesn't query the Windows Installer for on-demand installation of satellite assemblies.
- If the probe for a particular resource assembly fails, the runtime does not raise the [AppDomain.AssemblyResolve](#) event.

.NET Core resource fallback process

The .NET Core resource fallback process involves the following steps:

1. The runtime attempts to load a satellite assembly for the requested culture.

- Checks the directory of the currently executing assembly for a subdirectory that matches the requested culture. If it finds the subdirectory, it searches that subdirectory for a valid satellite assembly for the requested culture and loads it.

NOTE

On operating systems with case-sensitive file systems (that is, Linux and macOS), the culture name subdirectory search is case-sensitive. The subdirectory name must exactly match the case of the [CultureInfo.Name](#) (for example, `es` or `es-MX`).

NOTE

If the programmer has derived a custom assembly load context from [AssemblyLoadContext](#), the situation is complicated. If the executing assembly was loaded into the custom context, the runtime loads the satellite assembly into the custom context. The details are out of scope for this document. See [AssemblyLoadContext](#).

- If a satellite assembly has not been found, the [AssemblyLoadContext](#) raises the [AssemblyLoadContext.Resolving](#) event to indicate that it is unable to find the satellite assembly. If you choose to handle the event, your event handler can load and return a reference to the satellite assembly.
- If a satellite assembly still has not been found, the AssemblyLoadContext causes the AppDomain to trigger an [AppDomain.AssemblyResolve](#) event to indicate that it is unable to find the satellite assembly. If you choose to handle the event, your event handler can load and return a reference to the satellite assembly.

2. If a satellite assembly is found, the runtime searches it for the requested resource. If it finds the resource in the assembly, it uses it. If it doesn't find the resource, it continues the search.

NOTE

To find a resource within the satellite assembly, the runtime searches for the resource file requested by the [ResourceManager](#) for the current [CultureInfo.Name](#). Within the resource file it searches for the requested resource name. If either is not found, the resource is treated as not found.

3. The runtime next searches the parent culture assemblies through many potential levels, each time repeating steps 1 & 2.

The parent culture is defined as an appropriate fallback culture. Consider parents as fallback candidates, because providing any resource is preferable to throwing an exception. This process also allows you to reuse resources. You should include a particular resource at the parent level only if the child culture doesn't need to localize the requested resource. For example, if you supply satellite assemblies for `en` (neutral English), `en-GB` (English as spoken in the United Kingdom), and `en-US` (English as spoken in the United States), the `en` satellite contains the common terminology, and the `en-GB` and `en-US` satellites provides overrides for only those terms that differ.

Each culture has only one parent, which is defined by the [CultureInfo.Parent](#) property, but a parent might have its own parent. The search for parent cultures stops when a culture's [Parent](#) property returns [CultureInfo.InvariantCulture](#). For resource fallback, the invariant culture is not considered a parent culture or a culture that can have resources.

4. If the culture that was originally specified and all parents have been searched and the resource is still not found, the resource for the default (fallback) culture is used. Typically, the resources for the default culture are included in the main application assembly. However, you can specify a value of [Satellite](#) for the [Location](#) property to indicate that the ultimate fallback location for resources is a satellite assembly rather than the main assembly.

NOTE

The default resource is the only resource that can be compiled with the main assembly. Unless you specify a satellite assembly by using the [NeutralResourcesLanguageAttribute](#) attribute, it is the ultimate fallback (final parent). Therefore, we recommend that you always include a default set of resources in your main assembly. This helps prevent exceptions from being thrown. By including a default resource file, you provide a fallback for all resources and ensure that at least one resource is always present for the user, even if it is not culturally specific.

5. Finally, if the runtime doesn't find a resource file for a default (fallback) culture, a [MissingManifestResourceException](#) or [MissingSatelliteAssemblyException](#) exception is thrown to indicate that the resource could not be found. If the resource file is found but the requested resource is not present the request returns `null`.

Ultimate fallback to satellite assembly

You can optionally remove resources from the main assembly and specify that the runtime should load the ultimate fallback resources from a satellite assembly that corresponds to a specific culture. To control the fallback process, you use the [NeutralResourcesLanguageAttribute\(String, UltimateResourceFallbackLocation\)](#) constructor and supply a value for the [UltimateResourceFallbackLocation](#) parameter that specifies whether Resource Manager should extract the fallback resources from the main assembly or from a satellite assembly.

The following .NET Framework example uses the [NeutralResourcesLanguageAttribute](#) attribute to store an application's fallback resources in a satellite assembly for the French (`fr`) language. The example has two text-based resource files that define a single string resource named `Greeting`. The first, `resources.fr.txt`, contains a

French language resource.

```
Greeting=Bon jour!
```

The second, resources.ru.txt, contains a Russian language resource.

```
Greeting=Добрый день
```

These two files are compiled to .resources files by running [Resource File Generator \(resgen.exe\)](#) from the command line. For the French language resource, the command is:

```
resgen.exe resources.fr.txt
```

For the Russian language resource, the command is:

```
resgen.exe resources.ru.txt
```

The .resources files are embedded into dynamic link libraries by running [Assembly Linker \(al.exe\)](#) from the command line for the French language resource as follows:

```
al /t:lib /embed:resources.fr.resources /culture:fr /out:fr\Example1.resources.dll
```

and for the Russian language resource as follows:

```
al /t:lib /embed:resources.ru.resources /culture:ru /out:ru\Example1.resources.dll
```

The application source code resides in a file named Example1.cs or Example1.vb. It includes the [NeutralResourcesLanguageAttribute](#) attribute to indicate that the default application resource is in the fr subdirectory. It instantiates the Resource Manager, retrieves the value of the `Greeting` resource, and displays it to the console.

```
using System;
using System.Reflection;
using System.Resources;

[assembly:NeutralResourcesLanguage("fr", UltimateResourceFallbackLocation.Satellite)]

public class Example
{
    public static void Main()
    {
        ResourceManager rm = new ResourceManager("resources",
                                                typeof(Example).Assembly);
        string greeting = rm.GetString("Greeting");
        Console.WriteLine(greeting);
    }
}
```

```
Imports System.Reflection
Imports System.Resources

<Assembly: NeutralResourcesLanguage("fr", UltimateResourceFallbackLocation.Satellite)>
Module Example
    Public Sub Main()
        Dim rm As New ResourceManager("resources", GetType(Example).Assembly)
        Dim greeting As String = rm.GetString("Greeting")
        Console.WriteLine(greeting)
    End Sub
End Module
```

You can then compile C# source code from the command line as follows:

```
csc Example1.cs
```

The command for the Visual Basic compiler is very similar:

```
vbc Example1.vb
```

Because there are no resources embedded in the main assembly, you do not have to compile by using the `/resource` switch.

When you run the example from a system whose language is anything other than Russian, it displays the following output:

```
Bon jour!
```

Suggested packaging alternative

Time or budget constraints might prevent you from creating a set of resources for every subculture that your application supports. Instead, you can create a single satellite assembly for a parent culture that all related subcultures can use. For example, you can provide a single English satellite assembly (en) that is retrieved by users who request region-specific English resources, and a single German satellite assembly (de) for users who request region-specific German resources. For example, requests for German as spoken in Germany (de-DE), Austria (de-AT), and Switzerland (de-CH) would fall back to the German satellite assembly (de). The default resources are the final fallback and therefore should be the resources that will be requested by the majority of your application's users, so choose these resources carefully. This approach deploys resources that are less culturally specific, but can significantly reduce your application's localization costs.

See also

- [Resources in .NET apps](#)
- [Global Assembly Cache](#)
- [Create resource files](#)
- [Create satellite assemblies](#)

Retrieve resources in .NET apps

9/20/2022 • 15 minutes to read • [Edit Online](#)

When you work with localized resources in .NET apps, you should ideally package the resources for the default or neutral culture with the main assembly and create a separate satellite assembly for each language or culture that your app supports. You can then use the [ResourceManager](#) class as described in the next section to access named resources. If you choose not to embed your resources in the main assembly and satellite assemblies, you can also access binary *.resources* files directly, as discussed in the section [Retrieve resources from .resources files](#) later in this article.

Retrieve resources from assemblies

The [ResourceManager](#) class provides access to resources at run time. You use the [ResourceManager.GetString](#) method to retrieve string resources and the [ResourceManager.GetObject](#) or [ResourceManager.GetStream](#) method to retrieve non-string resources. Each method has two overloads:

- An overload whose single parameter is a string that contains the name of the resource. The method attempts to retrieve that resource for the current culture. For more information, see the [GetString\(String\)](#), [GetObject\(String\)](#), and [GetStream\(String\)](#) methods.
- An overload that has two parameters: a string containing the name of the resource, and a [CultureInfo](#) object that represents the culture whose resource is to be retrieved. If a resource set for that culture cannot be found, the resource manager uses fallback rules to retrieve an appropriate resource. For more information, see the [GetString\(String, CultureInfo\)](#), [GetObject\(String, CultureInfo\)](#), and [GetStream\(String, CultureInfo\)](#) methods.

The resource manager uses the resource fallback process to control how the app retrieves culture-specific resources. For more information, see the "Resource Fallback Process" section in [Package and deploy resources](#). For information about instantiating a [ResourceManager](#) object, see the "Instantiating a ResourceManager Object" section in the [ResourceManager](#) class topic.

Retrieve string data example

The following example calls the [GetString\(String\)](#) method to retrieve the string resources of the current UI culture. It includes a neutral string resource for the English (United States) culture and localized resources for the French (France) and Russian (Russia) cultures. The following English (United States) resource is in a file named Strings.txt:

```
TimeHeader=The current time is
```

The French (France) resource is in a file named Strings.fr-FR.txt:

```
TimeHeader=L'heure actuelle est
```

The Russian (Russia) resource is in a file named Strings.ru-RU-txt:

```
TimeHeader=Текущее время –
```

The source code for this example, which is in a file named GetString.cs for the C# version of the code and

GetString.vb for the Visual Basic version, defines a string array that contains the name of four cultures: the three cultures for which resources are available and the Spanish (Spain) culture. A loop that executes five times randomly selects one of these cultures and assigns it to the [Thread.CurrentCulture](#) and [CultureInfo.CurrentUICulture](#) properties. It then calls the [GetString\(String\)](#) method to retrieve the localized string, which it displays along with the time of day.

```
using System;
using System.Globalization;
using System.Resources;
using System.Threading;

[assembly: NeutralResourcesLanguageAttribute("en-US")]

public class Example
{
    public static void Main()
    {
        string[] cultureNames = { "en-US", "fr-FR", "ru-RU", "es-ES" };
        Random rnd = new Random();
        ResourceManager rm = new ResourceManager("Strings",
            typeof(Example).Assembly);

        for (int ctr = 0; ctr <= cultureNames.Length; ctr++) {
            string cultureName = cultureNames[rnd.Next(0, cultureNames.Length)];
            CultureInfo culture = CultureInfo.CreateSpecificCulture(cultureName);
            Thread.CurrentThread.CurrentCulture = culture;
            Thread.CurrentThread.CurrentUICulture = culture;

            Console.WriteLine("Current culture: {0}", culture.NativeName);
            string timeString = rm.GetString("TimeHeader");
            Console.WriteLine("{0} {1:T}\n", timeString, DateTime.Now);
        }
    }
}

// The example displays output like the following:
// Current culture: English (United States)
// The current time is 9:34:18 AM
//
// Current culture: Español (España, alfabetización internacional)
// The current time is 9:34:18
//
// Current culture: русский (Россия)
// Текущее время – 9:34:18
//
// Current culture: français (France)
// L'heure actuelle est 09:34:18
//
// Current culture: русский (Россия)
// Текущее время – 9:34:18
```

```

Imports System.Globalization
Imports System.Resources
Imports System.Threading

<Assembly: NeutralResourcesLanguageAttribute("en-US")>

Module Example
    Public Sub Main()
        Dim cultureNames() As String = {"en-US", "fr-FR", "ru-RU", "es-ES"}
        Dim rnd As New Random()
        Dim rm As New ResourceManager("Strings", GetType(Example).Assembly)

        For ctr As Integer = 0 To cultureNames.Length
            Dim cultureName As String = cultureNames(rnd.Next(0, cultureNames.Length))
            Dim culture As CultureInfo = CultureInfo.CreateSpecificCulture(cultureName)
            Thread.CurrentThread.CurrentCulture = culture
            Thread.CurrentThread.CurrentUICulture = culture

            Console.WriteLine("Current culture: {0}", culture.NativeName)
            Dim timeString As String = rm.GetString("TimeHeader")
            Console.WriteLine("{0} {1:T}", timeString, Date.Now)
            Console.WriteLine()

        Next
    End Sub
End Module
' The example displays output similar to the following:
' Current culture: English (United States)
' The current time is 9:34:18 AM
'
' Current culture: Español (España, alfabetización internacional)
' The current time is 9:34:18
'
' Current culture: русский (Россия)
' Текущее время – 9:34:18
'
' Current culture: français (France)
' L'heure actuelle est 09:34:18
'
' Current culture: русский (Россия)
' Текущее время – 9:34:18

```

The following batch (.bat) file compiles the example and generates satellite assemblies in the appropriate directories. The commands are provided for the C# language and compiler. For Visual Basic, change `csc` to `vbc`, and change `GetString.cs` to `GetString.vb`.

```

resgen strings.txt
csc GetString.cs -resource:string.resources

resgen strings.fr-FR.txt
md fr-FR
al -embed:string.fr-FR.resources -culture:fr-FR -out:fr-FR\GetString.resources.dll

resgen strings.ru-RU.txt
md ru-RU
al -embed:string.ru-RU.resources -culture:ru-RU -out:ru-RU\GetString.resources.dll

```

When the current UI culture is Spanish (Spain), note that the example displays English language resources, because Spanish language resources are unavailable, and English is the example's default culture.

Retrieve object data examples

You can use the `GetObject` and `GetStream` methods to retrieve object data. This includes primitive data types, serializable objects, and objects that are stored in binary format (such as images).

The following example uses the [GetStream\(String\)](#) method to retrieve a bitmap that is used in an app's opening splash window. The following source code in a file named CreateResources.cs (for C#) or CreateResources.vb (for Visual Basic) generates a .resx file that contains the serialized image. In this case, the image is loaded from a file named SplashScreen.jpg; you can modify the file name to substitute your own image.

```
using System;
using System.Drawing;
using System.Drawing.Imaging;
using System.IO;
using System.Resources;

public class Example
{
    public static void Main()
    {
        Bitmap bmp = new Bitmap(@".\SplashScreen.jpg");
        MemoryStream imageStream = new MemoryStream();
        bmp.Save(imageStream, ImageFormat.Jpeg);

        ResXResourceWriter writer = new ResXResourceWriter("AppResources.resx");
        writer.AddResource("SplashScreen", imageStream);
        writer.Generate();
        writer.Close();
    }
}
```

```
Imports System.Drawing
Imports System.Drawing.Imaging
Imports System.IO
Imports System.Resources

Module Example
    Public Sub Main()
        Dim bmp As New Bitmap(".\SplashScreen.jpg")
        Dim imageStream As New MemoryStream()
        bmp.Save(imageStream, ImageFormat.Jpeg)

        Dim writer As New ResXResourceWriter("AppResources.resx")
        writer.AddResource("SplashScreen", imageStream)
        writer.Generate()
        writer.Close()
    End Sub
End Module
```

The following code retrieves the resource and displays the image in a [PictureBox](#) control.

```

using System;
using System.Drawing;
using System.IO;
using System.Resources;
using System.Windows.Forms;

public class Example
{
    public static void Main()
    {
        ResourceManager rm = new ResourceManager("AppResources", typeof(Example).Assembly);
        Bitmap screen = (Bitmap) Image.FromStream(rm.GetStream("SplashScreen"));

        Form frm = new Form();
        frm.Size = new Size(300, 300);

        PictureBox pic = new PictureBox();
        pic.Bounds = frm.RestoreBounds;
        pic.BorderStyle = BorderStyle.Fixed3D;
        pic.Image = screen;
        pic.SizeMode = PictureBoxSizeMode.StretchImage;

        frm.Controls.Add(pic);
        pic.Anchor = AnchorStyles.Top | AnchorStyles.Bottom |
                     AnchorStyles.Left | AnchorStyles.Right;

        frm.ShowDialog();
    }
}

```

```

Imports System.Drawing
Imports System.IO
Imports System.Resources
Imports System.Windows.Forms

Module Example
    Public Sub Main()
        Dim rm As New ResourceManager("AppResources", GetType(Example).Assembly)
        Dim screen As Bitmap = CType(Image.FromStream(rm.GetStream("SplashScreen")), Bitmap)

        Dim frm As New Form()
        frm.Size = new Size(300, 300)

        Dim pic As New PictureBox()
        pic.Bounds = frm.RestoreBounds
        pic.BorderStyle = BorderStyle.Fixed3D
        pic.Image = screen
        pic.SizeMode = PictureBoxSizeMode.StretchImage

        frm.Controls.Add(pic)
        pic.Anchor = AnchorStyles.Top Or AnchorStyles.Bottom Or
                     AnchorStyles.Left Or AnchorStyles.Right

        frm.ShowDialog()
    End Sub
End Module

```

You can use the following batch file to build the C# example. For Visual Basic, change `csc` to `vbc`, and change the extension of the source code file from `.cs` to `.vb`.

```

csc CreateResources.cs
CreateResources

resgen AppResources.resx

csc GetStream.cs -resource:AppResources.resources

```

The following example uses the [ResourceManager.GetObject\(String\)](#) method to deserialize a custom object. The example includes a source code file named UIElements.cs (UIElements.vb for Visual Basic) that defines the following structure named `PersonTable`. This structure is intended to be used by a general table display routine that displays the localized names of table columns. Note that the `PersonTable` structure is marked with the [SerializableAttribute](#) attribute.

```

using System;

[Serializable] public struct PersonTable
{
    public readonly int nColumns;
    public readonly string column1;
    public readonly string column2;
    public readonly string column3;
    public readonly int width1;
    public readonly int width2;
    public readonly int width3;

    public PersonTable(string column1, string column2, string column3,
                      int width1, int width2, int width3)
    {
        this.column1 = column1;
        this.column2 = column2;
        this.column3 = column3;
        this.width1 = width1;
        this.width2 = width2;
        this.width3 = width3;
        this.nColumns = typeof(PersonTable).GetFields().Length / 2;
    }
}

```

```

<Serializable> Public Structure PersonTable
    Public ReadOnly nColumns As Integer
    Public ReadOnly column1 As String
    Public ReadOnly column2 As String
    Public ReadOnly column3 As String
    Public ReadOnly width1 As Integer
    Public ReadOnly width2 As Integer
    Public ReadOnly width3 As Integer

    Public Sub New(column1 As String, column2 As String, column3 As String,
                  width1 As Integer, width2 As Integer, width3 As Integer)
        Me.column1 = column1
        Me.column2 = column2
        Me.column3 = column3
        Me.width1 = width1
        Me.width2 = width2
        Me.width3 = width3
        Me.nColumns = Me.GetType().GetFields().Count \ 2
    End Sub
End Structure

```

The following code from a file named CreateResources.cs (CreateResources.vb for Visual Basic) creates an XML resource file named UIResources.resx that stores a table title and a `PersonTable` object that contains information

for an app that is localized for the English language.

```
using System;
using System.Resources;

public class CreateResource
{
    public static void Main()
    {
        PersonTable table = new PersonTable("Name", "Employee Number",
                                             "Age", 30, 18, 5);
        ResXResourceWriter rr = new ResXResourceWriter(@".\UIResources.resx");
        rr.AddResource("TableName", "Employees of Acme Corporation");
        rr.AddResource("Employees", table);
        rr.Generate();
        rr.Close();
    }
}
```

```
Imports System.Resources

Module CreateResource
    Public Sub Main()
        Dim table As New PersonTable("Name", "Employee Number", "Age", 30, 18, 5)
        Dim rr As New ResXResourceWriter(".\UIResources.resx")
        rr.AddResource("TableName", "Employees of Acme Corporation")
        rr.AddResource("Employees", table)
        rr.Generate()
        rr.Close()
    End Sub
End Module
```

The following code in a source code file named GetObject.cs (GetObject.vb) then retrieves the resources and displays them to the console.

```

using System;
using System.Resources;

[assembly: NeutralResourcesLanguageAttribute("en")]

public class Example
{
    public static void Main()
    {
        string fmtString = String.Empty;
        ResourceManager rm = new ResourceManager("UIResources", typeof(Example).Assembly);
        string title = rm.GetString("TableName");
        PersonTable tableInfo = (PersonTable) rm.GetObject("Employees");

        if (! String.IsNullOrEmpty(title)) {
            fmtString = "{0," + ((Console.WindowWidth + title.Length) / 2).ToString() + "}";
            Console.WriteLine(fmtString, title);
            Console.WriteLine();
        }

        for (int ctr = 1; ctr <= tableInfo.nColumns; ctr++) {
            string columnName = "column" + ctr.ToString();
            string widthName = "width" + ctr.ToString();
            string value = tableInfo.GetType().GetField(columnName).GetValue(tableInfo).ToString();
            int width = (int) tableInfo.GetType().GetField(widthName).GetValue(tableInfo);
            fmtString = "{0,-" + width.ToString() + "}";
            Console.Write(fmtString, value);
        }
        Console.WriteLine();
    }
}

```

```

Imports System.Resources

<Assembly: NeutralResourcesLanguageAttribute("en")>

Module Example
    Public Sub Main()
        Dim fmtString As String = String.Empty
        Dim rm As New ResourceManager("UIResources", GetType(Example).Assembly)
        Dim title As String = rm.GetString("TableName")
        Dim tableInfo As PersonTable = DirectCast(rm.GetObject("Employees"), PersonTable)

        If Not String.IsNullOrEmpty(title) Then
            fmtString = "{0," + ((Console.WindowWidth + title.Length) \ 2).ToString() + "}"
            Console.WriteLine(fmtString, title)
            Console.WriteLine()
        End If

        For ctr As Integer = 1 To tableInfo.nColumns
            Dim columnName As String = "column" + ctr.ToString()
            Dim widthName As String = "width" + ctr.ToString()
            Dim value As String = CStr(tableInfo.GetType().GetField(columnName).GetValue(tableInfo))
            Dim width As Integer = CInt(tableInfo.GetType().GetField(widthName).GetValue(tableInfo))
            fmtString = "{0,-" + width.ToString() + "}"
            Console.Write(fmtString, value)
        Next
        Console.WriteLine()
    End Sub
End Module

```

You can build the necessary resource file and assemblies and run the app by executing the following batch file. You must use the `/r` option to supply Resgen.exe with a reference to UIElements.dll so that it can access information about the `PersonTable` structure. If you're using C#, replace the `vbc` compiler name with `csc`, and

replace the `.vb` extension with `.cs`.

```
vbc -t:library UIElements.vb
vbc CreateResources.vb -r:UIElements.dll
CreateResources

resgen UIResources.resx -r:UIElements.dll
vbc GetObject.vb -r:UIElements.dll -resource:UIResources.resources

GetObject.exe
```

Version support for satellite assemblies

By default, when the [ResourceManager](#) object retrieves requested resources, it looks for satellite assemblies that have version numbers that match the version number of the main assembly. After you have deployed an app, you might want to update the main assembly or specific resource satellite assemblies. The .NET Framework provides support for versioning the main assembly and satellite assemblies.

The [SatelliteContractVersionAttribute](#) attribute provides versioning support for a main assembly. Specifying this attribute on an app's main assembly enables you to update and redeploy a main assembly without updating its satellite assemblies. After you update the main assembly, increment the main assembly's version number but leave the satellite contract version number unchanged. When the resource manager retrieves requested resources, it loads the satellite assembly version specified by this attribute.

Publisher policy assemblies provide support for versioning satellite assemblies. You can update and redeploy a satellite assembly without updating the main assembly. After you update a satellite assembly, increment its version number and ship it with a publisher policy assembly. In the publisher policy assembly, specify that your new satellite assembly is backward-compatible with its previous version. The resource manager will use the [SatelliteContractVersionAttribute](#) attribute to determine the version of the satellite assembly, but the assembly loader will bind to the satellite assembly version specified by the publisher policy. For more information about publisher policy assemblies, see [Create a publisher policy file](#).

To enable full assembly versioning support, we recommend that you deploy strong-named assemblies in the [global assembly cache](#) and deploy assemblies that don't have strong names in the application directory. If you want to deploy strong-named assemblies in the application directory, you will not be able to increment a satellite assembly's version number when you update the assembly. Instead, you must perform an in-place update where you replace the existing code with the updated code and maintain the same version number. For example, if you want to update version 1.0.0.0 of a satellite assembly with the fully specified assembly name "myApp.resources, Version=1.0.0.0, Culture=de, PublicKeyToken=b03f5f11d50a3a", overwrite it with the updated myApp.resources.dll that has been compiled with the same, fully specified assembly name "myApp.resources, Version=1.0.0.0, Culture=de, PublicKeyToken=b03f5f11d50a3a". Note that using in-place updates on satellite assembly files makes it difficult for an app to accurately determine the version of a satellite assembly.

For more information about assembly versioning, see [Assembly versioning](#) and [How the Runtime locates assemblies](#).

Retrieve resources from .resources Files

If you choose not to deploy resources in satellite assemblies, you can still use a [ResourceManager](#) object to access resources from .resources files directly. To do this, you must deploy the .resources files correctly. Then you use the [ResourceManager.CreateFileBasedResourceManager](#) method to instantiate a [ResourceManager](#) object and specify the directory that contains the standalone .resources files.

Deploy .resources Files

When you embed .resources files in an application assembly and satellite assemblies, each satellite assembly has the same file name, but is placed in a subdirectory that reflects the satellite assembly's culture. In contrast, when you access resources from .resources files directly, you can place all the .resources files in a single directory, usually a subdirectory of the application directory. The name of the app's default .resources file consists of a root name only, with no indication of its culture (for example, strings.resources). The resources for each localized culture are stored in a file whose name consists of the root name followed by the culture (for example, strings.ja.resources or strings.de-DE.resources).

The following illustration shows where resource files should be located in the directory structure. It also gives the naming conventions for .resource files.



Use the resource manager

After you have created your resources and placed them in the appropriate directory, you create a `ResourceManager` object to use the resources by calling the `CreateFileBasedResourceManager(String, String, Type)` method. The first parameter specifies the root name of the app's default .resources file (this would be "strings" for the example in the previous section). The second parameter specifies the location of the resources ("Resources" for the previous example). The third parameter specifies the `ResourceSet` implementation to use. If the third parameter is `null`, the default runtime `ResourceSet` is used.

NOTE

Do not deploy ASP.NET apps using standalone .resources files. This can cause locking issues and breaks XCOPY deployment. We recommend that you deploy ASP.NET resources in satellite assemblies. For more information, see [ASP.NET Web Page Resources Overview](#).

After you instantiate the `ResourceManager` object, you use the `GetString`, `GetObject`, and `GetStream` methods as discussed earlier to retrieve the resources. However, the retrieval of resources directly from .resources files differs from the retrieval of embedded resources from assemblies. When you retrieve resources from .resources files, the `GetString(String)`, `GetObject(String)`, and `GetStream(String)` methods always retrieve the default culture's resources regardless of the current culture. To retrieve the resources of either the app's current culture or a specific culture, you must call the `GetString(String, CultureInfo)`, `GetObject(String, CultureInfo)`, or `GetStream(String, CultureInfo)` method and specify the culture whose resources are to be retrieved. To retrieve the resources of the current culture, specify the value of the `CultureInfo.CurrentCulture` property as the `culture` argument. If the resource manager cannot retrieve the resources of `culture`, it uses the standard resource fallback rules to retrieve the appropriate resources.

An example

The following example illustrates how the resource manager retrieves resources directly from .resources files. The example consists of three text-based resource files for the English (United States), French (France), and Russian (Russia) cultures. English (United States) is the example's default culture. Its resources are stored in the following file named *Strings.txt*:

```
Greeting=Hello  
Prompt=What is your name?
```

Resources for the French (France) culture are stored in the following file, which is named Strings.fr-FR.txt:

```
Greeting=Bonjour  
Prompt=Comment vous appelez-vous?
```

Resources for the Russian (Russia) culture are stored in the following file, which is named Strings.ru-RU.txt:

```
Greeting=Здравствуйте  
Prompt=Как вас зовут?
```

The following is the source code for the example. The example instantiates [CultureInfo](#) objects for the English (United States), English (Canada), French (France), and Russian (Russia) cultures, and makes each the current culture. The [ResourceManager.GetString\(String, CultureInfo\)](#) method then supplies the value of the [CultureInfo.CurrentCulture](#) property as the `culture` argument to retrieve the appropriate culture-specific resources.

```
using System;  
using System.Globalization;  
using System.Resources;  
using System.Threading;  
  
[assembly: NeutralResourcesLanguage("en-US")]  
  
public class Example  
{  
    public static void Main()  
    {  
        string[] cultureNames = { "en-US", "en-CA", "ru-RU", "fr-FR" };  
        ResourceManager rm = ResourceManager.CreateFileBasedResourceManager("Strings", "Resources", null);  
  
        foreach (var cultureName in cultureNames) {  
            Thread.CurrentThread.CurrentCulture = CultureInfo.CreateSpecificCulture(cultureName);  
            string greeting = rm.GetString("Greeting", CultureInfo.CurrentCulture);  
            Console.WriteLine("\n{0}!", greeting);  
            Console.Write(rm.GetString("Prompt", CultureInfo.CurrentCulture));  
            string name = Console.ReadLine();  
            if (!String.IsNullOrEmpty(name))  
                Console.WriteLine("{0}, {1}!", greeting, name);  
        }  
        Console.WriteLine();  
    }  
    // The example displays output like the following:  
    //     Hello!  
    //     What is your name? Dakota  
    //     Hello, Dakota!  
    //  
    //     Hello!  
    //     What is your name? Koani  
    //     Hello, Koani!  
    //  
    //     Здравствуйте!  
    //     Как вас зовут?Samuel  
    //     Здравствуйте, Samuel!  
    //  
    //     Bon jour!  
    //     Comment vous appelez-vous?Yiska  
    //     Bon jour, Yiska!
```

```

Imports System.Globalization
Imports System.Resources
Imports System.Threading

<Assembly: NeutralResourcesLanguageAttribute("en-US")>

Module Example
    Public Sub Main()
        Dim cultureNames() As String = {"en-US", "en-CA", "ru-RU", "fr-FR"}
        Dim rm As ResourceManager = ResourceManager.CreateFileBasedResourceManager("Strings", "Resources",
Nothing)

        For Each cultureName In cultureNames
            Console.WriteLine()
            Thread.CurrentThread.CurrentCulture = CultureInfo.CreateSpecificCulture(cultureName)
            Dim greeting As String = rm.GetString("Greeting", CultureInfo.CurrentCulture)
            Console.WriteLine("{0}!", greeting)
            Console.WriteLine(rm.GetString("Prompt", CultureInfo.CurrentCulture))
            Dim name As String = Console.ReadLine()
            If Not String.IsNullOrEmpty(name) Then
                Console.WriteLine("{0}, {1}!", greeting, name)
            End If
        Next
        Console.WriteLine()
    End Sub
End Module
' The example displays output like the following:
' Hello!
' What is your name? Dakota
' Hello, Dakota!
'
' Hello!
' What is your name? Koani
' Hello, Koani!
'
' Здравствуйте!
' Как вас зовут?Samuel
' Здравствуйте, Samuel!
'
' Bon jour!
' Comment vous appelez-vous?Yiska
' Bon jour, Yiska!

```

You can compile the C# version of the example by running the following batch file. If you're using Visual Basic, replace `csc` with `vbc`, and replace the `.cs` extension with `.vb`.

```

md Resources
resgen Strings.txt Resources\Strings.resources
resgen Strings.fr-FR.txt Resources\Strings.fr-FR.resources
resgen Strings.ru-RU.txt Resources\Strings.ru-RU.resources

csc Example.cs

```

See also

- [ResourceManager](#)
- [Resources in .NET apps](#)
- [Package and deploy resources](#)
- [How the Runtime locates assemblies](#)

Worker Services in .NET

9/20/2022 • 6 minutes to read • [Edit Online](#)

There are numerous reasons for creating long-running services such as:

- Processing CPU-intensive data.
- Queuing work items in the background.
- Performing a time-based operation on a schedule.

Background service processing usually doesn't involve a user interface (UI), but UIs can be built around them. In the early days with .NET Framework, Windows developers could create Windows Services for these reasons.

Now with .NET, you can use the [BackgroundService](#), which is an implementation of [IHostedService](#), or implement your own.

With .NET, you're no longer restricted to Windows. You can develop cross-platform background services. Hosted services are logging, configuration, and dependency injection (DI) ready. They're a part of the extensions suite of libraries, meaning they're fundamental to all .NET workloads that work with the [generic host](#).

IMPORTANT

Installing the .NET SDK also installs the [Microsoft.NET.Sdk.Worker](#) and the worker template. In other words, after installing the .NET SDK, you could create a new worker by using the `dotnet new worker` command. If you're using Visual Studio, the template is hidden until the optional ASP.NET and web development workload is installed.

Terminology

Many terms are mistakenly used synonymously. In this section, there are definitions for some of these terms to make their intent more apparent.

- **Background Service:** Refers to the [BackgroundService](#) type.
- **Hosted Service:** Implementations of [IHostedService](#), or referring to the [IHostedService](#) itself.
- **Long-running Service:** Any service that runs continuously.
- **Windows Service:** The *Windows Service* infrastructure, originally .NET Framework centric but now accessible via .NET.
- **Worker Service:** Refers to the *Worker Service* template.

Worker Service template

The Worker Service template is available to the .NET CLI, and Visual Studio. For more information, see [.NET CLI](#), [dotnet new worker - template](#). The template consists of a [Program](#) and [Worker](#) class.

```
using App.WorkerService;

IHost host = Host.CreateDefaultBuilder(args)
    .ConfigureServices(services =>
{
    services.AddHostedService<Worker>();
})
.Build();

await host.RunAsync();
```

The preceding `Program` class:

- Creates the default `IHostBuilder`.
- Calls `ConfigureServices` to add the `Worker` class as a hosted service with `AddHostedService`.
- Builds an `IHost` from the builder.
- Calls `Run` on the `host` instance, which runs the app.

TIP

By default the Worker Service template doesn't enable server garbage collection (GC). All of the scenarios that require long-running services should consider performance implications of this default. To enable server GC, add the `ServerGarbageCollection` node to the project file:

```
<PropertyGroup>
    <ServerGarbageCollection>true</ServerGarbageCollection>
</PropertyGroup>
```

For more information regarding performance considerations, see [Server GC](#). For more information on configuring server GC, see [Server GC configuration examples](#).

The `Program.cs` file from the template can be rewritten using top-level statements:

```
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using App.WorkerService;

using IHost host = Host.CreateDefaultBuilder(args)
    .ConfigureServices((hostContext, services) =>
{
    services.AddHostedService<Worker>();
})
.Build();

await host.RunAsync();
```

This is functionally equivalent to the original template. For more information on C# 9 features, see [What's new in C# 9.0](#).

As for the `Worker`, the template provides a simple implementation.

```

namespace App.WorkerService
{
    public class Worker : BackgroundService
    {
        private readonly ILogger<Worker> _logger;

        public Worker(ILogger<Worker> logger)
        {
            _logger = logger;
        }

        protected override async Task ExecuteAsync(CancellationToken stoppingToken)
        {
            while (!stoppingToken.IsCancellationRequested)
            {
                _logger.LogInformation("Worker running at: {time}", DateTimeOffset.Now);
                await Task.Delay(1000, stoppingToken);
            }
        }
    }
}

```

The preceding `Worker` class is a subclass of `BackgroundService`, which implements `IHostedService`. The `BackgroundService` is an `abstract class` and requires the subclass to implement `BackgroundService.ExecuteAsync(CancellationToken)`. In the template implementation the `ExecuteAsync` loops once per second, logging the current date and time until the process is signaled to cancel.

The project file

The Worker Service template relies on the following project file `Sdk`:

```
<Project Sdk="Microsoft.NET.Sdk.Worker">
```

For more information, see [.NET project SDKs](#).

NuGet package

An app based on the Worker Service template uses the `Microsoft.NET.Sdk.Worker` SDK and has an explicit package reference to the `Microsoft.Extensions.Hosting` package.

Containers and cloud adaptability

With most modern .NET workloads, containers are a viable option. When creating a long-running service from the Worker Service template in Visual Studio, you can opt-in to **Docker support**. Doing so will create a `Dockerfile` that will containerize your .NET app. A `Dockerfile` is a set of instructions to build an image. For .NET apps, the `Dockerfile` usually sits in the root of the directory next to a solution file.

```

# See https://aka.ms/containerfastmode to understand how Visual Studio uses this
# Dockerfile to build your images for faster debugging.

FROM mcr.microsoft.com/dotnet/runtime:6.0 AS base
WORKDIR /app

FROM mcr.microsoft.com/dotnet/sdk:6.0 AS build
WORKDIR /src
COPY ["background-service/App.WorkerService.csproj", "background-service/"]
RUN dotnet restore "background-service/App.WorkerService.csproj"
COPY . .
WORKDIR "/src/background-service"
RUN dotnet build "App.WorkerService.csproj" -c Release -o /app/build

FROM build AS publish
RUN dotnet publish "App.WorkerService.csproj" -c Release -o /app/publish

FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "App.WorkerService.dll"]

```

The preceding *Dockerfile* steps include:

- Setting the base image from `mcr.microsoft.com/dotnet/runtime:6.0` as the alias `base`.
- Changing the working directory to `/app`.
- Setting the `build` alias from the `mcr.microsoft.com/dotnet/sdk:6.0` image.
- Changing the working directory to `/src`.
- Copying the contents and publishing the .NET app:
 - The app is published using the `dotnet publish` command.
- Relayering the .NET SDK image from `mcr.microsoft.com/dotnet/runtime:6.0` (the `base` alias).
- Copying the published build output from the `/publish`.
- Defining the entry point, which delegates to `dotnet App.BackgroundService.dll`.

TIP

The MCR in `mcr.microsoft.com` stands for "Microsoft Container Registry", and is Microsoft's syndicated container catalog from the official Docker hub. The [Microsoft syndicates container catalog](#) article contains additional details.

When targeting Docker as a deployment strategy for your .NET Worker Service, there are a few considerations in the project file:

```

<Project Sdk="Microsoft.NET.Sdk.Worker">

  <PropertyGroup>
    <TargetFramework>net6.0</TargetFramework>
    <Nullable>enable</Nullable>
    <ImplicitUsings>true</ImplicitUsings>
    <RootNamespace>App.WorkerService</RootNamespace>
    <DockerDefaultTargetOS>Linux</DockerDefaultTargetOS>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.Extensions.Hosting" Version="6.0.0" />
    <PackageReference Include="Microsoft.VisualStudio.Azure.Containers.Tools.Targets" Version="1.11.1" />
  </ItemGroup>
</Project>

```

In the preceding project file, the `<DockerDefaultTargetOS>` element specifies `Linux` as its target. To target Windows containers, use `Windows` instead. The [Microsoft.VisualStudio.Azure.Containers.Tools.Targets](#) NuGet package is automatically added as a package reference when Docker support is selected from the template.

For more information on Docker with .NET, see [Tutorial: Containerize a .NET app](#). For more information on deploying to Azure, see [Tutorial: Deploy a Worker Service to Azure](#).

IMPORTANT

If you want to leverage *User Secrets* with the Worker Service template, you'd have to explicitly reference the [Microsoft.Extensions.Configuration.UserSecrets](#) NuGet package.

Hosted Service extensibility

The [IHostedService](#) interface defines two methods:

- [IHostedService.StartAsync\(CancellationToken\)](#)
- [IHostedService.StopAsync\(CancellationToken\)](#)

These two methods serve as *lifecycle* methods - they're called during host start and stop events respectively.

NOTE

When overriding either [StartAsync](#) or [StopAsync](#) methods, you must call and `await` the `base` class method to ensure the service starts and/or shuts down properly.

IMPORTANT

The interface serves as a generic-type parameter constraint on the [AddHostedService<THostedService>\(!ServiceCollection\)](#) extension method, meaning only implementations are permitted. You're free to use the provided [BackgroundService](#) with a subclass, or implement your own entirely.

Signal completion

In most common scenarios, you do not need to explicitly signal the completion of a hosted service. When the host starts the services, they're designed to run until the host is stopped. In some scenarios, however, you may need to signal the completion of the entire host application when the service completes. To achieve this, consider the following `Worker` class:

```

namespace App.SignalCompletionService;

public class Worker : BackgroundService
{
    private readonly IHostApplicationLifetime _hostApplicationLifetime;
    private readonly ILogger<Worker> _logger;

    public Worker(
        IHostApplicationLifetime hostApplicationLifetime, ILogger<Worker> logger) =>
        (_hostApplicationLifetime, _logger) = (hostApplicationLifetime, logger);

    protected override async Task ExecuteAsync(CancellationToken stoppingToken)
    {
        // TODO: implement single execution logic here.
        _logger.LogInformation(
            "Worker running at: {time}", DateTimeOffset.Now);

        await Task.Delay(1000, stoppingToken);

        // When completed, the entire app host will stop.
        _hostApplicationLifetime.StopApplication();
    }
}

```

In the preceding code, the `ExecuteAsync` method doesn't loop, and when it's complete it calls `IHostApplicationLifetime.StopApplication()`.

IMPORTANT

This will signal to the host that it should stop, and without this call to `StopApplication` the host will continue to run indefinitely.

For more information, see:

- [.NET Generic Host: IHostApplicationLifetime](#)
- [.NET Generic Host: Host shutdown](#)
- [.NET Generic Host: Hosting shutdown process](#)

See also

- [BackgroundService](#) subclass tutorials:
 - [Create a Queue Service in .NET](#)
 - [Use scoped services within a `BackgroundService` in .NET](#)
 - [Create a Windows Service using `BackgroundService` in .NET](#)
- Custom [IHostedService](#) implementation:
 - [Implement the `IHostedService` interface in .NET](#)

Create a Queue Service

9/20/2022 • 5 minutes to read • [Edit Online](#)

A queue service is a great example of a long-running service, where work items can be queued and worked on sequentially as previous work items are completed. Relying on the Worker Service template, you'll build out new functionality on top of the [BackgroundService](#).

In this tutorial, you learn how to:

- Create a queue service.
- Delegate work to a task queue.
- Register a console key-listener from [IHostApplicationLifetime](#) events.

TIP

All of the "Workers in .NET" example source code is available in the [Samples Browser](#) for download. For more information, see [Browse code samples: Workers in .NET](#).

Prerequisites

- The [.NET 5.0 SDK or later](#)
- A .NET integrated development environment (IDE)
 - Feel free to use [Visual Studio](#)

Create a new project

To create a new Worker Service project with Visual Studio, you'd select **File > New > Project....** From the **Create a new project** dialog search for "Worker Service", and select Worker Service template. If you'd rather use the .NET CLI, open your favorite terminal in a working directory. Run the `dotnet new` command, and replace the `<Project.Name>` with your desired project name.

```
dotnet new worker --name <Project.Name>
```

For more information on the .NET CLI new worker service project command, see [dotnet new worker](#).

TIP

If you're using Visual Studio Code, you can run .NET CLI commands from the integrated terminal. For more information, see [Visual Studio Code: Integrated Terminal](#).

Create queuing services

You may be familiar with the [QueueBackgroundWorkItem\(Func<CancellationToken,Task>\)](#) functionality from the `System.Web.Hosting` namespace. To model a service that is inspired by this functionality, start by adding an `IBackgroundTaskQueue` interface to the project:

```

namespace App.QueueService;

public interface IBackgroundTaskQueue
{
    ValueTask QueueBackgroundWorkItemAsync(
        Func< CancellationToken, ValueTask> workItem);

    ValueTask<Func< CancellationToken, ValueTask>> DequeueAsync(
        CancellationToken cancellationToken);
}

```

There are two methods, one that exposes queuing functionality, and another that dequeues previously queued work items. A *work item* is a `Func< CancellationToken, ValueTask>`. Next, add the default implementation to the project.

```

using System.Threading.Channels;

namespace App.QueueService;

public class DefaultBackgroundTaskQueue : IBackgroundTaskQueue
{
    private readonly Channel<Func< CancellationToken, ValueTask>> _queue;

    public DefaultBackgroundTaskQueue(int capacity)
    {
        BoundedChannelOptions options = new(capacity)
        {
            FullMode = BoundedChannelFullMode.Wait
        };
        _queue = Channel.CreateBounded<Func< CancellationToken, ValueTask>>(options);
    }

    public async ValueTask QueueBackgroundWorkItemAsync(
        Func< CancellationToken, ValueTask> workItem)
    {
        if (workItem is null)
        {
            throw new ArgumentNullException(nameof(workItem));
        }

        await _queue.Writer.WriteAsync(workItem);
    }

    public async ValueTask<Func< CancellationToken, ValueTask>> DequeueAsync(
        CancellationToken cancellationToken)
    {
        Func< CancellationToken, ValueTask>? workItem =
            await _queue.Reader.ReadAsync(cancellationToken);

        return workItem;
    }
}

```

The preceding implementation relies on a `Channel<T>` as a queue. The `BoundedChannelOptions(Int32)` is called with an explicit capacity. Capacity should be set based on the expected application load and number of concurrent threads accessing the queue. `BoundedChannelFullMode.Wait` will cause calls to `ChannelWriter<T>.WriteAsync` to return a task, which completes only when space becomes available. This leads to backpressure, in case too many publishers/calls start accumulating.

Rewrite the Worker class

In the following `QueueHostedService` example:

- The `ProcessTaskQueueAsync` method returns a `Task` in `ExecuteAsync`.
- Background tasks in the queue are dequeued and executed in `ProcessTaskQueueAsync`.
- Work items are awaited before the service stops in `StopAsync`.

Replace the existing `Worker` class with the following C# code, and rename the file to `QueueHostedService.cs`.

```
namespace App.QueueService;

public sealed class QueuedHostedService : BackgroundService
{
    private readonly IBackgroundTaskQueue _taskQueue;
    private readonly ILogger<QueuedHostedService> _logger;

    public QueuedHostedService(
        IBackgroundTaskQueue taskQueue,
        ILogger<QueuedHostedService> logger) =>
        (_taskQueue, _logger) = (taskQueue, logger);

    protected override Task ExecuteAsync(CancellationToken stoppingToken)
    {
        _logger.LogInformation(
            $"{nameof(QueuedHostedService)} is running.{Environment.NewLine}" +
            $"{Environment.NewLine}Tap W to add a work item to the " +
            $"background queue.{Environment.NewLine}");

        return ProcessTaskQueueAsync(stoppingToken);
    }

    private async Task ProcessTaskQueueAsync(CancellationToken stoppingToken)
    {
        while (!stoppingToken.IsCancellationRequested)
        {
            try
            {
                Func< CancellationToken, ValueTask>? workItem =
                    await _taskQueue.DequeueAsync(stoppingToken);

                await workItem(stoppingToken);
            }
            catch (OperationCanceledException)
            {
                // Prevent throwing if stoppingToken was signaled
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, "Error occurred executing task work item.");
            }
        }
    }

    public override async Task StopAsync(CancellationToken stoppingToken)
    {
        _logger.LogInformation(
            $"{nameof(QueuedHostedService)} is stopping.");

        await base.StopAsync(stoppingToken);
    }
}
```

A `MonitorLoop` service handles enqueueing tasks for the hosted service whenever the `w` key is selected on an input device:

- The `IBackgroundTaskQueue` is injected into the `MonitorLoop` service.
- `IBackgroundTaskQueue.QueueBackgroundWorkItemAsync` is called to enqueue a work item.

- The work item simulates a long-running background task:
 - Three 5-second delays are executed [Delay](#).
 - A `try-catch` statement traps [OperationCanceledException](#) if the task is canceled.

```

namespace App.QueueService;

public class MonitorLoop
{
    private readonly IBackgroundTaskQueue _taskQueue;
    private readonly ILogger<MonitorLoop> _logger;
    private readonly CancellationToken _cancellationToken;

    public MonitorLoop(
        IBackgroundTaskQueue taskQueue,
        ILogger<MonitorLoop> logger,
        IHostApplicationLifetime applicationLifetime)
    {
        _taskQueue = taskQueue;
        _logger = logger;
        _cancellationToken = applicationLifetime.ApplicationStopping;
    }

    public void StartMonitorLoop()
    {
        _logger.LogInformation($"'{nameof(MonitorAsync)}' loop is starting.");

        // Run a console user input loop in a background thread
        Task.Run(async () => await MonitorAsync());
    }

    private async ValueTask MonitorAsync()
    {
        while (!_cancellationToken.IsCancellationRequested)
        {
            var keyStroke = Console.ReadKey();
            if (keyStroke.Key == ConsoleKey.W)
            {
                // Enqueue a background work item
                await _taskQueue.QueueBackgroundWorkItemAsync(BuildWorkItemAsync);
            }
        }
    }

    private async ValueTask BuildWorkItemAsync(CancellationToken token)
    {
        // Simulate three 5-second tasks to complete
        // for each enqueued work item

        int delayLoop = 0;
        var guid = Guid.NewGuid();

        _logger.LogInformation("Queued work item {Guid} is starting.", guid);

        while (!token.IsCancellationRequested && delayLoop < 3)
        {
            try
            {
                await Task.Delay(TimeSpan.FromSeconds(5), token);
            }
            catch (OperationCanceledException)
            {
                // Prevent throwing if the Delay is cancelled
            }

            ++delayLoop;

            _logger.LogInformation("Queued work item {Guid} is running. {DelayLoop}/{3}", guid, delayLoop);
        }
    }
}

```

```

    _logger.LogInformation($"Queued work item {guid} is running. {delayLoop}/{_, delayLoop}, {guid}, delayLoop");
}

string format = delayLoop switch
{
    3 => "Queued Background Task {Guid} is complete.",
    _ => "Queued Background Task {Guid} was cancelled."
};

_logger.LogInformation(format, guid);
}
}

```

Replace the existing `Program` contents with the following C# code:

```

using App.QueueService;

using IHost host = Host.CreateDefaultBuilder(args)
    .ConfigureServices((context, services) =>
{
    services.AddSingleton<MonitorLoop>();
    services.AddHostedService<QueuedHostedService>();
    services.AddSingleton<IBackgroundTaskQueue>(_ =>
{
        if (!int.TryParse(context.Configuration["QueueCapacity"], out var queueCapacity))
        {
            queueCapacity = 100;
        }

        return new DefaultBackgroundTaskQueue(queueCapacity);
    });
})
.Build();

MonitorLoop monitorLoop = host.Services.GetRequiredService<MonitorLoop>()!;
monitorLoop.StartMonitorLoop();

await host.RunAsync();

```

The services are registered in `IHostBuilder.ConfigureServices` (*Program.cs*). The hosted service is registered with the `AddHostedService` extension method. `MonitorLoop` is started in *Program.cs* top-level statement:

```

MonitorLoop monitorLoop = host.Services.GetRequiredService<MonitorLoop>()!;
monitorLoop.StartMonitorLoop();

```

For more information on registering services, see [Dependency injection in .NET](#).

Verify service functionality

To run the application from Visual Studio, select F5 or select the **Debug > Start Debugging** menu option. If you're using the .NET CLI, run the `dotnet run` command from the working directory:

```
dotnet run
```

For more information on the .NET CLI run command, see [dotnet run](#).

When prompted enter the `w` (or `w`) at least once to queue an emulated work item. You will see output similar to the following:

```
info: App.QueueService.MonitorLoop[0]
    MonitorAsync loop is starting.
info: App.QueueService.QueuedHostedService[0]
    QueuedHostedService is running.

    Tap W to add a work item to the background queue.

info: Microsoft.Hosting.Lifetime[0]
    Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
    Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
    Content root path: .\queue-service
winfo: App.QueueService.MonitorLoop[0]
    Queued work item 8453f845-ea4a-4bcb-b26e-c76c0d89303e is starting.
info: App.QueueService.MonitorLoop[0]
    Queued work item 8453f845-ea4a-4bcb-b26e-c76c0d89303e is running. 1/3
info: App.QueueService.MonitorLoop[0]
    Queued work item 8453f845-ea4a-4bcb-b26e-c76c0d89303e is running. 2/3
info: App.QueueService.MonitorLoop[0]
    Queued work item 8453f845-ea4a-4bcb-b26e-c76c0d89303e is running. 3/3
info: App.QueueService.MonitorLoop[0]
    Queued Background Task 8453f845-ea4a-4bcb-b26e-c76c0d89303e is complete.
info: Microsoft.Hosting.Lifetime[0]
    Application is shutting down...
info: App.QueueService.QueuedHostedService[0]
    QueuedHostedService is stopping.
```

If running the application from within Visual Studio, select **Debug > Stop Debugging**.... Alternatively, select **Ctrl + C** from the console window to signal cancellation.

See also

- [Worker Services in .NET](#)
- [Use scoped services within a `BackgroundService`](#)
- [Create a Windows Service using `BackgroundService`](#)
- [Implement the `IHostedService` interface](#)

Use scoped services within a `BackgroundService`

9/20/2022 • 3 minutes to read • [Edit Online](#)

When you register implementations of [IHostedService](#) using any of the [AddHostedService](#) extension methods - the service is registered as a singleton. There may be scenarios where you'd like to rely on a scoped service. For more information, see [Dependency injection in .NET: Service lifetimes](#).

In this tutorial, you learn how to:

- Resolve scoped dependencies in a singleton `BackgroundService`.
- Delegate work to a scoped service.
- Implement an `override` of `BackgroundService.StopAsync(CancellationToken)`.

TIP

All of the "Workers in .NET" example source code is available in the [Samples Browser](#) for download. For more information, see [Browse code samples: Workers in .NET](#).

Prerequisites

- The [.NET 5.0 SDK or later](#)
- A .NET integrated development environment (IDE)
 - Feel free to use [Visual Studio](#)

Create a new project

To create a new Worker Service project with Visual Studio, you'd select **File > New > Project....** From the **Create a new project** dialog search for "Worker Service", and select Worker Service template. If you'd rather use the .NET CLI, open your favorite terminal in a working directory. Run the `dotnet new` command, and replace the `<Project.Name>` with your desired project name.

```
dotnet new worker --name <Project.Name>
```

For more information on the .NET CLI new worker service project command, see [dotnet new worker](#).

TIP

If you're using Visual Studio Code, you can run .NET CLI commands from the integrated terminal. For more information, see [Visual Studio Code: Integrated Terminal](#).

Create scoped services

To use [scoped services](#) within a `BackgroundService`, create a scope. No scope is created for a hosted service by default. The scoped background service contains the background task's logic.

```

namespace App.ScopedService;

public interface IScopeProcessingService
{
    Task DoWorkAsync(CancellationToken stoppingToken);
}

```

The preceding interface defines a single `DoWorkAsync` method. To define the default implementation:

- The service is asynchronous. The `DoWorkAsync` method returns a `Task`. For demonstration purposes, a delay of ten seconds is awaited in the `DoWorkAsync` method.
- An `ILogger` is injected into the service.:

```

namespace App.ScopedService;

public class DefaultScopeProcessingService : IScopeProcessingService
{
    private int _executionCount;
    private readonly ILogger<DefaultScopeProcessingService> _logger;

    public DefaultScopeProcessingService(
        ILogger<DefaultScopeProcessingService> logger) =>
        _logger = logger;

    public async Task DoWorkAsync(CancellationToken stoppingToken)
    {
        while (!stoppingToken.IsCancellationRequested)
        {
            ++ _executionCount;

            _logger.LogInformation(
                "{ServiceName} working, execution count: {Count}",
                nameof(DefaultScopeProcessingService),
                _executionCount);

            await Task.Delay(10_000, stoppingToken);
        }
    }
}

```

The hosted service creates a scope to resolve the scoped background service to call its `DoWorkAsync` method. `DoWorkAsync` returns a `Task`, which is awaited in `ExecuteAsync` :

Rewrite the Worker class

Replace the existing `Worker` class with the following C# code, and rename the file to `ScopedBackgroundService.cs`.

```

namespace App.ScopedService;

public sealed class ScopedBackgroundService : BackgroundService
{
    private readonly IServiceProvider _serviceProvider;
    private readonly ILogger<ScopedBackgroundService> _logger;

    public ScopedBackgroundService(
        IServiceProvider serviceProvider,
        ILogger<ScopedBackgroundService> logger) =>
        (_serviceProvider, _logger) = (serviceProvider, logger);

    protected override async Task ExecuteAsync(CancellationToken stoppingToken)
    {
        _logger.LogInformation(
            $"{nameof(ScopedBackgroundService)} is running.");

        await DoWorkAsync(stoppingToken);
    }

    private async Task DoWorkAsync(CancellationToken stoppingToken)
    {
        _logger.LogInformation(
            $"{nameof(ScopedBackgroundService)} is working.");

        using (IServiceScope scope = _serviceProvider.CreateScope())
        {
            IScopedProcessingService scopedProcessingService =
                scope.ServiceProvider.GetRequiredService<IScopedProcessingService>();

            await scopedProcessingService.DoWorkAsync(stoppingToken);
        }
    }

    public override async Task StopAsync(CancellationToken stoppingToken)
    {
        _logger.LogInformation(
            $"{nameof(ScopedBackgroundService)} is stopping.");

        await base.StopAsync(stoppingToken);
    }
}

```

In the preceding code, an explicit scope is created and the `IScopedProcessingService` implementation is resolved from the dependency injection service provider. The resolved service instance is scoped, and its `DoWorkAsync` method is awaited.

Replace the template *Program.cs* file contents with the following C# code:

```

using App.ScopedService;

using IHost host = Host.CreateDefaultBuilder(args)
    .ConfigureServices(services =>
{
    services.AddHostedService<ScopedBackgroundService>();
    services.AddScoped<IScopedProcessingService, DefaultScopedProcessingService>();
})
.Build();

await host.RunAsync();

```

The services are registered in `IHostBuilder.ConfigureServices` (*Program.cs*). The hosted service is registered with the `AddHostedService` extension method.

For more information on registering services, see [Dependency injection in .NET](#).

Verify service functionality

To run the application from Visual Studio, select F5 or select the **Debug > Start Debugging** menu option. If you're using the .NET CLI, run the `dotnet run` command from the working directory:

```
dotnet run
```

For more information on the .NET CLI run command, see [dotnet run](#).

Let the application run for a bit to generate several execution count increments. You will see output similar to the following:

```
info: App.ScopedService.ScopedBackgroundService[0]
      ScopedBackgroundService is running.
info: App.ScopedService.ScopedBackgroundService[0]
      ScopedBackgroundService is working.
info: App.ScopedService.DefaultScopedProcessingService[0]
      DefaultScopedProcessingService working, execution count: 1
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
      Content root path: .\scoped-service
info: App.ScopedService.DefaultScopedProcessingService[0]
      DefaultScopedProcessingService working, execution count: 2
info: App.ScopedService.DefaultScopedProcessingService[0]
      DefaultScopedProcessingService working, execution count: 3
info: App.ScopedService.DefaultScopedProcessingService[0]
      DefaultScopedProcessingService working, execution count: 4
info: Microsoft.Hosting.Lifetime[0]
      Application is shutting down...
info: App.ScopedService.ScopedBackgroundService[0]
      ScopedBackgroundService is stopping.
```

If running the application from within Visual Studio, select **Debug > Stop Debugging**.... Alternatively, select **Ctrl + C** from the console window to signal cancellation.

See also

- [Worker Services in .NET](#)
- [Create a Queue Service](#)
- [Create a Windows Service using `BackgroundService`](#)
- [Implement the `IHostedService` interface](#)

Create a Windows Service using

BackgroundService

9/20/2022 • 13 minutes to read • [Edit Online](#)

.NET Framework developers are probably familiar with Windows Service apps. Before .NET Core and .NET 5+, developers who relied on .NET Framework could create Windows Services to perform background tasks or execute long-running processes. This functionality is still available and you can create Worker Services that run as a Windows Service.

In this tutorial, you'll learn how to:

- Publish a .NET worker app as a single file executable.
- Create a Windows Service.
- Create the `BackgroundService` app as a Windows Service.
- Start and stop the Windows Service.
- View event logs.
- Delete the Windows Service.

TIP

All of the "Workers in .NET" example source code is available in the [Samples Browser](#) for download. For more information, see [Browse code samples: Workers in .NET](#).

IMPORTANT

Installing the .NET SDK also installs the `Microsoft.NET.Sdk.Worker` and the worker template. In other words, after installing the .NET SDK, you could create a new worker by using the `dotnet new worker` command. If you're using Visual Studio, the template is hidden until the optional ASP.NET and web development workload is installed.

Prerequisites

- The [.NET 6.0 SDK or later](#)
- A Windows OS
- A .NET integrated development environment (IDE)
 - Feel free to use [Visual Studio](#)

Create a new project

To create a new Worker Service project with Visual Studio, you'd select **File > New > Project....** From the **Create a new project** dialog search for "Worker Service", and select Worker Service template. If you'd rather use the .NET CLI, open your favorite terminal in a working directory. Run the `dotnet new` command, and replace the `<Project.Name>` with your desired project name.

```
dotnet new worker --name <Project.Name>
```

For more information on the .NET CLI new worker service project command, see [dotnet new worker](#).

TIP

If you're using Visual Studio Code, you can run .NET CLI commands from the integrated terminal. For more information, see [Visual Studio Code: Integrated Terminal](#).

Install NuGet package

To interop with native Windows Services from .NET [IHostedService](#) implementations, you'll need to install the [Microsoft.Extensions.Hosting.WindowsServices](#) NuGet package.

To install this from Visual Studio, use the **Manage NuGet Packages...** dialog. Search for "Microsoft.Extensions.Hosting.WindowsServices", and install it. If you'd rather use the .NET CLI, run the `dotnet add package` command:

```
dotnet add package Microsoft.Extensions.Hosting.WindowsServices
```

For more information on the .NET CLI add package command, see [dotnet add package](#).

After successfully adding the packages, your project file should now contain the following package references:

```
<ItemGroup>
  <PackageReference Include="Microsoft.Extensions.Hosting" Version="6.0.0" />
  <PackageReference Include="Microsoft.Extensions.Hosting.WindowsServices" Version="6.0.0" />
</ItemGroup>
</Project>
```

Update project file

This worker project makes use of C#'s [nullable reference types](#). To enable them for the entire project, update the project file accordingly:

```
<Project Sdk="Microsoft.NET.Sdk.Worker">

  <PropertyGroup>
    <TargetFramework>net6.0</TargetFramework>
    <Nullable>enable</Nullable>
    <ImplicitUsings>true</ImplicitUsings>
    <RootNamespace>App.WindowsService</RootNamespace>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.Extensions.Hosting" Version="6.0.0" />
    <PackageReference Include="Microsoft.Extensions.Hosting.WindowsServices" Version="6.0.0" />
  </ItemGroup>
</Project>
```

The preceding project file changes add the `<Nullable>enable</Nullable>` node. For more information, see [Setting the nullable context](#).

Create the service

Add a new class to the project named *JokeService.cs*, and replace its contents with the following C# code:

```

namespace App.WindowsService;

public class JokeService
{
    public string GetJoke()
    {
        Joke joke = _jokes.ElementAt(
            Random.Shared.Next(_jokes.Count));

        return $"{joke.Setup}{Environment.NewLine}{joke.Punchline}";
    }

    // Programming jokes borrowed from:
    // https://github.com/eklavyadev/karljoke/blob/main/source/jokes.json
    readonly HashSet<Joke> _jokes = new()
    {
        new Joke("What's the best thing about a Boolean?", "Even if you're wrong, you're only off by a bit."),
        new Joke("What's the object-oriented way to become wealthy?", "Inheritance"),
        new Joke("Why did the programmer quit their job?", "Because they didn't get arrays."),
        new Joke("Why do programmers always mix up Halloween and Christmas?", "Because Oct 31 == Dec 25"),
        new Joke("How many programmers does it take to change a lightbulb?", "None that's a hardware problem"),
        new Joke("If you put a million monkeys at a million keyboards, one of them will eventually write a Java program", "the rest of them will write Perl"),
        new Joke("[['hip', 'hip']]", "(hip hip array)"),
        new Joke("To understand what recursion is...", "You must first understand what recursion is"),
        new Joke("There are 10 types of people in this world...", "Those who understand binary and those who don't"),
        new Joke("Which song would an exception sing?", "Can't catch me - Avicii"),
        new Joke("Why do Java programmers wear glasses?", "Because they don't C#"),
        new Joke("How do you check if a webpage is HTML5?", "Try it out on Internet Explorer"),
        new Joke("A user interface is like a joke.", "If you have to explain it then it is not that good."),
        new Joke("I was gonna tell you a joke about UDP...", "...but you might not get it."),
        new Joke("The punchline often arrives before the set-up.", "Do you know the problem with UDP jokes?"),
        new Joke("Why do C# and Java developers keep breaking their keyboards?", "Because they use a strongly typed language."),
        new Joke("Knock-knock.", "A race condition. Who is there?"),
        new Joke("What's the best part about TCP jokes?", "I get to keep telling them until you get them."),
        new Joke("A programmer puts two glasses on their bedside table before going to sleep.", "A full one, in case they gets thirsty, and an empty one, in case they don't."),
        new Joke("There are 10 kinds of people in this world.", "Those who understand binary, those who don't, and those who weren't expecting a base 3 joke."),
        new Joke("What did the router say to the doctor?", "It hurts when IP."),
        new Joke("An IPv6 packet is walking out of the house.", "He goes nowhere."),
        new Joke("3 SQL statements walk into a NoSQL bar. Soon, they walk out", "They couldn't find a table.")
    };
}

public record Joke(string Setup, string Punchline);

```

The preceding joke service source code exposes a single piece of functionality, the `GetJoke` method. This is a `string` returning method that represents a random programming joke. The class-scoped `_jokes` field is used to store the list of jokes. A random joke is selected from the list and returned.

Rewrite the `Worker` class

Replace the existing `Worker` from the template with the following C# code, and rename the file to `WindowsBackgroundService.cs`.

```

namespace App.WindowsService;

public sealed class WindowsBackgroundService : BackgroundService
{
    private readonly JokeService _jokeService;
    private readonly ILogger<WindowsBackgroundService> _logger;

    public WindowsBackgroundService(
        JokeService jokeService,
        ILogger<WindowsBackgroundService> logger) =>
        (_jokeService, _logger) = (jokeService, logger);

    protected override async Task ExecuteAsync(CancellationToken stoppingToken)
    {
        try
        {
            while (!stoppingToken.IsCancellationRequested)
            {
                string joke = _jokeService.GetJoke();
                _logger.LogWarning("{Joke}", joke);

                await Task.Delay(TimeSpan.FromMinutes(1), stoppingToken);
            }
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, "{Message}", ex.Message);

            // Terminates this process and returns an exit code to the operating system.
            // This is required to avoid the 'BackgroundServiceExceptionBehavior', which
            // performs one of two scenarios:
            // 1. When set to "Ignore": will do nothing at all, errors cause zombie services.
            // 2. When set to "StopHost": will cleanly stop the host, and log errors.
            //
            // In order for the Windows Service Management system to leverage configured
            // recovery options, we need to terminate the process with a non-zero exit code.
            Environment.Exit(1);
        }
    }
}

```

In the preceding code, the `JokeService` is injected along with an `ILogger`. Both are made available to the class as `private readonly` fields. In the `ExecuteAsync` method, the joke service requests a joke and writes it to the logger. In this case, the logger is implemented by the Windows Event Log - [Microsoft.Extensions.Logging.EventLog.EventLogLogger](#). Logs are written to, and available for viewing in the [Event Viewer](#).

NOTE

By default, the *Event Log* severity is [Warning](#). This can be configured, but for demonstration purposes the `WindowsBackgroundService` logs with the [LogWarning](#) extension method. To specifically target the `EventLog` level, add an entry in the `appsettings.{Environment}.json`, or provide an `EventLogSettings.Filter` value.

```
{  
    "Logging": {  
        "LogLevel": {  
            "Default": "Warning"  
        },  
        "EventLog": {  
            "SourceName": "The Joke Service",  
            "LogName": "Application",  
            "LogLevel": {  
                "Microsoft": "Information",  
                "Microsoft.Hosting.Lifetime": "Information"  
            }  
        }  
    }  
}
```

For more information on configuring log levels, see [Logging providers in .NET: Configure Windows EventLog](#).

Rewrite the `Program` class

Replace the template `Program.cs` file contents with the following C# code:

```
using App.WindowsService;  
using Microsoft.Extensions.Logging.Configuration;  
using Microsoft.Extensions.Logging.EventLog;  
  
using IHost host = Host.CreateDefaultBuilder(args)  
    .UseWindowsService(options =>  
    {  
        options.ServiceName = ".NET Joke Service";  
    })  
    .ConfigureServices(services =>  
    {  
        LoggerProviderOptions.RegisterProviderOptions<  
            EventLogSettings, EventLogLoggerProvider>(services);  
  
        services.AddSingleton<JokeService>();  
        services.AddHostedService<WindowsBackgroundService>();  
    })  
    .ConfigureLogging((context, logging) =>  
    {  
        // See: https://github.com/dotnet/runtime/issues/47303  
        logging.AddConfiguration(  
            context.Configuration.GetSection("Logging"));  
    })  
    .Build();  
  
await host.RunAsync();
```

The [UseWindowsService](#) extension method configures the app to work as a Windows Service. The service name is set to `".NET Joke Service"`. The hosted service is registered for dependency injection.

For more information on registering services, see [Dependency injection in .NET](#).

Publish the app

To create the .NET Worker Service app as a Windows Service, it's recommended that you publish the app as a single file executable. It's less error-prone to have a self-contained executable, as there aren't any dependent files lying around the file system. But you may choose a different publishing modality, which is perfectly acceptable, so long as you create an *.exe file that can be targeted by the Windows Service Control Manager.

IMPORTANT

An alternative publishing approach is to build the *.dll (instead of an *.exe), and when you install the published app using the Windows Service Control Manager you delegate to the .NET CLI and pass the DLL. For more information, see [.NET CLI: dotnet command](#).

```
sc.exe create ".NET Joke Service" binpath="C:\Path\To\dotnet.exe C:\Path\To\App.WindowsService.dll"
```

```
<Project Sdk="Microsoft.NET.Sdk.Worker">

<PropertyGroup>
  <TargetFramework>net6.0</TargetFramework>
  <Nullable>enable</Nullable>
  <ImplicitUsings>true</ImplicitUsings>
  <RootNamespace>App.WindowsService</RootNamespace>
  <OutputType>exe</OutputType>
  <PublishSingleFile Condition="@(Configuration) == 'Release'">true</PublishSingleFile>
  <RuntimeIdentifier>win-x64</RuntimeIdentifier>
  <PlatformTarget>x64</PlatformTarget>
</PropertyGroup>

<ItemGroup>
  <PackageReference Include="Microsoft.Extensions.Hosting" Version="6.0.0" />
  <PackageReference Include="Microsoft.Extensions.Hosting.WindowsServices" Version="6.0.0" />
</ItemGroup>
</Project>
```

The preceding highlighted lines of the project file define the following behaviors:

- `<OutputType>exe</OutputType>` : Creates a console application.
- `<PublishSingleFile Condition="@(Configuration) == 'Release'">true</PublishSingleFile>` : Enables single-file publishing.
- `<RuntimeIdentifier>win-x64</RuntimeIdentifier>` : Specifies the RID of `win-x64`.
- `<PlatformTarget>x64</PlatformTarget>` : Specify the target platform CPU of 64-bit.

To publish the app from Visual Studio, you can create a publish profile that is persisted. The publish profile is XML-based, and has the `.pubxml` file extension. Visual Studio uses this profile to publish the app implicitly, whereas if you're using the .NET CLI — you must explicitly specify the publish profile for it to be used.

Right-click on the project in the **Solution Explorer**, and select **Publish....** Then, select **Add a publish profile** to create a profile. From the **Publish** dialog, select **Folder** as your **Target**.

Publish

Where are you publishing today?

Target

Azure
Publish your application to the Microsoft cloud

Docker Container Registry
Publish your application to any supported Container Registry that works with Docker images

Folder
Publish your application to a local folder or file share

Import Profile
Import your publish settings to deploy your app

Back Next Finish Cancel

Leave the default **Location**, and then select **Finish**. Once the profile is created, select **Show all settings**, and verify your **Profile settings**.

Profile settings

Profile name: FolderProfile

Configuration: Release | Any CPU

Target framework: net6.0

Deployment mode: Self-contained

Target runtime: win-x64

Target location: bin\Release\net6.0\win-x64\publish\ ...

File publish options

Produce single file

Enable ReadyToRun compilation

Trim unused code

Save Cancel

Ensure that the following settings are specified:

- **Deployment mode:** Self-contained
- **Produce single file:** checked
- **Enable ReadyToRun compilation:** checked
- **Trim unused assemblies (in preview):** unchecked

Finally, select **Publish**. The app is compiled, and the resulting .exe file is published to the `/publish` output directory.

Alternatively, you could use the .NET CLI to publish the app:

```
dotnet publish --output "C:\custom\publish\directory"
```

For more information, see [dotnet publish](#).

IMPORTANT

With .NET 6, if you attempt to debug the app with the `<PublishSingleFile>true</PublishSingleFile>` setting, you will not be able to debug the app. For more information, see [Unable to attach to CoreCLR when debugging a 'PublishSingleFile' .NET 6 app](#).

Create the Windows Service

To create the Windows Service, use the native Windows Service Control Manager's (sc.exe) create command. Run PowerShell as an Administrator.

```
sc.exe create ".NET Joke Service" binpath="C:\Path\To\App.WindowsService.exe"
```

TIP

If you need to change the content root of the [host configuration](#), you can pass it as a command-line argument when specifying the `binpath`:

```
sc.exe create "Svc Name" binpath="C:\Path\To\App.exe --contentRoot C:\Other\Path"
```

You'll see an output message:

```
[SC] CreateService SUCCESS
```

For more information, see [sc.exe create](#).

Configure the Windows Service

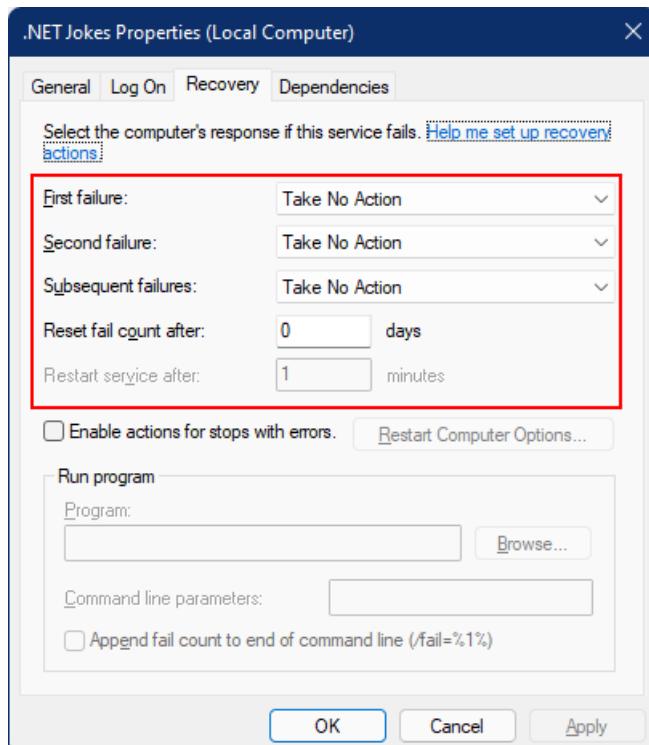
After the service is created, you can optionally configure it. If you're fine with the service defaults, skip to the [Verify service functionality](#) section.

Windows Services provide recovery configuration options. You can query the current configuration using the `sc.exe qfailure "<Service Name>"` (where `<Service Name>` is your services' name) command to read the current recovery configuration values:

```
sc qfailure ".NET Joke Service"
[SC] QueryServiceConfig2 SUCCESS

SERVICE_NAME: .NET Joke Service
    RESET_PERIOD (in seconds)      : 0
    REBOOT_MESSAGE                 :
    COMMAND_LINE                   :
```

The command will output the recovery configuration, which is the default values—since they've not yet been configured.



To configure recovery, use the `sc.exe failure "<Service Name>"` where `<Service Name>` is the name of your service:

```
sc.exe failure ".NET Joke Service" reset=0 actions=restart/60000/restart/60000/run/1000
[SC] ChangeServiceConfig2 SUCCESS
```

TIP

To configure the recovery options, your terminal session needs to run as an Administrator.

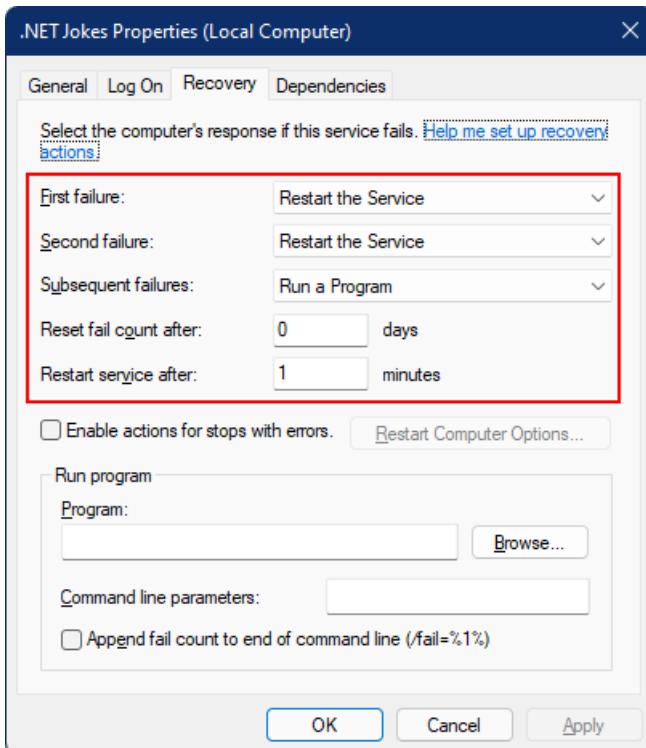
After it's been successfully configured, you can query the values once again using the

```
sc.exe qfailure "<Service Name>"
```

```
sc qfailure ".NET Joke Service"
[SC] QueryServiceConfig2 SUCCESS

SERVICE_NAME: .NET Joke Service
    RESET_PERIOD (in seconds)      : 0
    REBOOT_MESSAGE                 :
    COMMAND_LINE                   :
    FAILURE_ACTIONS                : RESTART -- Delay = 60000 milliseconds.
                                         RESTART -- Delay = 60000 milliseconds.
                                         RUN PROCESS -- Delay = 1000 milliseconds.
```

You will see the configured restart values.



Service recovery options and .NET `BackgroundService` instances

With .NET 6, [new hosting exception handling behaviors](#) have been added to .NET. The `BackgroundServiceExceptionBehavior` enum was added to the `Microsoft.Extensions.Hosting` namespace, and is used to specify the behavior of the service when an exception is thrown. The following table lists the available options:

OPTION	DESCRIPTION
<code>Ignore</code>	Ignore exceptions thrown in <code>BackgroundService</code> .
<code>StopHost</code>	The <code>IHost</code> will be stopped when an unhandled exception is thrown.

The default behavior before .NET 6 is `Ignore`, which resulted in *zombie processes* (a running process that didn't do anything). With .NET 6, the default behavior is `StopHost`, which results in the host being stopped when an exception is thrown. But it stops cleanly, meaning that the Windows Service management system will not restart the service. To correctly allow the service to be restarted, you can call `Environment.Exit` with a non-zero exit code. Consider the following highlighted `catch` block:

```

namespace App.WindowsService;

public sealed class WindowsBackgroundService : BackgroundService
{
    private readonly JokeService _jokeService;
    private readonly ILogger<WindowsBackgroundService> _logger;

    public WindowsBackgroundService(
        JokeService jokeService,
        ILogger<WindowsBackgroundService> logger) =>
        (_jokeService, _logger) = (jokeService, logger);

    protected override async Task ExecuteAsync(CancellationToken stoppingToken)
    {
        try
        {
            while (!stoppingToken.IsCancellationRequested)
            {
                string joke = _jokeService.GetJoke();
                _logger.LogWarning("{Joke}", joke);

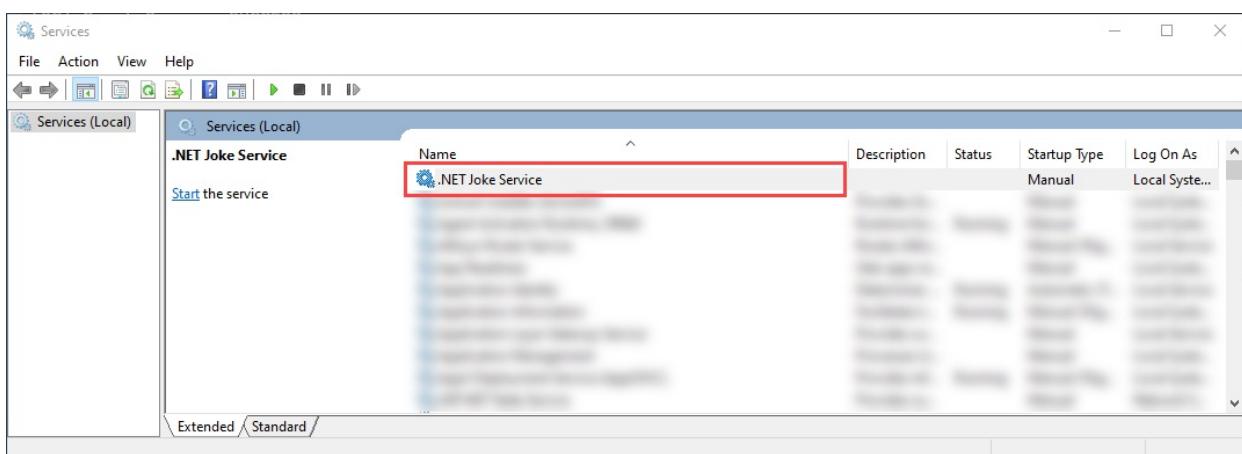
                await Task.Delay(TimeSpan.FromMinutes(1), stoppingToken);
            }
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, "{Message}", ex.Message);

            // Terminates this process and returns an exit code to the operating system.
            // This is required to avoid the 'BackgroundServiceExceptionBehavior', which
            // performs one of two scenarios:
            // 1. When set to "Ignore": will do nothing at all, errors cause zombie services.
            // 2. When set to "StopHost": will cleanly stop the host, and log errors.
            //
            // In order for the Windows Service Management system to leverage configured
            // recovery options, we need to terminate the process with a non-zero exit code.
            Environment.Exit(1);
        }
    }
}

```

Verify service functionality

To see the app created as a Windows Service, open **Services**. Select the Windows key (or **Ctrl + Esc**), and search from "Services". From the **Services** app, you should be able to find your service by its name.



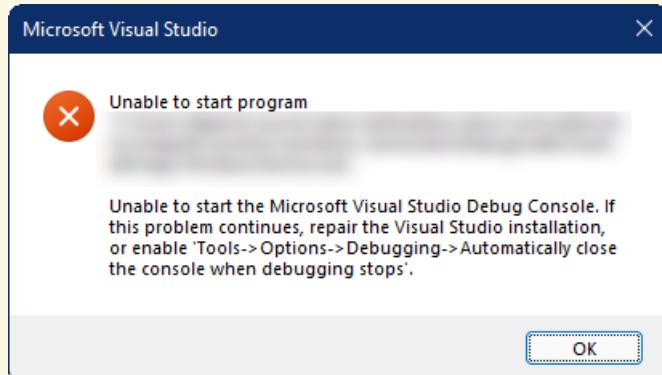
To verify that the service is functioning as expected, you need to:

- Start the service

- View the logs
- Stop the service

IMPORTANT

To debug the application, ensure that you're *not* attempting to debug the executable that is actively running within the Windows Services process.



Start the Windows Service

To start the Windows Service, use the `sc.exe start` command:

```
sc.exe start ".NET Joke Service"
```

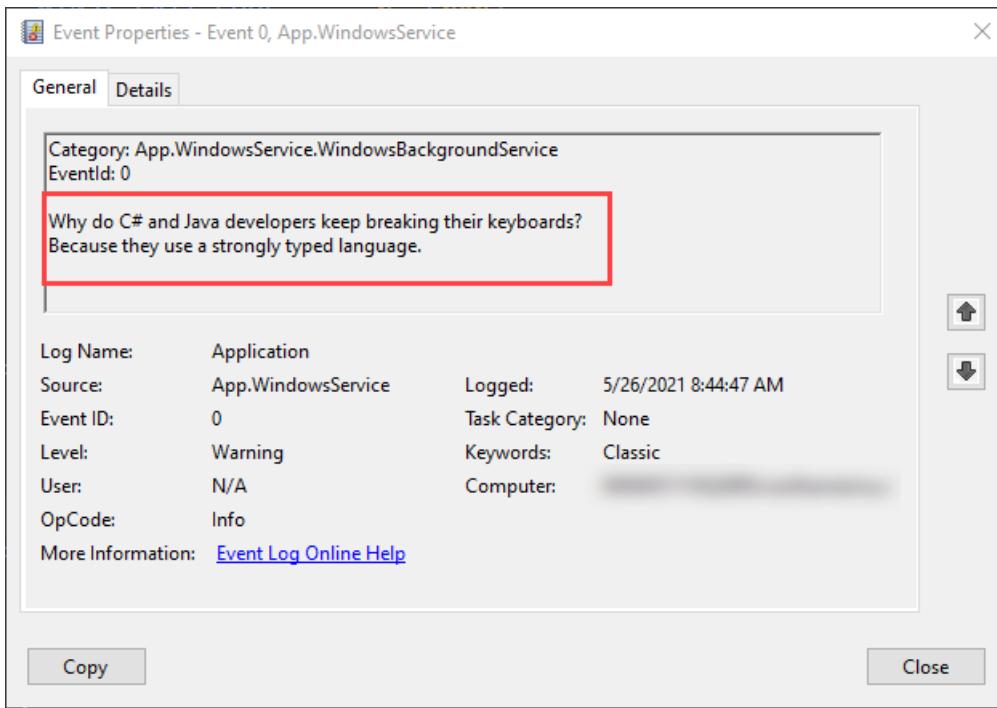
You'll see output similar to the following:

```
SERVICE_NAME: .NET Joke Service
TYPE          : 10  WIN32_OWN_PROCESS
STATE         : 2   START_PENDING
               (NOT_STOPPABLE, NOT_PAUSABLE, IGNORES_SHUTDOWN)
WIN32_EXIT_CODE  : 0  (0x0)
SERVICE_EXIT_CODE : 0  (0x0)
CHECKPOINT     : 0x0
WAIT_HINT      : 0x7d0
PID            : 37636
FLAGS
```

The service **Status** will transition out of `START_PENDING` to **Running**.

View logs

To view logs, open the **Event Viewer**. Select the Windows key (or **Ctrl + Esc**), and search for `"Event Viewer"`. Select the **Event Viewer (Local) > Windows Logs > Application** node. You should see a **Warning** level entry with a **Source** matching the apps namespace. Double-click the entry, or right-click and select **Event Properties** to view the details.



After seeing logs in the **Event Log**, you should stop the service. It's designed to log a random joke once per minute. This is intentional behavior but is *not* practical for production services.

Stop the Windows Service

To stop the Windows Service, use the `sc.exe stop` command:

```
sc.exe stop ".NET Joke Service"
```

You'll see output similar to the following:

```
SERVICE_NAME: .NET Joke Service
TYPE          : 10  WIN32_OWN_PROCESS
STATE         : 3   STOP_PENDING
               (STOPPABLE, NOT_PAUSABLE, ACCEPTS_SHUTDOWN)
WIN32_EXIT_CODE : 0  (0x0)
SERVICE_EXIT_CODE : 0  (0x0)
CHECKPOINT    : 0x0
WAIT_HINT     : 0x0
```

The service **Status** will transition out of `STOP_PENDING` to **Stopped**.

Delete the Windows Service

To delete the Windows Service, use the native Windows Service Control Manager's (sc.exe) delete command. Run PowerShell as an Administrator.

IMPORTANT

If the service is not in the **Stopped** state, it will not be immediately deleted. Ensure that the service is stopped before issuing the delete command.

```
sc.exe delete ".NET Joke Service"
```

You'll see an output message:

```
[SC] DeleteService SUCCESS
```

For more information, see [sc.exe delete](#).

See also

- [Worker Services in .NET](#)
- [Create a Queue Service](#)
- [Use scoped services within a `BackgroundService`](#)
- [Implement the `IHostedService` interface](#)

Implement the `IHostedService` interface

9/20/2022 • 3 minutes to read • [Edit Online](#)

When you need finite control beyond the provided [BackgroundService](#), you can implement your own [IHostedService](#). The [IHostedService](#) interface is the basis for all long running services in .NET. Custom implementations are registered with the [AddHostedService<THostedService>\(IServiceCollection\)](#) extension method.

In this tutorial, you learn how to:

- Implement the [IHostedService](#), and [IAutoDisposable](#) interfaces.
- Create a timer-based service.
- Register the custom implementation with dependency injection and logging.

TIP

All of the "Workers in .NET" example source code is available in the [Samples Browser](#) for download. For more information, see [Browse code samples: Workers in .NET](#).

Prerequisites

- The [.NET 5.0 SDK or later](#)
- A .NET integrated development environment (IDE)
 - Feel free to use [Visual Studio](#)

Create a new project

To create a new Worker Service project with Visual Studio, you'd select **File > New > Project....** From the **Create a new project** dialog search for "Worker Service", and select Worker Service template. If you'd rather use the .NET CLI, open your favorite terminal in a working directory. Run the `dotnet new` command, and replace the `<Project.Name>` with your desired project name.

```
dotnet new worker --name <Project.Name>
```

For more information on the .NET CLI new worker service project command, see [dotnet new worker](#).

TIP

If you're using Visual Studio Code, you can run .NET CLI commands from the integrated terminal. For more information, see [Visual Studio Code: Integrated Terminal](#).

Create timer service

The timer-based background service makes use of the [System.Threading.Timer](#) class. The timer triggers the `DoWork` method. The timer is disabled on [IHostLifetime.StopAsync\(CancellationToken\)](#) and disposed when the service container is disposed on [IAutoDisposable.DisposeAsync\(\)](#):

Replace the contents of the `Worker` from the template with the following C# code, and rename the file to

TimerService.cs:

```
namespace App.TimerHostedService;

public sealed class TimerService : IHostedService, IAsyncDisposable
{
    private readonly Task _completedTask = Task.CompletedTask;
    private readonly ILogger<TimerService> _logger;
    private int _executionCount = 0;
    private Timer? _timer;

    public TimerService(ILogger<TimerService> logger) => _logger = logger;

    public Task StartAsync(CancellationToken stoppingToken)
    {
        _logger.LogInformation("{Service} is running.", nameof(TimerHostedService));
        _timer = new Timer(DoWork, null, TimeSpan.Zero, TimeSpan.FromSeconds(5));

        return _completedTask;
    }

    private void DoWork(object? state)
    {
        int count = Interlocked.Increment(ref _executionCount);

        _logger.LogInformation(
            "{Service} is working, execution count: {Count:#,0}",
            nameof(TimerHostedService),
            count);
    }

    public Task StopAsync(CancellationToken stoppingToken)
    {
        _logger.LogInformation(
            "{Service} is stopping.", nameof(TimerHostedService));

        _timer?.Change(Timeout.Infinite, 0);

        return _completedTask;
    }

    public async ValueTask DisposeAsync()
    {
        if (_timer is IAsyncDisposable timer)
        {
            await timer.DisposeAsync();
        }

        _timer = null;
    }
}
```

IMPORTANT

The `Worker` was a subclass of `BackgroundService`. Now, the `TimerService` implements both the `IHostedService`, and `IAsyncDisposable` interfaces.

The `TimerService` is `sealed`, and cascades the `DisposeAsync` call from its `_timer` instance. For more information on the "cascading dispose pattern", see [Implement a `DisposeAsync` method](#).

When `StartAsync` is called, the timer is instantiated, thus starting the timer.

TIP

The `Timer` doesn't wait for previous executions of `Dowork` to finish, so the approach shown might not be suitable for every scenario. `Interlocked.Increment` is used to increment the execution counter as an atomic operation, which ensures that multiple threads don't update `_executionCount` concurrently.

Replace the existing `Program` contents with the following C# code:

```
using App.TimerHostedService;

using IHost host = Host.CreateDefaultBuilder(args)
    .ConfigureServices(services =>
{
    services.AddHostedService<TimerService>();
})
.Build();

await host.RunAsync();
```

The service is registered in `IHostBuilder.ConfigureServices` (`Program.cs`) with the `AddHostedService` extension method. This is the same extension method you use when registering `BackgroundService` subclasses, as they both implement the `IHostedService` interface.

For more information on registering services, see [Dependency injection in .NET](#).

Verify service functionality

To run the application from Visual Studio, select F5 or select the **Debug > Start Debugging** menu option. If you're using the .NET CLI, run the `dotnet run` command from the working directory:

```
dotnet run
```

For more information on the .NET CLI run command, see [dotnet run](#).

Let the application run for a bit to generate several execution count increments. You will see output similar to the following:

```
info: App.TimerHostedService.TimerService[0]
      TimerHostedService is running.
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
      Content root path: .\timer-service
info: App.TimerHostedService.TimerService[0]
      TimerHostedService is working, execution count: 1
info: App.TimerHostedService.TimerService[0]
      TimerHostedService is working, execution count: 2
info: App.TimerHostedService.TimerService[0]
      TimerHostedService is working, execution count: 3
info: App.TimerHostedService.TimerService[0]
      TimerHostedService is working, execution count: 4
info: Microsoft.Hosting.Lifetime[0]
      Application is shutting down...
info: App.TimerHostedService.TimerService[0]
      TimerHostedService is stopping.
```

If running the application from within Visual Studio, select **Debug > Stop Debugging**.... Alternatively, select **Ctrl + C** from the console window to signal cancellation.

See also

There are several related tutorials to consider:

- [Worker Services in .NET](#)
- [Create a Queue Service](#)
- [Use scoped services within a `BackgroundService`](#)
- [Create a Windows Service using `BackgroundService`](#)

Deploy a Worker Service to Azure

9/20/2022 • 11 minutes to read • [Edit Online](#)

In this article, you'll learn how to deploy a .NET Worker Service to Azure. With your Worker running as an [Azure Container Instance \(ACI\)](#) from the [Azure Container Registry \(ACR\)](#), it can act as a microservice in the cloud. There are many use cases for long-running services, and the Worker Service exists for this reason.

In this tutorial, you learn how to:

- Create a worker service.
- Create container registry resource.
- Push an image to container registry.
- Deploy as container instance.
- Verify worker service functionality.

TIP

All of the "Workers in .NET" example source code is available in the [Samples Browser](#) for download. For more information, see [Browse code samples: Workers in .NET](#).

Prerequisites

- The [.NET 5.0 SDK or later](#).
- Docker Desktop ([Windows](#) or [Mac](#)).
- An Azure account with an active subscription. [Create an account for free](#).
- Depending on your developer environment of choice:
 - [Visual Studio](#), [Visual Studio Code](#), or [Visual Studio for Mac](#).
 - [.NET CLI](#)
 - [Azure CLI](#).

Create a new project

To create a new Worker Service project with Visual Studio, select **File > New > Project....** From the **Create a new project** dialog search for "Worker Service", and select Worker Service template. Enter the desired project name, select an appropriate location, and select **Next**. On the **Additional information** page, for the **Target Framework** select [.NET 5.0](#), and check the **Enable Docker** option to enable docker support. Select the desired Docker OS.

To create a new Worker Service project with Visual Studio Code, you can run .NET CLI commands from the integrated terminal. For more information, see [Visual Studio Code: Integrated Terminal](#).

Open the integrated terminal, and run the `dotnet new` command, and replace the `<Project.Name>` with your desired project name.

```
dotnet new worker --name <Project.Name>
```

For more information on the .NET CLI new worker service project command, see [dotnet new worker](#).

To create a new Worker Service project with the .NET CLI, open your favorite terminal in a working directory.

Run the `dotnet new` command, and replace the `<Project.Name>` with your desired project name.

```
dotnet new worker --name <Project.Name>
```

For more information on the .NET CLI new worker service project command, see [dotnet new worker](#).

Build the application to ensure it restores the dependent packages, and compiles without error.

To build the application from Visual Studio, select F6 or select the **Build > Build Solution** menu option.

To build the application from Visual Studio Code, open the integrated terminal window and run the

```
dotnet build
```

```
dotnet build
```

For more information on the .NET CLI build command, see [dotnet build](#).

To build the application from the .NET CLI, run the `dotnet build` command from the working directory.

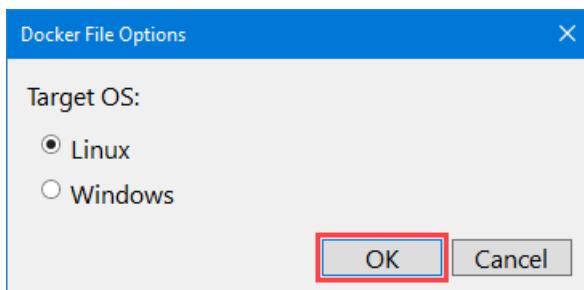
```
dotnet build <path/to/project.csproj>
```

Specify your `<path/to/project.csproj>` value, which is the path to the project file to build. For more information on the .NET CLI build command, see [dotnet build](#).

Add Docker support

If you correctly selected the **Enable Docker** checkbox when creating a new Worker project, skip to the [Build the Docker image](#) step.

If you didn't select this option, no worries—you can still add it now. In Visual Studio, right-click on the *project node* in the **Solution Explorer**, and select **Add > Docker Support**. You'll be prompted to select a **Target OS**, select **OK** with the default OS selection.



In Visual Studio Code, you'll need the [Docker extension](#) and the [Azure Account extension](#) installed. Open the Command Palette, and select the **Docker: Add Docker files to workspace** option. If prompted to **Select Application Platform**, choose **.NET: Core Console**. If prompted to **Select Project**, choose the Worker Service project you created. When prompted to **Select Operating System**, choose the first listed OS. When prompted whether or not to **Include optional Docker Compose files**, select **No**.

Docker support requires a *Dockerfile*. This file is a set of comprehensive instructions, for building your .NET Worker Service as a Docker image. The *Dockerfile* is a file *without* a file extension. The following is an example *Dockerfile*, and should exist at the root directory of the project file.

With the CLI, the *Dockerfile* is **not** created for you. You'll need to copy its contents into a new file name *Dockerfile*, and again this file should be in the root directory of the project.

```
FROM mcr.microsoft.com/dotnet/runtime:6.0 AS base
WORKDIR /app

# Creates a non-root user with an explicit UID and adds permission to access the /app folder
# For more info, please refer to https://aka.ms/vscode-docker-dotnet-configure-containers
RUN adduser -u 5678 --disabled-password --gecos "" appuser && chown -R appuser /app
USER appuser

FROM mcr.microsoft.com/dotnet/sdk:6.0 AS build
WORKDIR /src
COPY ["App.CloudService.csproj", "./"]
RUN dotnet restore "App.CloudService.csproj"
COPY .
WORKDIR "/src/."
RUN dotnet build "App.CloudService.csproj" -c Release -o /app/build

FROM build AS publish
RUN dotnet publish "App.CloudService.csproj" -c Release -o /app/publish

FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "App.CloudService.dll"]
```

NOTE

You need to update the various lines in the *Dockerfile* that reference `*App.CloudService`—replace this with the name of your project.

For more information on the official .NET images, see [Docker Hub: .NET Runtime](#) and [Docker Hub: .NET SDK](#).

Build the Docker image

To build the Docker image, the Docker Engine must be running.

IMPORTANT

When using Docker Desktop and Visual Studio, to avoid errors related to *volume sharing*—ensure that it is enabled.

1. On the **Settings** screen in Docker Desktop, select **Shared Drives**.
2. Select the drive(s) containing your project files.

For more information, see [Troubleshoot Visual Studio development with Docker](#).

Right-click on the *Dockerfile* in the **Solution Explorer**, and select **Build Docker Image**. The **Output** window displays, reporting the `docker build` command progress.

Right-click on the *Dockerfile* in the **Explorer**, and select **Build Image**. When prompted to **Tag image as**, enter `appcloudservice:latest`. The **Docker Task** output terminal displays, reporting the Docker build command progress.

NOTE

If you're *not* prompted to tag the image, it's possible that Visual Studio Code is relying on an existing `tasks.json`. If the tag used is undesirable, you can change it by updating the `docker-build` configuration item's `dockerBuild/tag` value in the `tasks` array. Consider the following example configuration section:

```
{  
  "type": "docker-build",  
  "label": "docker-build: release",  
  "dependsOn": [  
    "build"  
  ],  
  "dockerBuild": {  
    "tag": "appcloudservice:latest",  
    "dockerfile": "${workspaceFolder}/cloud-service/Dockerfile",  
    "context": "${workspaceFolder}",  
    "pull": true  
  },  
  "netCore": {  
    "appProject": "${workspaceFolder}/cloud-service/App.CloudService.csproj"  
  }  
}
```

Open a terminal window in the root directory of the `Dockerfile`, and run the following docker command:

```
docker build -t appcloudservice:latest -f Dockerfile .
```

As the `docker build` command runs, it processes each line in the `Dockerfile` as an instruction step. This command builds the image and creates a local repository named `appcloudservice` that points to the image.

TIP

The generated `Dockerfile` differs between development environments. For example, if you [Add Docker support](#) from Visual Studio you may experience issues if you attempt to [Build the Docker image](#) from Visual Studio Code — as the `Dockerfile` steps vary. It is best to choose a single *development environment* and use it through out this tutorial.

Create container registry

An Azure Container Registry (ACR) resource allows you to build, store, and manage container images and artifacts in a private registry. To create a container registry, you'll need to [create a new resource](#) in the Azure portal.

1. Select the **Subscription**, and corresponding **Resource group** (or create a new one).
2. Enter a **Registry name**.
3. Select a **Location**.
4. Select an appropriate **SKU**, for example **Basic**.
5. Select **Review + create**.
6. After seeing **Validation passed**, select **Create**.

IMPORTANT

In order to use this container registry when creating a container instance, you must enable **Admin user**. Select **Access keys**, and enable **Admin user**.

An Azure Container Registry (ACR) resource allows you to build, store, and manage container images and

artifacts in a private registry. Open a terminal window in the root directory of the *Dockerfile*, and run the following Azure CLI command:

IMPORTANT

To interact with Azure resources from the Azure CLI, you must be authenticated for your terminal session. To authenticate, use the `az login` command:

```
az login
```

After you're logged in, use the `az account set` command to specify your subscription when you have more than one and no default subscription set.

```
az account set --subscription <subscription name or id>
```

Once you sign in to the Azure CLI, your session can interact with resources accordingly.

If you do not already have a resource group you'd like to associate your worker service with, create one using the `az group create` command:

```
az group create -n <resource group> -l <location>
```

Provide the `<resource group>` name, and the `<location>`. To create a container registry, you'll need to call the `az acr create` command.

```
az acr create -n <registry name> -g <resource group> --sku <sku> --admin-enabled true
```

Replace placeholders with your own appropriate values:

- `<registry name>` : the name of the registry.
- `<resource group>` : the resource group name that you used above.
- `<sku>` : accepted values, **Basic**, **Classic**, **Premium**, or **Standard**.

The preceding command:

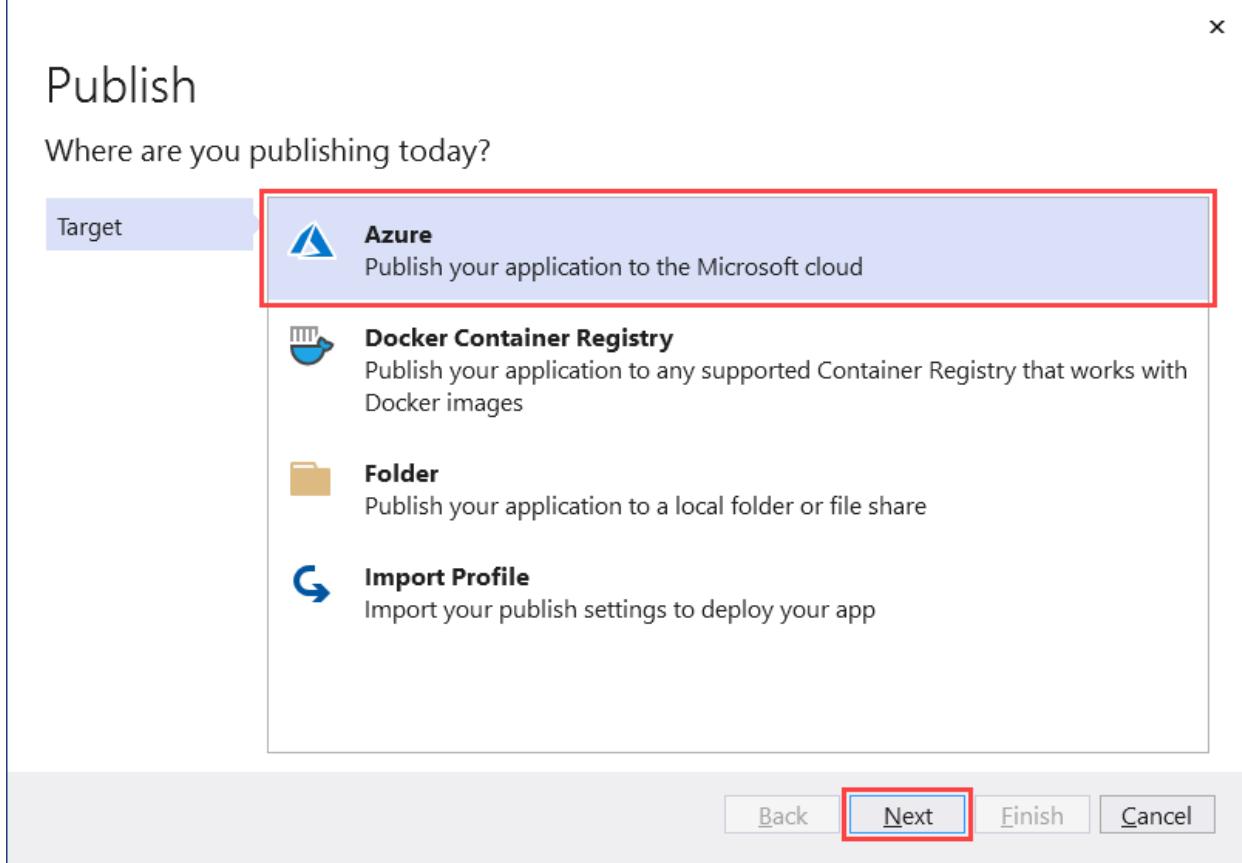
- Creates an Azure Container Registry, given a registry name, in the specified resource group.
- Enabled an Admin user—this is required for Azure Container Instances.

For more information, see [Quickstart: Create an Azure container registry](#).

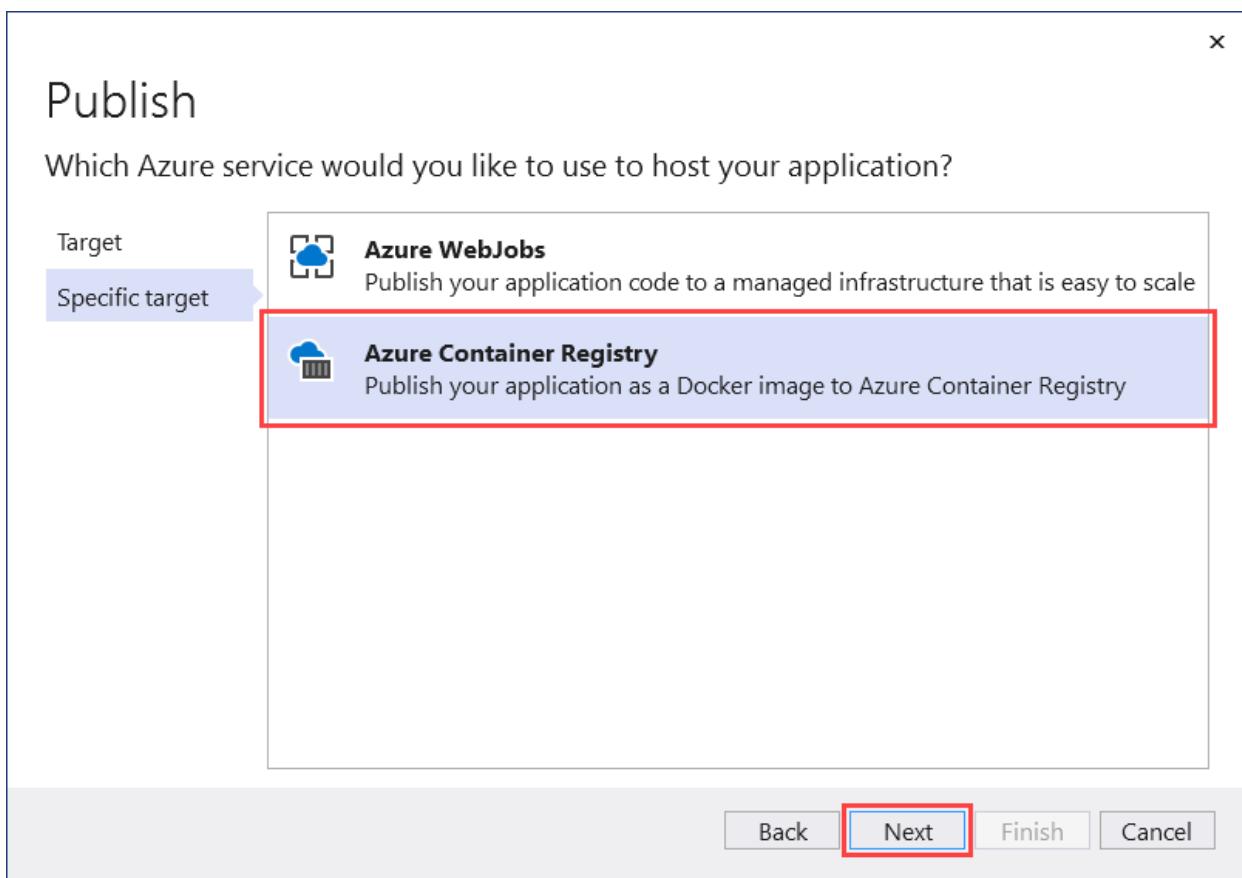
Push image to container registry

With the .NET Docker image built, and the container registry resource created, you can now push the image to container registry.

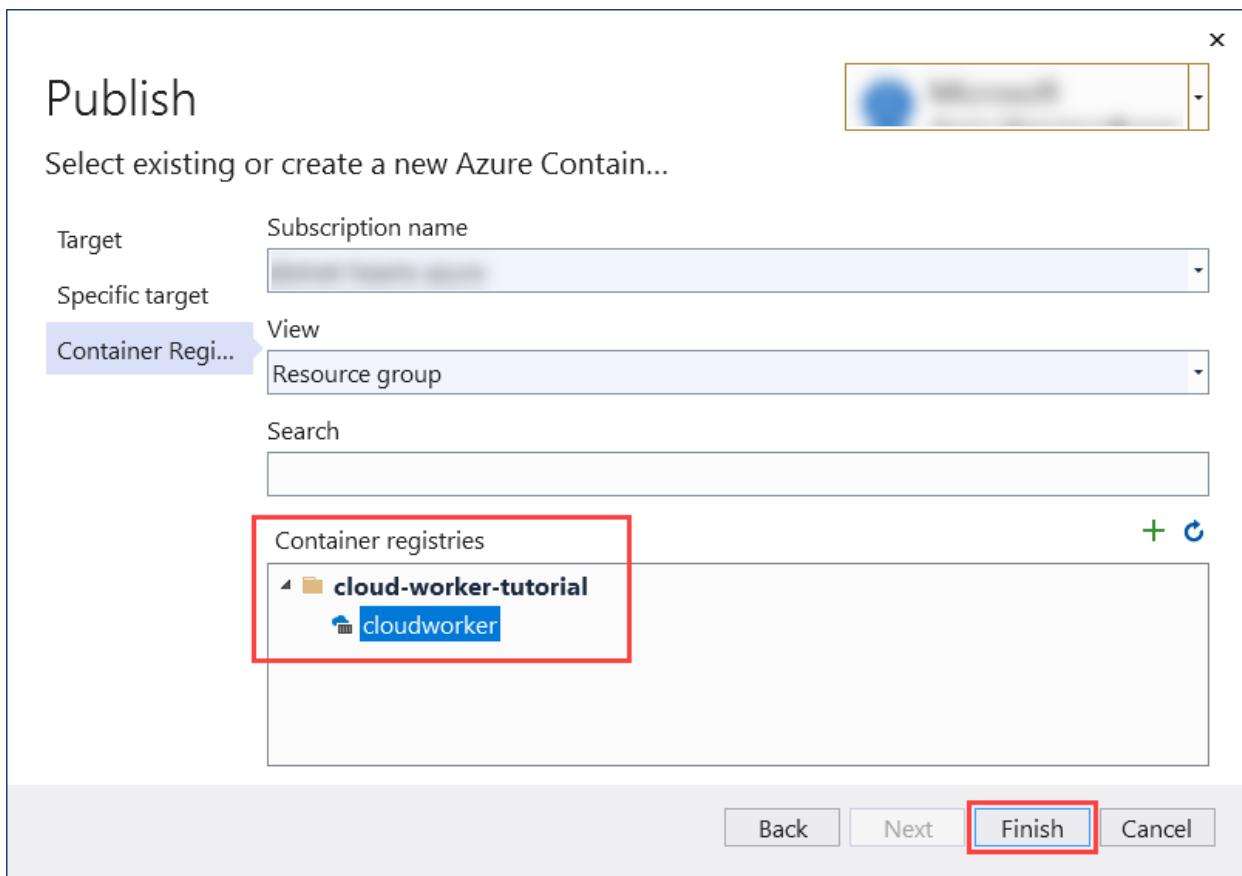
Right-click on the project in the **Solution Explorer**, and select **Publish**. The **Publish** dialog displays. For the **Target**, select **Azure** and then **Next**.



For the Specific Target, select Azure Container Registry and then Next.

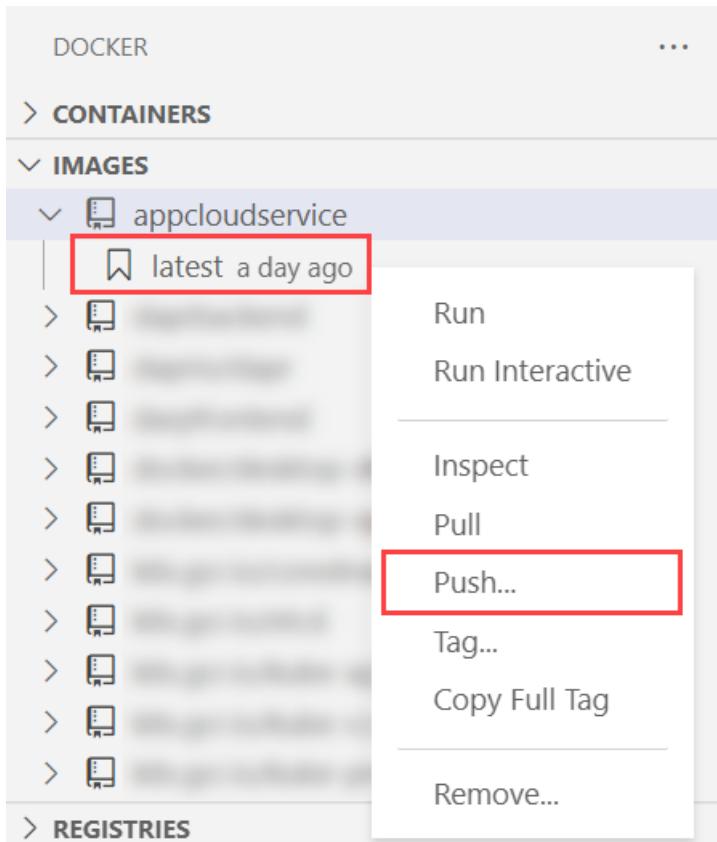


Next, for the Container Registry, select the Subscription name that you used to create the ACR resource. From the Container registries selection area, select the container registry that you created, and then select Finish.



This creates a publish profile, which can be used to publish the image to container registry. Select the **Publish** button to push the image to the container registry, the **Output** window reports the publish progress—and when it completes successfully, you'll see a "Successfully published" message.

Select **Docker** from the **Activity Bar** in Visual Studio Code. Expand the **IMAGES** tree view panel, then expand the `appcloudservice` image node and right-click on the `latest` tag.



The integrated terminal window will report the progress of the `docker push` command to the container registry.

To push an image to the container registry, you need to sign in to the registry first:

```
az acr login -n <registry name>
```

The `az acr login` command will sign in to a container registry through the Docker CLI. To push the image to the container registry, use the `az acr build` command with your container registry name as the `<registry name>`:

```
az acr build -r <registry name> -t appcloudservice .
```

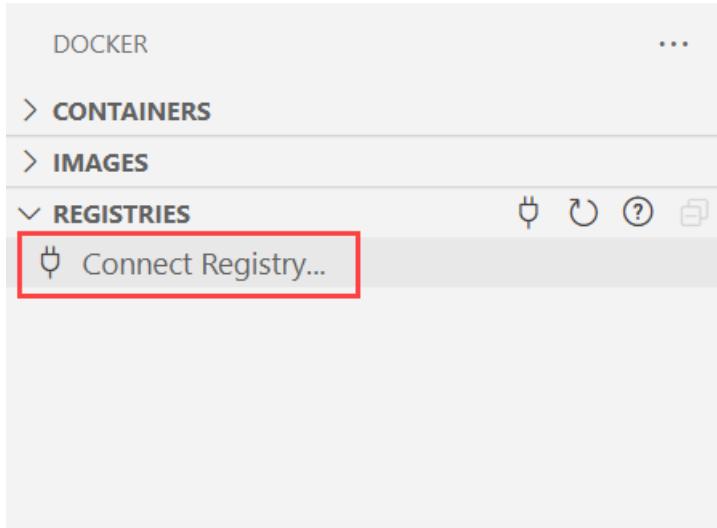
The preceding command:

- Packs the source into a *tar* file.
- Uploads it to the container registry.
- The container registry unpacks the *tar* file.
- Runs the `docker build` command in the container registry resource against the *Dockerfile*.
- Adds the image to the container registry.

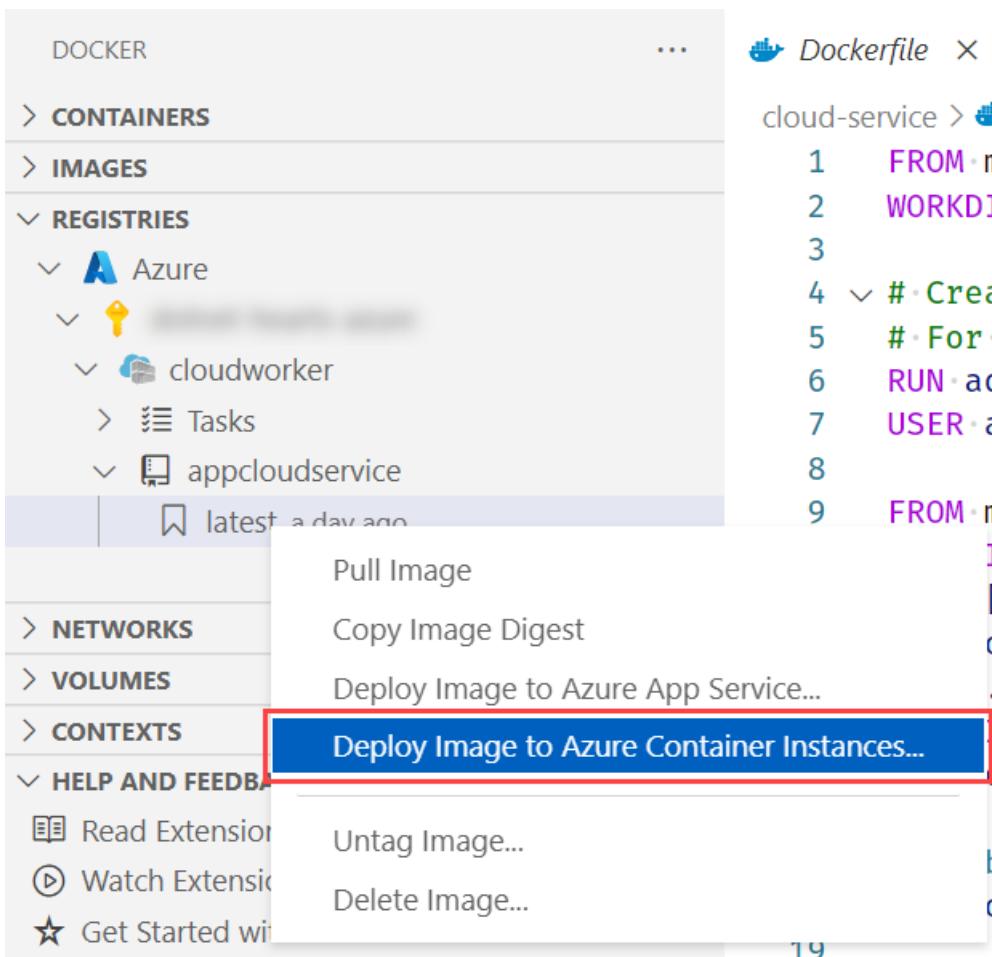
To verify that the image was successfully pushed to the container registry, navigate to the Azure portal. Open the container registry resource, under **Services**, select **Repositories**. You should see the image.

Deploy as container instance

From Visual Studio Code, select **Docker** from the **Activity Bar**. Expand the **REGISTRIES** node, and select **Connect Registry**. Select **Azure** when prompted, and sign in if necessary.



Expand the **REGISTRIES** node, select **Azure**, your subscription > the container registry > the image, and then right-click the tag. Select **Deploy Image to Azure Container Instances**.



To create a container instance, you'll need to create a container group using the `az container create` command.

```
az container create -g <resource group> \
--name <instance name> \
--image <registry name>.azurecr.io/<image name>:latest \
--registry-password <password>
```

Provide the appropriate values:

- `<resource group>`: the resource group name that you've been using in this tutorial.
- `<instance name>`: the name of the container instance.
- `<registry name>`: the name of the container registry.
- `<image name>`: the name of the image.
- `<password>`: the password to the container registry—you can get this from the Azure portal, Container Registry resource > Access Keys.

To create a container instance, you'll need to [create a new resource](#) in the Azure portal.

1. Select the same **Subscription**, and corresponding **Resource group** from the previous section.
2. Enter a **Container name**—`appcloudservice-container`.
3. Select a **Region** that corresponds to the previous **Location** selection.
4. For **Image source**, select **Azure Container Registry**.
5. Select the **Registry** by the name provided in the previous step.
6. Select the **Image** and **Image tag**.
7. Select **Review + create**.
8. Assuming **Validation passed**, select **Create**.

It may take a moment for the resources to be created, once created select the **Go to resource** button.

For more information, see [Quickstart: Create an Azure container instance](#).

Verify service functionality

Immediately after the container instance is created, it starts running.

To verify your worker service is functioning correctly, navigate to the Azure portal in the container instance resource, select the **Containers** option.

The screenshot shows the Azure portal interface for a container instance named 'worker-service'. On the left, there's a sidebar with options like Overview, Activity log, Access control (IAM), Tags, Settings (with 'Containers' selected), Identity, Properties, Locks, Monitoring (Metrics, Alerts), Automation, and Tasks (preview). The main area shows '1 container' with a table:

Name	Image	State
worker-service	cloudworker.azurecr.io/appclou...	Running

The 'Logs' tab is selected, showing the following log output:

```
[40m[32minfo[39m[22m[49m: App.CloudService.Worker[0]
Worker running at: 06/17/2021 19:19:32 +00:00
[40m[32minfo[39m[22m[49m: Microsoft.Hosting.Lifetime[0]
Application started. Press Ctrl+C to shut down.
[40m[32minfo[39m[22m[49m: Microsoft.Hosting.Lifetime[0]
Hosting environment: Production
[40m[32minfo[39m[22m[49m: Microsoft.Hosting.Lifetime[0]
Content root path: /app
[40m[32minfo[39m[22m[49m: App.CloudService.Worker[0]
Worker running at: 06/17/2021 19:19:37 +00:00
[40m[32minfo[39m[22m[49m: App.CloudService.Worker[0]
Worker running at: 06/17/2021 19:19:42 +00:00
```

You'll see the containers, and their current **State**. In this case, it will be **Running**. Select **Logs** to see the .NET worker service output.

To verify your worker service is functioning correctly, you can view the logs from your running application. Use the `az container logs` command:

```
az container logs -g <resource group> --name <instance name>
```

Provide the appropriate values:

- `<resource group>`: the resource group name that you've been using in this tutorial.
- `<instance name>`: the name of the container instance.

You'll see the .NET worker service output logs, which means you've successfully deployed your containerized app to ACI.

See also

- [Worker Services in .NET](#)
- [Use scoped services within a `BackgroundService`](#)
- [Create a Windows Service using `BackgroundService`](#)
- [Implement the `IHostedService` interface](#)
- [Tutorial: Containerize a .NET Core app](#)

Caching in .NET

9/20/2022 • 14 minutes to read • [Edit Online](#)

In this article, you'll learn about various caching mechanisms. Caching is the act of storing data in an intermediate-layer, making subsequent data retrievals faster. Conceptually, caching is a performance optimization strategy and design consideration. Caching can significantly improve app performance by making infrequently changing (or expensive to retrieve) data more readily available. This article introduces the two primary types of caching, and provides sample source code for both:

- [Microsoft.Extensions.Caching.Memory](#)
- [Microsoft.Extensions.Caching.Distributed](#)

IMPORTANT

There are two `MemoryCache` classes within .NET, one in the `System.Runtime.Caching` namespace and the other in the `Microsoft.Extensions.Caching` namespace:

- `System.Runtime.Caching.MemoryCache`
- `Microsoft.Extensions.Caching.Memory.MemoryCache`

While this article focuses on caching, it doesn't include the `System.Runtime.Caching` NuGet package. All references to `MemoryCache` are within the `Microsoft.Extensions.Caching` namespace.

All of the `Microsoft.Extensions.*` packages come dependency injection (DI) ready, both the `IMemoryCache` and `IDistributedCache` interfaces can be used as services.

In-memory caching

In this section, you'll learn about the `Microsoft.Extensions.Caching.Memory` package. The current implementation of the `IMemoryCache` is a wrapper around the `ConcurrentDictionary< TKey, TValue >`, exposing a feature-rich API. Entries within the cache are represented by the `ICacheEntry`, and can be any `object`. The in-memory cache solution is great for apps that run in a single server, where all the cached data rents memory in the app's process.

TIP

For multi-server caching scenarios, consider the [Distributed caching](#) approach as an alternative to in-memory caching.

In-memory caching API

The consumer of the cache has control over both sliding and absolute expirations:

- `ICacheEntry.AbsoluteExpiration`
- `ICacheEntry.AbsoluteExpirationRelativeToNow`
- `ICacheEntry SlidingExpiration`

Setting an expiration will cause entries in the cache to be *evicted* if they're not accessed within the expiration time allotment. Consumers have additional options for controlling cache entries, through the `MemoryCacheEntryOptions`. Each `ICacheEntry` is paired with `MemoryCacheEntryOptions` which exposes expiration eviction functionality with `IChangeToken`, priority settings with `CacheltemPriority`, and controlling the `ICacheEntry.Size`. Consider the following extension methods:

- [MemoryCacheEntryExtensions.AddExpirationToken](#)
- [MemoryCacheEntryExtensions.RegisterPostEvictionCallback](#)
- [MemoryCacheEntryExtensions.SetSize](#)
- [MemoryCacheEntryExtensions.SetPriority](#)

In-memory cache example

To use the default [IMemoryCache](#) implementation, call the [AddMemoryCache](#) extension method to register all the required services with DI. In the following code sample, the generic host is used to expose the [ConfigureServices](#) functionality:

```
using Microsoft.Extensions.Caching.Memory;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;

IHost host = Host.CreateDefaultBuilder(args)
    .ConfigureServices(services => services.AddMemoryCache())
    .Build();
```

Depending on your .NET workload, you may access the [IMemoryCache](#) differently; such as constructor injection. In this sample, you use the [IServiceProvider](#) instance on the [host](#) and call generic [GetRequiredService<T>](#) ([IServiceProvider](#)) extension method:

```
IMemoryCache cache =
    host.Services.GetRequiredService<IMemoryCache>();
```

With in-memory caching services registered, and resolved through DI — you're ready to start caching. This sample iterates through the letters in the English alphabet 'A' through 'Z'. The [record AlphabetLetter](#) type holds the reference to the letter, and generates a message.

```
record AlphabetLetter(char Letter)
{
    internal string Message =>
        $"The '{Letter}' character is the {Letter - 64} letter in the English alphabet.";
```

The sample includes a helper function that iterates through the alphabet letters:

```
static async ValueTask IterateAlphabetAsync(
    Func<char, Task> asyncFunc)
{
    for (char letter = 'A'; letter <= 'Z'; ++ letter)
    {
        await asyncFunc(letter);
    }

    Console.WriteLine();
}
```

In the preceding C# code:

- The [Func<char, Task> asyncFunc](#) is awaited on each iteration, passing the current [letter](#).
- After all letters have been processed, a blank line is written to the console.

To add items to the cache call one of the [Create](#), or [Set](#) APIs:

```

var addLettersToCacheTask = IterateAlphabetAsync(letter =>
{
    MemoryCacheEntryOptions options = new()
    {
        AbsoluteExpirationRelativeToNow =
            TimeSpan.FromMilliseconds(MillisecondsAbsoluteExpiration)
    };

    _ = options.RegisterPostEvictionCallback(OnPostEviction);

    AlphabetLetter alphabetLetter =
        cache.Set(
            letter, new AlphabetLetter(letter), options);

    Console.WriteLine($"{alphabetLetter.Letter} was cached.");
}

return Task.Delay(
    TimeSpan.FromMilliseconds(MillisecondsDelayAfterAdd));
});
await addLettersToCacheTask;

```

In the preceding C# code:

- The variable `addLettersToCacheTask` delegates to `IterateAlphabetAsync` and is awaited.
- The `Func<char, Task> asyncFunc` is argued with a lambda.
- The `MemoryCacheEntryOptions` is instantiated with an absolute expiration relative to now.
- A post eviction callback is registered.
- An `AlphabetLetter` object is instantiated, and passed into `Set` along with `letter` and `options`.
- The letter is written to the console as being cached.
- Finally, a `Task.Delay` is returned.

For each letter in the alphabet, a cache entry is written with an expiration, and post eviction callback.

The post eviction callback writes the details of the value that was evicted to the console:

```

static void OnPostEviction(
    object key, object letter, EvictionReason reason, object state)
{
    if (letter is AlphabetLetter alphabetLetter)
    {
        Console.WriteLine($"{alphabetLetter.Letter} was evicted for {reason}.");
    }
}

```

Now that the cache is populated, another call to `IterateAlphabetAsync` is awaited, but this time you'll call `IMemoryCache.TryGetValue`:

```

var readLettersFromCacheTask = IterateAlphabetAsync(letter =>
{
    if (cache.TryGetValue(letter, out object? value) &&
        value is AlphabetLetter alphabetLetter)
    {
        Console.WriteLine($"{letter} is still in cache. {alphabetLetter.Message}");
    }

    return Task.CompletedTask;
});
await readLettersFromCacheTask;

```

If the `cache` contains the `letter` key, and the `value` is an instance of an `AlphabetLetter` it's written to the console. When the `letter` key is not in the cache, it was evicted and its post eviction callback was invoked.

Additional extension methods

The `IMemoryCache` comes with many convenience-based extension methods, including an asynchronous `GetOrCreateAsync`:

- [CacheExtensions.Get](#)
- [CacheExtensions.GetOrCreate](#)
- [CacheExtensions.GetOrCreateAsync](#)
- [CacheExtensions.Set](#)
- [CacheExtensions.TryGetValue](#)

Put it all together

The entire sample app source code is a top-level program and requires two NuGet packages:

- [Microsoft.Extensions.Caching.Memory](#)
- [Microsoft.Extensions.Hosting](#)

```
using Microsoft.Extensions.Caching.Memory;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;

IHost host = Host.CreateDefaultBuilder(args)
    .ConfigureServices(services => services.AddMemoryCache())
    .Build();

IMemoryCache cache =
    host.Services.GetRequiredService<IMemoryCache>();

const int MillisecondsDelayAfterAdd = 50;
const int MillisecondsAbsoluteExpiration = 750;

static void OnPostEviction(
    object key, object letter, EvictionReason reason, object state)
{
    if (letter is AlphabetLetter alphabetLetter)
    {
        Console.WriteLine($"{alphabetLetter.Letter} was evicted for {reason}.");
    }
};

static async ValueTask IterateAlphabetAsync(
    Func<char, Task> asyncFunc)
{
    for (char letter = 'A'; letter <= 'Z'; ++ letter)
    {
        await asyncFunc(letter);
    }

    Console.WriteLine();
}

var addLettersToCacheTask = IterateAlphabetAsync(letter =>
{
    MemoryCacheEntryOptions options = new()
    {
        AbsoluteExpirationRelativeToNow =
            TimeSpan.FromMilliseconds(MillisecondsAbsoluteExpiration)
    };

    _ = options.RegisterPostEvictionCallback(OnPostEviction);

    AlphabetLetter alphabetLetter =
```

```

        cache.Set(
            letter, new AlphabetLetter(letter), options);

        Console.WriteLine($"{alphabetLetter.Letter} was cached.");

        return Task.Delay(
            TimeSpan.FromMilliseconds(MillisecondsDelayAfterAdd));
    });
}

await addLettersToCacheTask;

var readLettersFromCacheTask = IterateAlphabetAsync(letter =>
{
    if (cache.TryGetValue(letter, out object? value) &&
        value is AlphabetLetter alphabetLetter)
    {
        Console.WriteLine($"{letter} is still in cache. {alphabetLetter.Message}");
    }

    return Task.CompletedTask;
});
await readLettersFromCacheTask;

await host.RunAsync();

record AlphabetLetter(char Letter)
{
    internal string Message =>
        $"The '{Letter}' character is the {Letter - 64} letter in the English alphabet.";
}

```

Feel free to adjust the `MillisecondsDelayAfterAdd` and `MillisecondsAbsoluteExpiration` values to observe the changes in behavior to the expiration and eviction of cached entries. The following is sample output from running this code, due to the non-deterministic nature of .NET events — there is no guarantee that your output will be identical.

```
A was cached.  
B was cached.  
C was cached.  
D was cached.  
E was cached.  
F was cached.  
G was cached.  
H was cached.  
I was cached.  
J was cached.  
K was cached.  
L was cached.  
M was cached.  
N was cached.  
O was cached.  
P was cached.  
Q was cached.  
R was cached.  
S was cached.  
T was cached.  
U was cached.  
V was cached.  
W was cached.  
X was cached.  
Y was cached.  
Z was cached.  
  
A was evicted for Expired.  
C was evicted for Expired.  
B was evicted for Expired.  
E was evicted for Expired.  
D was evicted for Expired.  
F was evicted for Expired.  
H was evicted for Expired.  
K was evicted for Expired.  
L was evicted for Expired.  
J was evicted for Expired.  
G was evicted for Expired.  
M was evicted for Expired.  
N was evicted for Expired.  
I was evicted for Expired.  
P was evicted for Expired.  
R was evicted for Expired.  
O was evicted for Expired.  
Q was evicted for Expired.  
S is still in cache. The 'S' character is the 19 letter in the English alphabet.  
T is still in cache. The 'T' character is the 20 letter in the English alphabet.  
U is still in cache. The 'U' character is the 21 letter in the English alphabet.  
V is still in cache. The 'V' character is the 22 letter in the English alphabet.  
W is still in cache. The 'W' character is the 23 letter in the English alphabet.  
X is still in cache. The 'X' character is the 24 letter in the English alphabet.  
Y is still in cache. The 'Y' character is the 25 letter in the English alphabet.  
Z is still in cache. The 'Z' character is the 26 letter in the English alphabet.
```

Since the absolute expiration ([MemoryCacheEntryOptions.AbsoluteExpirationRelativeToNow](#)) is set, all the cached items will eventually be evicted.

Worker Service caching

One common strategy for caching data, is updating the cache independently from the consuming data services. The *Worker Service* template is a great example, as the [BackgroundService](#) runs independent (or in the background) from the other application code. When an application starts running that hosts an implementation of the [IHostedService](#), the corresponding implementation (in this case the [BackgroundService](#) or "worker") starts running in the same process. These hosted services are registered with DI as singletons, through the

`AddHostedService<THostedService>(IServiceCollection)` extension method. Other services can be registered with DI with any [service lifetime](#).

IMPORTANT

The service lifetime's are very important to understand. When you call `AddMemoryCache` to register all of the in-memory caching services, the services are registered as singletons.

Photo service scenario

Imagine you're developing a photo service that relies on third-party API accessible via HTTP. This photo data doesn't change very often, but there is a lot of it. Each photo is represented by a simple `record`:

```
namespace CachingExamples.Memory;

public record Photo(
    int AlbumId,
    int Id,
    string Title,
    string Url,
    string ThumbnailUrl);
```

In the following example, you'll see several services being registered with DI. Each service has a single responsibility.

```
using CachingExamples.Memory;

using IHost host = Host.CreateDefaultBuilder(args)
    .ConfigureServices(services =>
{
    services.AddMemoryCache();
    services.AddHttpClient<CacheWorker>();
    services.AddHostedService<CacheWorker>();
    services.AddScoped<PhotoService>();
    services.AddSingleton(typeof(CacheSignal<>));
})
.Build();

await host.StartAsync();
```

In the preceding C# code:

- The generic host is created with [defaults](#).
- In-memory caching services are registered with `AddMemoryCache`.
- An `HttpClient` instance is registered for the `CacheWorker` class with `AddHttpClient<TClient>(IServiceCollection)`.
- The `CacheWorker` class is registered with `AddHostedService<THostedService>(IServiceCollection)`.
- The `PhotoService` class is registered with `AddScoped<TService>(IServiceCollection)`.
- The `CacheSignal<T>` class is registered with `AddSingleton`.
- The `host` is instantiated from the builder, and started asynchronously.

The `PhotoService` is responsible for getting photos that match a given criteria (or `filter`):

```

using Microsoft.Extensions.Caching.Memory;

namespace CachingExamples.Memory;

public sealed class PhotoService
{
    private readonly IMemoryCache _cache;
    private readonly CacheSignal<Photo> _cacheSignal;
    private readonly ILogger<PhotoService> _logger;

    public PhotoService(
        IMemoryCache cache,
        CacheSignal<Photo> cacheSignal,
        ILogger<PhotoService> logger) =>
        (_cache, _cacheSignal, _logger) = (cache, cacheSignal, logger);

    public async IAsyncEnumerable<Photo> GetPhotosAsync(Func<Photo, bool>? filter = default)
    {
        try
        {
            await _cacheSignal.WaitAsync();

            Photo[] photos =
                await _cache.GetOrCreateAsync(
                    "Photos", _ =>
                {
                    _logger.LogWarning("This should never happen!");

                    return Task.FromResult(Array.Empty<Photo>());
                });

            // If no filter is provided, use a pass-thru.
            filter ??= _ => true;

            foreach (Photo? photo in photos)
            {
                if (photo is not null && filter(photo))
                {
                    yield return photo;
                }
            }
        }
        finally
        {
            _cacheSignal.Release();
        }
    }
}

```

In the preceding C# code:

- The constructor requires an `IMemoryCache`, `CacheSignal<Photo>`, and `ILogger`.
- The `GetPhotosAsync` method:
 - Defines a `Func<Photo, bool> filter` parameter, and returns an `IAsyncEnumerable<Photo>`.
 - Calls and waits for the `_cacheSignal.WaitAsync()` to release, this ensures that the cache is populated before accessing the cache.
 - Calls `_cache.GetOrCreateAsync()`, asynchronously getting all of the photos in the cache.
 - The `factory` argument logs a warning, and returns an empty photo array - this should never happen.
 - Each photo in the cache is iterated, filtered and materialized with `yield return`.
 - Finally, the cache signal is reset.

Consumers of this service are free to call `GetPhotosAsync` method, and handle photos accordingly. No

`HttpClient` is required as the cache contains the photos.

The `CacheWorker` is a subclass of [BackgroundService](#):

```
using System.Net.Http.Json;
using Microsoft.Extensions.Caching.Memory;

namespace CachingExamples.Memory;

public class CacheWorker : BackgroundService
{
    private readonly ILogger<CacheWorker> _logger;
    private readonly HttpClient _httpClient;
    private readonly CacheSignal<Photo> _cacheSignal;
    private readonly IMemoryCache _cache;
    private readonly TimeSpan _updateInterval = TimeSpan.FromHours(3);

    private const string Url = "https://jsonplaceholder.typicode.com/photos";

    public CacheWorker(
        ILogger<CacheWorker> logger,
        HttpClient httpClient,
        CacheSignal<Photo> cacheSignal,
        IMemoryCache cache) =>
        (_logger, _httpClient, _cacheSignal, _cache) = (logger, httpClient, cacheSignal, cache);

    public override async Task StartAsync(CancellationToken cancellationToken)
    {
        await _cacheSignal.WaitAsync();
        await base.StartAsync(cancellationToken);
    }

    protected override async Task ExecuteAsync(CancellationToken stoppingToken)
    {
        while (!stoppingToken.IsCancellationRequested)
        {
            _logger.LogInformation("Updating cache.");

            try
            {
                Photo[]? photos =
                    await _httpClient.GetFromJsonAsync<Photo[]>(
                        Url, stoppingToken);

                if (photos is { Length: > 0 })
                {
                    _cache.Set("Photos", photos);
                    _logger.LogInformation(
                        "Cache updated with {Count:#,#} photos.", photos.Length);
                }
                else
                {
                    _logger.LogWarning(
                        "Unable to fetch photos to update cache.");
                }
            }
            finally
            {
                _cacheSignal.Release();
            }
        }
    }

    try
    {
        _logger.LogInformation(
            "Will attempt to update the cache in {Hours} hours from now.",
            _updateInterval.Hours);
    }

    await Task.Delay(_updateInterval, stoppingToken);
}
```

```

        }
        catch (OperationCanceledException)
        {
            _logger.LogWarning("Cancellation acknowledged: shutting down.");
            break;
        }
    }
}

```

IMPORTANT

You need to `override BackgroundService.StartAsync` and call `await _cacheSignal.WaitAsync()` in order to prevent a race condition between the starting of the `CacheWorker` and a call to `PhotoService.GetPhotosAsync`.

In the preceding C# code:

- The constructor requires an `ILogger`, `HttpClient`, `CacheSignal<Photo>`, and `IMemoryCache`.
- It defines an `_updateInterval` of three hours.
- The `ExecuteAsync` method:
 - Loops while the app is running.
 - Makes an HTTP request to `"https://jsonplaceholder.typicode.com/photos"`, and maps the response as an array of `Photo` objects.
 - The array of photos is placed in the `IMemoryCache` under the `"Photos"` key.
 - The `_cacheSignal.Release()` is called, releasing any consumers who were waiting for the signal.
 - The call to `Task.Delay` is awaited, given the update interval.
 - After delaying for three hours, the cache is again updated.

The asynchronous signal is based on an encapsulated `SemaphoreSlim` instance, within a generic-type constrained singleton. The `CacheSignal<T>` relies on an instance of `SemaphoreSlim`:

```

namespace CachingExamples.Memory;

public sealed class CacheSignal<T>
{
    private readonly SemaphoreSlim _semaphore = new(1, 1);

    /// <summary>
    /// Exposes a <see cref="Task"/> that represents the asynchronous wait operation.
    /// When signaled (consumer calls <see cref="Release"/>), the
    /// <see cref="Task.Status"/> is set as <see cref="TaskStatus.RanToCompletion"/>.
    /// </summary>
    public Task WaitAsync() => _semaphore.WaitAsync();

    /// <summary>
    /// Exposes the ability to signal the release of the <see cref="WaitAsync"/>'s operation.
    /// Callers who were waiting, will be able to continue.
    /// </summary>
    public void Release() => _semaphore.Release();
}

```

In the preceding C# code, the decorator pattern is used to wrap an instance of the `SemaphoreSlim`. Since the `CacheSignal<T>` is registered as a singleton, it can be used across all service lifetimes with any generic type — in this case, the `Photo`. It is responsible for signaling the seeding of the cache.

Distributed caching

In some scenarios, a distributed cache is required — such is the case with multiple app servers. A distributed cache supports higher scale-out than the in-memory caching approach. Using a distributed cache offloads the cache memory to an external process, but does require extra network I/O and introduces a bit more latency (even if nominal).

The distributed caching abstractions are part of the [Microsoft.Extensions.Caching.Memory](#) NuGet package, and there is even an [AddDistributedMemoryCache](#) extension method.

Caution

The [AddDistributedMemoryCache](#) should only be used in development and/or testing scenarios, and is **not** a viable production implementation.

Consider any of the available implementations of the [IDistributedCache](#) from the following packages:

- [Microsoft.Extensions.Caching.SqlServer](#)
- [Microsoft.Extensions.Caching.StackExchangeRedis](#)
- [NCache.Microsoft.Extensions.Caching.OpenSource](#)

Distributed caching API

The distributed caching APIs are a bit more primitive than their in-memory caching API counterparts. The key-value pairs are a bit more basic. In-memory caching keys are based on an [object](#), whereas the distributed keys are a [string](#). With in-memory caching, the value can be any strongly-typed generic, whereas values in distributed caching are persisted as [byte\[\]](#). That's not to say that various implementations don't expose strongly-typed generic values but that would be an implementation detail.

Create values

To create values in the distributed cache, call one of the set APIs:

- [IDistributedCache.SetAsync](#)
- [IDistributedCache.Set](#)

Using the [AlphabetLetter](#) record from the in-memory cache example, you could serialize the object to JSON and then encode the [string](#) as a [byte\[\]](#):

```
DistributedCacheEntryOptions options = new()
{
    AbsoluteExpirationRelativeToNow =
        TimeSpan.FromMilliseconds(MillisecondsAbsoluteExpiration)
};

AlphabetLetter alphabetLetter = new(letter);
string json = JsonSerializer.Serialize(alphabetLetter);
byte[] bytes = Encoding.UTF8.GetBytes(json);

await cache.SetAsync(letter.ToString(), bytes, options);
```

Much like in-memory caching, cache entries can have options to help fine-tune their existence in the cache — in this case, the [DistributedCacheEntryOptions](#).

Create extension methods

There are several convenience-based extension methods for creating values, that help to avoid encoding [string](#) representations of objects into a [byte\[\]](#):

- [DistributedCacheExtensions.SetStringAsync](#)
- [DistributedCacheExtensions.SetString](#)

Read values

To read values from the distributed cache, call one of the get APIs:

- [IDistributedCache.GetAsync](#)
- [IDistributedCache.Get](#)

```
AlphabetLetter? alphabetLetter = null;
byte[]? bytes = await cache.GetAsync(letter.ToString());
if (bytes is { Length: > 0 })
{
    string json = Encoding.UTF8.GetString(bytes);
    alphabetLetter = JsonSerializer.Deserialize<AlphabetLetter>(json);
}
```

Once a cache entry is read out of the cache, you can get the UTF8 encoded `string` representation from the `byte[]`

Read extension methods

There are several convenience-based extension methods for reading values, that help to avoid decoding `byte[]` into `string` representations of objects:

- [DistributedCacheExtensions.GetStringAsync](#)
- [DistributedCacheExtensions.GetString](#)

Update values

There is no way to actually update the values in the distributed cache with a single API call, instead values can have their sliding expirations reset with one of the refresh APIs:

- [IDistributedCache.RefreshAsync](#)
- [IDistributedCache.Refresh](#)

If the actual value needs to be updated, you'd have to delete the value and then re-add it.

Delete values

To delete values in the distributed cache, call one of the remove APIs:

- [IDistributedCache.RemoveAsync](#)
- [IDistributedCache.Remove](#)

TIP

While there are synchronous versions of the aforementioned APIs, please consider the fact that implementations of distributed caches are reliant on network I/O. For this reason, it is preferred more often than not to use the asynchronous APIs.

See also

- [Dependency injection in .NET](#)
- [.NET Generic Host](#)
- [Worker Services in .NET](#)
- [Azure for .NET developers](#)
- [Cache in-memory in ASP.NET Core](#)
- [Distributed caching in ASP.NET Core](#)

System.Threading.Channels library

9/20/2022 • 8 minutes to read • [Edit Online](#)

The [System.Threading.Channels](#) namespace provides a set of synchronization data structures for passing data between producers and consumers asynchronously. The library targets .NET Standard and works on all .NET implementations.

This library is available in the [System.Threading.Channels](#) NuGet package (or included when targeting .NET Core 2.1+).

Producer/consumer conceptual programming model

Channels are an implementation of the producer/consumer conceptual programming model. In this programming model, producers asynchronously produce data, and consumers asynchronously consume that data. In other words, this model hands off data from one party to another. Try to think of channels as you would any other common generic collection type, such as a `List<T>`. The primary difference is that this collection manages synchronization and provides various consumption models through factory creation options. These options control the behavior of the channels, such as how many elements they're allowed to store and what happens if that limit is reached, or whether the channel may be accessed by multiple producers or multiple consumers concurrently.

Bounding strategies

Depending on how a `Channel<T>` is created, its reader and writer will behave differently.

To create a channel that specifies a maximum capacity, call [Channel.CreateBounded](#). To create a channel that can be used by any number of readers and writers concurrently, call [Channel.CreateUnbounded](#). Each bounding strategy exposes various creator-defined options, either [BoundedChannelOptions](#) or [UnboundedChannelOptions](#) respectively.

NOTE

Regardless of the bounding strategy, a channel will always throw a [ChannelClosedException](#) when it's used after it's been closed.

Unbounded channels

To create an unbounded channel, call one of the [Channel.CreateUnbounded](#) overloads:

```
var channel = Channel.CreateUnbounded<T>();
```

When you create an unbounded channel, by default, the channel can be used by any number of readers and writers concurrently. Alternatively, you can specify non-default behavior when creating an unbounded channel by providing an `UnboundedChannelOptions` instance. The channel's capacity is unbounded and all writes are performed synchronously. For additional examples, see [Unbounded creation patterns](#).

Bounded channels

To create a bounded channel, call one of the [Channel.CreateBounded](#) overloads:

```
var channel = Channel.CreateBounded<T>(7);
```

The preceding code creates a channel that has a maximum capacity of 7 items. When you create a bounded channel, the channel is bound to a maximum capacity. When the bound is reached, the default behavior is that the channel will asynchronously block the producer until space becomes available. You can configure this behavior by specifying an option when you create the channel. Bounded channels can be created with any capacity value greater than zero. For additional examples, see [Bounded creation patterns](#).

Full mode behavior

When using a bounded channel, you can specify the behavior the channel will adhere to when the configured bound is reached. The following table lists the full mode behaviors for each `BoundedChannelFullMode` value:

VALUE	BEHAVIOR
<code>BoundedChannelFullMode.Wait</code>	This is the default value. When calling <code>WriteAsync</code> , waits for space to be available in order to complete the write operation. When calling <code>TryWrite</code> , returns <code>false</code> immediately.
<code>BoundedChannelFullMode.DropNewest</code>	Removes and ignores the newest item in the channel in order to make room for the item being written.
<code>BoundedChannelFullMode.DropOldest</code>	Removes and ignores the oldest item in the channel in order to make room for the item being written.
<code>BoundedChannelFullMode.DropWrite</code>	Drops the item being written.

IMPORTANT

Whenever a `Channel<TWrite,TRead>.Writer` produces faster than a `Channel<TWrite,TRead>.Reader` can consume, the channel's writer experiences back pressure.

Producer APIs

The producer functionality is exposed on the `Channel<TWrite,TRead>.Writer`. The producer APIs and expected behavior are detailed in the following table:

API	EXPECTED BEHAVIOR
<code>ChannelWriter<T>.Complete</code>	Marks the channel as being complete, meaning no more items will be written to it.
<code>ChannelWriter<T>.TryComplete</code>	Attempts to mark the channel as being completed, meaning no more data will be written to it.
<code>ChannelWriter<T>.TryWrite</code>	Attempts to write the specified item to the channel. When used with an unbounded channel, this always returns <code>true</code> unless the channel's writer signals completion with either <code>ChannelWriter<T>.Complete</code> , or <code>ChannelWriter<T>.TryComplete</code> .
<code>ChannelWriter<T>.WaitToWriteAsync</code>	Returns a <code>ValueTask<TResult></code> that will complete when space is available to write an item.

API	EXPECTED BEHAVIOR
<code>ChannelWriter<T>.WriteAsync</code>	Asynchronously writes an item to the channel.

Consumer APIs

The consumer functionality is exposed on the `Channel<TWrite,TRead>.Reader`. The consumer APIs and expected behavior are detailed in the following table:

API	EXPECTED BEHAVIOR
<code>ChannelReader<T>.ReadAllAsync</code>	Creates an <code>IAsyncEnumerable<T></code> that enables reading all of the data from the channel.
<code>ChannelReader<T>.ReadAsync</code>	Asynchronously reads an item from the channel.
<code>ChannelReader<T>.TryPeek</code>	Attempts to peek at an item from the channel.
<code>ChannelReader<T>.TryRead</code>	Attempts to read an item from the channel.
<code>ChannelReader<T>.WaitToReadAsync</code>	Returns a <code>ValueTask<TResult></code> that will complete when data is available to read.

Common usage patterns

There are several usage patterns for channels. The API is designed to be simple, consistent, and as flexible as possible. All of the asynchronous methods return a `ValueTask` (or `ValueTask<bool>`) that represents a lightweight asynchronous operation that can avoid allocating if the operation completes synchronously and potentially even asynchronously. Additionally, the API is designed to be composable, in that the creator of a channel makes promises about its intended usage. When a channel is created with certain parameters, the internal implementation can operate more efficiently knowing these promises.

Creation patterns

Imagine that you're creating a producer/consumer solution for a global position system (GPS). You want to track the coordinates of a device over time. A sample coordinates object might look like this:

```
/// <summary>
/// A representation of a device's coordinates,
/// which includes latitude and longitude.
/// </summary>
/// <param name="DeviceId">A unique device identifier.</param>
/// <param name="Latitude">The latitude of the device.</param>
/// <param name="Longitude">The longitude of the device.</param>
public readonly record struct Coordinates(
    Guid DeviceId,
    double Latitude,
    double Longitude);
```

Unbounded creation patterns

One common usage pattern is to create a default unbounded channel:

```
var channel = Channel.CreateUnbounded<Coordinates>();
```

But instead, let's imagine that you want to create an unbounded channel with multiple producers and

consumers:

```
var channel = Channel.CreateUnbounded<Coordinates>(
    new UnboundedChannelOptions
    {
        SingleWriter = false,
        SingleReader = false,
        AllowSynchronousContinuations = true
    });
});
```

In this case, all writes are synchronous, even the `WriteAsync`. This is because an unbounded channel always has available room for a write effectively immediately. However, with `AllowSynchronousContinuations` set to `true`, the writes may end up doing work associated with a reader by executing their continuations. This doesn't affect the synchronicity of the operation.

Bounded creation patterns

When working with bounded channels, the configurability of the channel should be known to the consumer to help ensure proper consumption. That is, the consumer should know what behavior the channel will exhibit when the configured bound is reached. Let's explore some of the common bounded creation patterns.

The simplest way to create a bounded channel is to specify a capacity:

```
var channel = Channel.CreateBounded<Coordinates>(1);
```

The preceding code creates a bounded channel with a max capacity of `1`. Additional options are available, some options are the same as an unbounded channel, while others are specific to unbounded channels:

```
var channel = Channel.CreateBounded<Coordinates>(
    new BoundedChannelOptions(1_000)
    {
        SingleWriter = true,
        SingleReader = false,
        AllowSynchronousContinuations = false,
        FullMode = BoundedChannelFullMode.DropWrite
    });
});
```

In the preceding code, the channel is created as a bounded channel that's limited to 1,000 items, with a single writer but many readers. Its full mode behavior is defined as `DropWrite`, which means that it will drop the item being written if the channel is full.

When using a bounded channel, to observe items that are dropped register an `itemDropped` callback:

```
var channel = Channel.CreateBounded(
    new BoundedChannelOptions(10)
    {
        AllowSynchronousContinuations = true,
        FullMode = BoundedChannelFullMode.DropOldest
    },
    static void (Coordinates dropped) =>
        Console.WriteLine($"Coordinates dropped: {dropped}"));
});
```

Whenever the channel is full and a new item is added, the `itemDropped` callback will be invoked. In this example, the provided callback writes the item to the console, but you're free to take any other action you want.

Producer patterns

Imagine that the producer in this scenario is writing new coordinates to the channel. The producer can do this by calling `TryWrite`:

```

static void ProduceWithWhileAndTryWrite(
    ChannelWriter<Coordinates> writer, Coordinates coordinates)
{
    while (coordinates is { Latitude: < 90, Longitude: < 180 })
    {
        var tempCoordinates = coordinates with
        {
            Latitude = coordinates.Latitude + .5,
            Longitude = coordinates.Longitude + 1
        };

        if (writer.TryWrite(item: tempCoordinates))
        {
            coordinates = tempCoordinates;
        }
    }
}

```

The preceding producer code:

- Accepts the `Channel<Coordinates>.Writer` (`ChannelWriter<Coordinates>`) as an argument, along with the initial `Coordinates`.
- Defines a conditional `while` loop that attempts to move the coordinates using `TryWrite`.

An alternative producer might use the `WriteAsync` method:

```

static async ValueTask ProduceWithWhileWriteAsync(
    ChannelWriter<Coordinates> writer, Coordinates coordinates)
{
    while (coordinates is { Latitude: < 90, Longitude: < 180 })
    {
        await writer.WriteAsync(
            item: coordinates = coordinates with
            {
                Latitude = coordinates.Latitude + .5,
                Longitude = coordinates.Longitude + 1
            });
    }

    writer.Complete();
}

```

Again, the `Channel<Coordinates>.Writer` is used within a `while` loop. But this time, the `WriteAsync` method is called. The method will continue only after the coordinates have been written. When the `while` loop exits, a call to `Complete` is made, which signals that no more data will be written to the channel.

Another producer pattern is to use the `WaitToWriteAsync` method, consider the following code:

```

static async ValueTask ProduceWithWaitToWriteAsync(
    ChannelWriter<Coordinates> writer, Coordinates coordinates)
{
    while (coordinates is { Latitude: < 90, Longitude: < 180 } &&
        await writer.WaitToWriteAsync())
    {
        var tempCoordinates = coordinates with
        {
            Latitude = coordinates.Latitude + .5,
            Longitude = coordinates.Longitude + 1
        };

        if (writer.TryWrite(item: tempCoordinates))
        {
            coordinates = tempCoordinates;
        }

        await Task.Delay(TimeSpan.FromMilliseconds(10));
    }

    writer.Complete();
}

```

As part of the conditional `while`, the result of the `WaitToWriteAsync` call is used to determine whether to continue the loop.

Consumer patterns

There are several common channel consumer patterns. When a channel is never ending, meaning it will infinitely produce data, the consumer could use a `while (true)` loop, and read data as it becomes available:

```

static async ValueTask ConsumeWithWhileAsync(
    ChannelReader<Coordinates> reader)
{
    while (true)
    {
        // May throw ChannelClosedException if
        // the parent channel's writer signals complete.
        Coordinates coordinates = await reader.ReadAsync();
        Console.WriteLine(coordinates);
    }
}

```

NOTE

This code will throw an exception if the channel is closed.

An alternative consumer could avoid this concern by using a nested while loop, as shown in the following code:

```

static async ValueTask ConsumeWithNestedWhileAsync(
    ChannelReader<Coordinates> reader)
{
    while (await reader.WaitToReadAsync())
    {
        while (reader.TryRead(out Coordinates coordinates))
        {
            Console.WriteLine(coordinates);
        }
    }
}

```

In the preceding code, the consumer waits to read data. Once the data is available, the consumer tries to read it. These loops will continue to evaluate until the producer of the channel signals that it no longer has data to be read. With that said, when a producer is known to have a finite number of items it will produce and it signals completion, the consumer can use `await foreach` semantics to iterate over the items:

```
static async ValueTask ConsumeWithAwaitForeachAsync(
    ChannelReader<Coordinates> reader)
{
    await foreach (Coordinates coordinates in reader.ReadAllAsync())
    {
        Console.WriteLine(coordinates);
    }
}
```

The preceding code uses the [ReadAllAsync](#) method to read all of the coordinates from the channel.

See also

- [On .NET show: Working with Channels in .NET](#)
- [.NET Blog: An Introduction to System.Threading.Channels](#)
- [Managed threading basics](#)

LINQ overview

9/20/2022 • 8 minutes to read • [Edit Online](#)

Language-Integrated Query (LINQ) provides language-level querying capabilities, and a [higher-order function](#) API to C# and Visual Basic, that enable you to write expressive declarative code.

Language-level query syntax

This is the language-level query syntax:

```
var linqExperts = from p in programmers
                  where p.IsNewToLINQ
                  select new LINQExpert(p);
```

```
Dim linqExperts = From p in programmers
                  Where p.IsNewToLINQ
                  Select New LINQExpert(p)
```

This is the same example using the `IEnumerable<T>` API:

```
var linqExperts = programmers.Where(p => p.IsNewToLINQ)
                               .Select(p => new LINQExpert(p));
```

```
Dim linqExperts = programmers.Where(Function(p) p.IsNewToLINQ).
                               Select(Function(p) New LINQExpert(p))
```

LINQ is expressive

Imagine you have a list of pets, but want to convert it into a dictionary where you can access a pet directly by its `RFID` value.

This is traditional imperative code:

```
var petLookup = new Dictionary<int, Pet>();

foreach (var pet in pets)
{
    petLookup.Add(pet.RFID, pet);
}
```

```
Dim petLookup = New Dictionary(Of Integer, Pet)()

For Each pet in pets
    petLookup.Add(pet.RFID, pet)
Next
```

The intention behind the code isn't to create a new `Dictionary<int, Pet>` and add to it via a loop, it's to convert an existing list into a dictionary! LINQ preserves the intention whereas the imperative code doesn't.

This is the equivalent LINQ expression:

```
var petLookup = pets.ToDictionary(pet => pet.RFID);
```

```
Dim petLookup = pets.ToDictionary(Function(pet) pet.RFID)
```

The code using LINQ is valuable because it evens the playing field between intent and code when reasoning as a programmer. Another bonus is code brevity. Imagine reducing large portions of a codebase by 1/3 as done above. Sweet deal, right?

LINQ providers simplify data access

For a significant chunk of software out in the wild, everything revolves around dealing with data from some source (Databases, JSON, XML, and so on). Often this involves learning a new API for each data source, which can be annoying. LINQ simplifies this by abstracting common elements of data access into a query syntax that looks the same no matter which data source you pick.

This finds all XML elements with a specific attribute value:

```
public static IEnumerable< XElement> FindAllElementsWithAttribute(XElement documentRoot, string elementName,
                                                               string attributeName, string value)
{
    return from el in documentRoot.Elements(elementName)
           where (string)el.Element(attributeName) == value
           select el;
}
```

```
Public Shared Function FindAllElementsWithAttribute(documentRoot As XElement, elementName As String,
                                                   attributeName As String, value As String) As IEnumerable(Of
XElement)
    Return From el In documentRoot.Elements(elementName)
           Where el.Element(attributeName).ToString() = value
           Select el
End Function
```

Writing code to manually traverse the XML document to do this task would be far more challenging.

Interacting with XML isn't the only thing you can do with LINQ Providers. [Linq to SQL](#) is a fairly bare-bones Object-Relational Mapper (ORM) for an MSSQL Server Database. The [Json.NET](#) library provides efficient JSON Document traversal via LINQ. Furthermore, if there isn't a library that does what you need, you can also [write your own LINQ Provider!](#)

Reasons to use the query syntax

Why use query syntax? This is a question that often comes up. After all, the following code:

```
var filteredItems = myItems.Where(item => item.Foo);
```

```
Dim filteredItems = myItems.Where(Function(item) item.Foo)
```

is a lot more concise than this:

```
var filteredItems = from item in myItems
                    where item.Foo
                    select item;
```

```
Dim filteredItems = From item In myItems
                      Where item.Foo
                      Select item
```

Isn't the API syntax just a more concise way to do the query syntax?

No. The query syntax allows for the use of the `let` clause, which allows you to introduce and bind a variable within the scope of the expression, using it in subsequent pieces of the expression. Reproducing the same code with only the API syntax can be done, but will most likely lead to code that's hard to read.

So this begs the question, **should you just use the query syntax?**

The answer to this question is **yes** if:

- Your existing codebase already uses the query syntax.
- You need to scope variables within your queries because of complexity.
- You prefer the query syntax and it won't distract from your codebase.

The answer to this question is **no** if...

- Your existing codebase already uses the API syntax
- You have no need to scope variables within your queries
- You prefer the API syntax and it won't distract from your codebase

Essential LINQ

For a truly comprehensive list of LINQ samples, visit [101 LINQ Samples](#).

The following examples are a quick demonstration of some of the essential pieces of LINQ. This is in no way comprehensive, as LINQ provides more functionality than what is showcased here.

The bread and butter - `Where` , `Select` , **and** `Aggregate`

```
// Filtering a list.
var germanShepherds = dogs.Where(dog => dog.Breed == DogBreed.GermanShepherd);

// Using the query syntax.
var queryGermanShepherds = from dog in dogs
                           where dog.Breed == DogBreed.GermanShepherd
                           select dog;

// Mapping a list from type A to type B.
var cats = dogs.Select(dog => dog.TurnIntoACat());

// Using the query syntax.
var queryCats = from dog in dogs
                  select dog.TurnIntoACat();

// Summing the lengths of a set of strings.
int seed = 0;
int sumOfStrings = strings.Aggregate(seed, (s1, s2) => s1.Length + s2.Length);
```

```

' Filtering a list.
Dim germanShepherds = dogs.Where(Function(dog) dog.Breed = DogBreed.GermanShepherd)

' Using the query syntax.
Dim queryGermanShepherds = From dog In dogs
    Where dog.Breed = DogBreed.GermanShepherd
    Select dog

' Mapping a list from type A to type B.
Dim cats = dogs.Select(Function(dog) dog.TurnIntoACat())

' Using the query syntax.
Dim queryCats = From dog In dogs
    Select dog.TurnIntoACat()

' Summing the lengths of a set of strings.
Dim seed As Integer = 0
Dim sumOfStrings As Integer = strings.Aggregate(seed, Function(s1, s2) s1.Length + s2.Length)

```

Flattening a list of lists

```

// Transforms the list of kennels into a list of all their dogs.
var allDogsFromKennels = kennels.SelectMany(kennel => kennel.Dogs);

```

```

' Transforms the list of kennels into a list of all their dogs.
Dim allDogsFromKennels = kennels.SelectMany(Function(kennel) kennel.Dogs)

```

Union between two sets (with custom comparator)

```

public class DogHairLengthComparer : IEqualityComparer<Dog>
{
    public bool Equals(Dog a, Dog b)
    {
        if (a == null && b == null)
        {
            return true;
        }
        else if ((a == null && b != null) ||
                  (a != null && b == null))
        {
            return false;
        }
        else
        {
            return a.HairLengthType == b.HairLengthType;
        }
    }

    public int GetHashCode(Dog d)
    {
        // Default hashcode is enough here, as these are simple objects.
        return d.GetHashCode();
    }
}

// Gets all the short-haired dogs between two different kennels.
var allShortHairedDogs = kennel1.Dogs.Union(kennel2.Dogs, new DogHairLengthComparer());

```

```

Public Class DogHairLengthComparer
    Inherits IEqualityComparer(Of Dog)

    Public Function Equals(a As Dog, b As Dog) As Boolean
        If a Is Nothing AndAlso b Is Nothing Then
            Return True
        ElseIf (a Is Nothing AndAlso b IsNot Nothing) OrElse (a IsNot Nothing AndAlso b Is Nothing) Then
            Return False
        Else
            Return a.HairLengthType = b.HairLengthType
        End If
    End Function

    Public Function GetHashCode(d As Dog) As Integer
        ' Default hashcode is enough here, as these are simple objects.
        Return d.GetHashCode()
    End Function
End Class

...

' Gets all the short-haired dogs between two different kennels.
Dim allShortHairedDogs = kennel1.Dogs.Union(kennel2.Dogs, New DogHairLengthComparer())

```

Intersection between two sets

```

// Gets the volunteers who spend share time with two humane societies.
var volunteers = humaneSociety1.Volunteers.Intersect(humaneSociety2.Volunteers,
    new VolunteerTimeComparer());

```

```

' Gets the volunteers who spend share time with two humane societies.
Dim volunteers = humaneSociety1.Volunteers.Intersect(humaneSociety2.Volunteers,
    New VolunteerTimeComparer())

```

Ordering

```

// Get driving directions, ordering by if it's toll-free before estimated driving time.
var results = DirectionsProcessor.GetDirections(start, end)
    .OrderBy(direction => direction.HasNoTolls)
    .ThenBy(direction => direction.EstimatedTime);

```

```

' Get driving directions, ordering by if it's toll-free before estimated driving time.
Dim results = DirectionsProcessor.GetDirections(start, end).
    OrderBy(Function(direction) direction.HasNoTolls).
    ThenBy(Function(direction) direction.EstimatedTime)

```

Equality of instance properties

Finally, a more advanced sample: determining if the values of the properties of two instances of the same type are equal (Borrowed and modified from [this StackOverflow post](#)):

```

public static bool PublicInstancePropertiesEqual<T>(this T self, T to, params string[] ignore) where T : class
{
    if (self == null || to == null)
    {
        return self == to;
    }

    // Selects the properties which have unequal values into a sequence of those properties.
    var unequalProperties = from property in typeof(T).GetProperties(BindingFlags.Public | BindingFlags.Instance)
                            where !ignore.Contains(property.Name)
                            let selfValue = property.GetValue(self, null)
                            let toValue = property.GetValue(to, null)
                            where !Equals(selfValue, toValue)
                            select property;
    return !unequalProperties.Any();
}

```

```

<System.Runtime.CompilerServices.Extension()>
Public Function PublicInstancePropertiesEqual(Of T As Class)(self As T, [to] As T, ParamArray ignore As String()) As Boolean
    If self Is Nothing OrElse [to] Is Nothing Then
        Return self Is [to]
    End If

    ' Selects the properties which have unequal values into a sequence of those properties.
    Dim unequalProperties = From [property] In GetType(T).GetProperties(BindingFlags.Public Or
BindingFlags.Instance)
                            Where Not ignore.Contains([property].Name)
                            Let selfValue = [property].GetValue(self, Nothing)
                            Let toValue = [property].GetValue([to], Nothing)
                            Where Not Equals(selfValue, toValue) Select [property]
    Return Not unequalProperties.Any()
End Function

```

PLINQ

PLINQ or Parallel LINQ is a parallel execution engine for LINQ expressions. In other words, a regular LINQ expression can be trivially parallelized across any number of threads. This is accomplished via a call to `AsParallel()` preceding the expression.

Consider the following:

```

public static string GetAllFacebookUserLikesMessage(IEnumerable<FacebookUser> facebookUsers)
{
    var seed = default(UInt64);

    Func<UInt64, UInt64, UInt64> threadAccumulator = (t1, t2) => t1 + t2;
    Func<UInt64, UInt64, UInt64> threadResultAccumulator = (t1, t2) => t1 + t2;
    Func<UInt64, string> resultSelector = total => $"Facebook has {total} likes!";

    return facebookUsers.AsParallel()
                        .Aggregate(seed, threadAccumulator, threadResultAccumulator, resultSelector);
}

```

```

Public Shared GetAllFacebookUserLikesMessage(facebookUsers As IEnumerable(Of FacebookUser)) As String
{
    Dim seed As UInt64 = 0

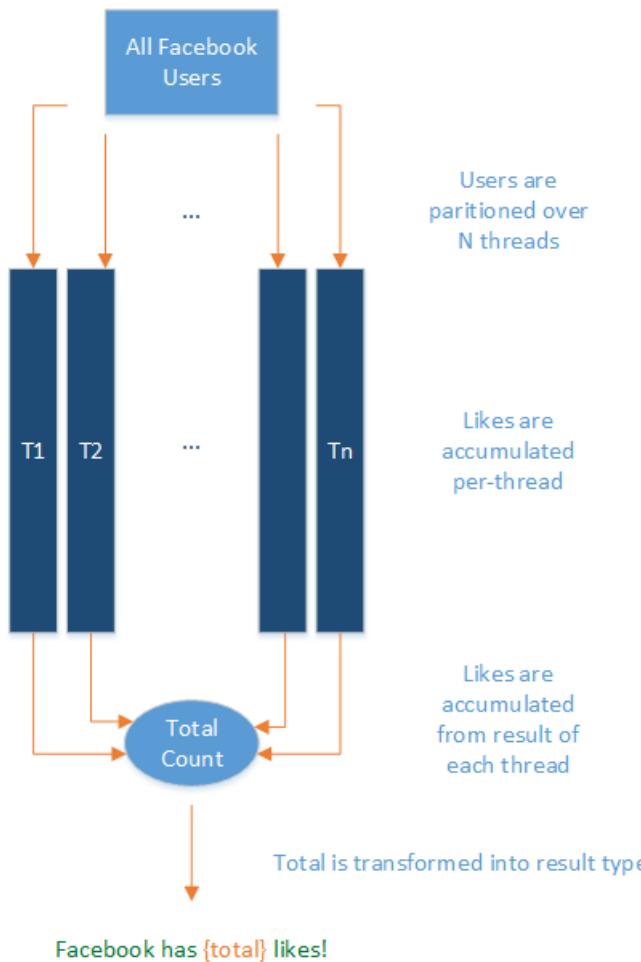
    Dim threadAccumulator As Func(Of UInt64, UInt64, UInt64) = Function(t1, t2) t1 + t2
    Dim threadResultAccumulator As Func(Of UInt64, UInt64, UInt64) = Function(t1, t2) t1 + t2
    Dim resultSelector As Func(Of UInt64, String) = Function(total) $"Facebook has {total} likes!"

    Return facebookUsers.AsParallel().
        Aggregate(seed, threadAccumulator, threadResultAccumulator, resultSelector)
}

```

This code will partition `facebookUsers` across system threads as necessary, sum up the total likes on each thread in parallel, sum the results computed by each thread, and project that result into a nice string.

In diagram form:



Parallelizable CPU-bound jobs that can be easily expressed via LINQ (in other words, are pure functions and have no side effects) are a great candidate for PLINQ. For jobs that *do* have a side effect, consider using the [Task Parallel Library](#).

More resources

- [101 LINQ Samples](#)
- [Linqpad](#), a playground environment and Database querying engine for C#/F#/Visual Basic
- [EduLinq](#), an e-book for learning how LINQ-to-objects is implemented

XML Documents and Data

9/20/2022 • 3 minutes to read • [Edit Online](#)

The .NET Framework provides a comprehensive and integrated set of classes that enable you to build XML-aware apps easily. The classes in the following namespaces support parsing and writing XML, editing XML data in memory, data validation, and XSLT transformation.

- [System.Xml](#)
- [System.Xml.XPath](#)
- [System.Xml.Xsl](#)
- [System.Xml.Schema](#)
- [System.Xml.Linq](#)

For a full list, search for "System.Xml" on the [.NET API browser](#).

The classes in these namespaces support World Wide Web Consortium (W3C) recommendations. For example:

- The [System.Xml.XmlDocument](#) class implements the [W3C Document Object Model \(DOM\) Level 1 Core](#) and [DOM Level 2 Core](#) recommendations.
- The [System.Xml.XmlReader](#) and [System.Xml.XmlWriter](#) classes support the [W3C XML 1.0](#) and the [Namespaces in XML](#) recommendations.
- Schemas in the [System.Xml.Schema.XmlSchemaSet](#) class support the [W3C XML Schema Part 1: Structures](#) and [XML Schema Part 2: Datatypes](#) recommendations.
- Classes in the [System.Xml.Xsl](#) namespace support XSLT transformations that conform to the [W3C XSLT 1.0](#) recommendation.

The XML classes in the .NET Framework provide these benefits:

- **Productivity.** [LINQ to XML \(C#\)](#) and [LINQ to XML \(Visual Basic\)](#) makes it easier to program with XML and provides a query experience that is similar to SQL.
- **Extensibility.** The XML classes in the .NET Framework are extensible through the use of abstract base classes and virtual methods. For example, you can create a derived class of the [XmlUrlResolver](#) class that stores the cache stream to the local disk.
- **Pluggable architecture.** The .NET Framework provides an architecture in which components can utilize one another, and data can be streamed between components. For example, a data store, such as an [XPathDocument](#) or [XmlDocument](#) object, can be transformed with the [XslCompiledTransform](#) class, and the output can then be streamed either into another store or returned as a stream from a web service.
- **Performance.** For better app performance, some of the XML classes in the .NET Framework support a streaming-based model with the following characteristics:
 - Minimal caching for forward-only, pull-model parsing ([XmlReader](#)).
 - Forward-only validation ([XmlReader](#)).
 - Cursor style navigation that minimizes node creation to a single virtual node while providing random access to the document ([XPathNavigator](#)).

For better performance whenever XSLT processing is required, you can use the [XPathDocument](#) class, which is an optimized, read-only store for XPath queries designed to work efficiently with the [XslCompiledTransform](#) class.

- **Integration with ADO.NET.** The XML classes and [ADO.NET](#) are tightly integrated to bring together relational data and XML. The [DataSet](#) class is an in-memory cache of data retrieved from a database. The [DataSet](#) class has the ability to read and write XML by using the [XmlReader](#) and [XmlWriter](#) classes, to persist its internal relational schema structure as XML schemas (XSD), and to infer the schema structure of an XML document.

In This Section

[XML Processing Options](#) Discusses options for processing XML data.

[Processing XML Data In-Memory](#) Discusses the three models for processing XML data in-memory: [LINQ to XML \(C#\)](#) and [LINQ to XML \(Visual Basic\)](#), the [XmlDocument](#) class (based on the W3C Document Object Model), and the [XPathDocument](#) class (based on the XPath data model).

[XSLT Transformations](#)

Describes how to use the XSLT processor.

[XML Schema Object Model \(SOM\)](#)

Describes the classes used for building and manipulating XML Schemas (XSD) by providing an [XmlSchema](#) class to load and edit a schema.

[XML Integration with Relational Data and ADO.NET](#)

Describes how the .NET Framework enables real-time, synchronous access to both the relational and hierarchical representations of data through the [DataSet](#) object and the [XmlDataDocument](#) object.

[Managing Namespaces in an XML Document](#)

Describes how the [XmlNamespaceManager](#) class is used to store and maintain namespace information.

[Type Support in the System.Xml Classes](#)

Describes how XML data types map to CLR types, how to convert XML data types, and other type support features in the [System.Xml](#) classes.

Related Sections

[ADO.NET](#)

Provides information on how to access data using ADO.NET.

[Security](#)

Provides an overview of the .NET Framework security system.

Microsoft.Data.Sqlite overview

9/20/2022 • 2 minutes to read • [Edit Online](#)

Microsoft.Data.Sqlite is a lightweight [ADO.NET](#) provider for SQLite. The [Entity Framework Core](#) provider for SQLite is built on top of this library. However, it can also be used independently or with other data access libraries.

Installation

The latest stable version is available on [NuGet](#).

- [.NET Core CLI](#)
- [Visual Studio](#)

```
dotnet add package Microsoft.Data.Sqlite
```

Usage

This library implements the common ADO.NET abstractions for connections, commands, data readers, and so on.

```
using (var connection = new SqliteConnection("Data Source=hello.db"))
{
    connection.Open();

    var command = connection.CreateCommand();
    command.CommandText =
    @"
        SELECT name
        FROM user
        WHERE id = $id
    ";
    command.Parameters.AddWithValue("$id", id);

    using (var reader = command.ExecuteReader())
    {
        while (reader.Read())
        {
            var name = reader.GetString(0);

            Console.WriteLine($"Hello, {name}!");
        }
    }
}
```

TIP

You can see the full code for this example at [HelloWorldSample](#).

See also

- [Connection strings](#)

- [API Reference](#)
- [SQL Syntax](#)

Asynchronous programming patterns

9/20/2022 • 2 minutes to read • [Edit Online](#)

.NET provides three patterns for performing asynchronous operations:

- **Task-based Asynchronous Pattern (TAP)**, which uses a single method to represent the initiation and completion of an asynchronous operation. TAP was introduced in .NET Framework 4. It's the recommended approach to asynchronous programming in .NET. The `async` and `await` keywords in C# and the `Async` and `Await` operators in Visual Basic add language support for TAP. For more information, see [Task-based Asynchronous Pattern \(TAP\)](#).
- **Event-based Asynchronous Pattern (EAP)**, which is the event-based legacy model for providing asynchronous behavior. It requires a method that has the `Async` suffix and one or more events, event handler delegate types, and `EventArgs`-derived types. EAP was introduced in .NET Framework 2.0. It's no longer recommended for new development. For more information, see [Event-based Asynchronous Pattern \(EAP\)](#).
- **Asynchronous Programming Model (APM)** pattern (also called the `IAsyncResult` pattern), which is the legacy model that uses the `IAsyncResult` interface to provide asynchronous behavior. In this pattern, synchronous operations require `Begin` and `End` methods (for example, `BeginWrite` and `EndWrite`) to implement an asynchronous write operation). This pattern is no longer recommended for new development. For more information, see [Asynchronous Programming Model \(APM\)](#).

Comparison of patterns

For a quick comparison of how the three patterns model asynchronous operations, consider a `Read` method that reads a specified amount of data into a provided buffer starting at a specified offset:

```
public class MyClass
{
    public int Read(byte [] buffer, int offset, int count);
}
```

The TAP counterpart of this method would expose the following single `ReadAsync` method:

```
public class MyClass
{
    public Task<int> ReadAsync(byte [] buffer, int offset, int count);
}
```

The EAP counterpart would expose the following set of types and members:

```
public class MyClass
{
    public void ReadAsync(byte [] buffer, int offset, int count);
    public event ReadCompletedEventHandler ReadCompleted;
}
```

The APM counterpart would expose the `BeginRead` and `EndRead` methods:

```
public class MyClass
{
    public IAsyncResult BeginRead(
        byte [] buffer, int offset, int count,
        AsyncCallback callback, object state);
    public int EndRead(IAsyncResult asyncResult);
}
```

See also

- [C# - Asynchronous programming with async and await](#)
- [Visual Basic - Asynchronous Programming with Async and Await](#)
- [F# - Asynchronous Programming](#)

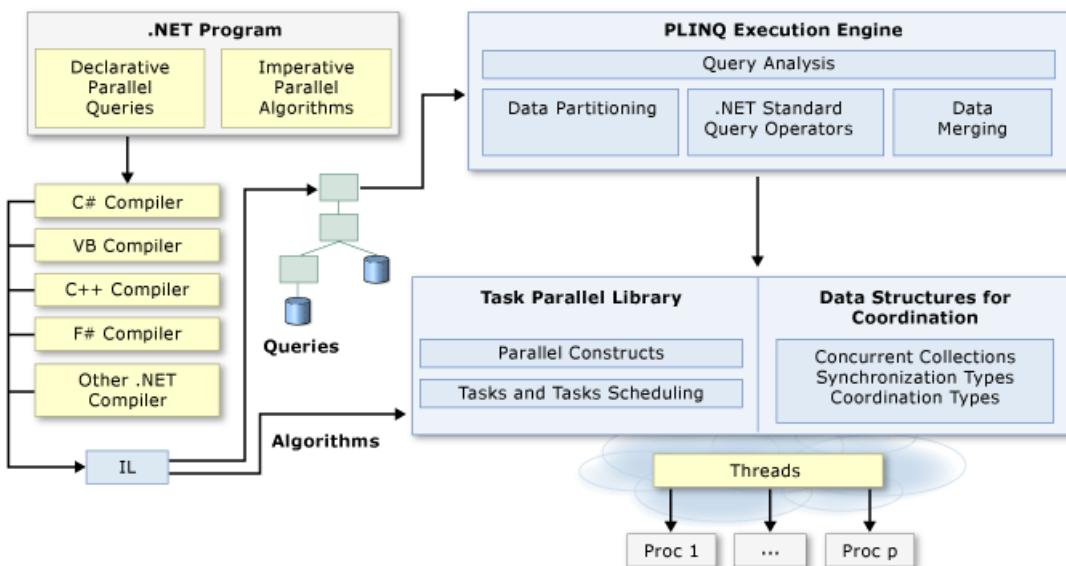
Parallel programming in .NET: A guide to the documentation

9/20/2022 • 2 minutes to read • [Edit Online](#)

Many personal computers and workstations have multiple CPU cores that enable multiple threads to be executed simultaneously. To take advantage of the hardware, you can parallelize your code to distribute work across multiple processors.

In the past, parallelization required low-level manipulation of threads and locks. Visual Studio and .NET enhance support for parallel programming by providing a runtime, class library types, and diagnostic tools. These features, which were introduced in .NET Framework 4, simplify parallel development. You can write efficient, fine-grained, and scalable parallel code in a natural idiom without having to work directly with threads or the thread pool.

The following illustration provides a high-level overview of the parallel programming architecture in .NET.



Related Topics

TECHNOLOGY	DESCRIPTION
Task Parallel Library (TPL)	Provides documentation for the <code>System.Threading.Tasks.Parallel</code> class, which includes parallel versions of <code>For</code> and <code>ForEach</code> loops, and also for the <code>System.Threading.Tasks.Task</code> class, which represents the preferred way to express asynchronous operations.
Parallel LINQ (PLINQ)	A parallel implementation of LINQ to Objects that significantly improves performance in many scenarios.
Data Structures for Parallel Programming	Provides links to documentation for thread-safe collection classes, lightweight synchronization types, and types for lazy initialization.

TECHNOLOGY	DESCRIPTION
Parallel Diagnostic Tools	Provides links to documentation for Visual Studio debugger windows for tasks and parallel stacks, and for the Concurrency Visualizer .
Custom Partitioners for PLINQ and TPL	Describes how partitioners work and how to configure the default partitioners or create a new partitioner.
Task Schedulers	Describes how schedulers work and how the default schedulers may be configured.
Lambda Expressions in PLINQ and TPL	Provides a brief overview of lambda expressions in C# and Visual Basic, and shows how they are used in PLINQ and the Task Parallel Library.
For Further Reading	Provides links to additional information and sample resources for parallel programming in .NET.

See also

- [Managed Threading](#)
- [Asynchronous programming patterns](#)

Task Parallel Library (TPL)

9/20/2022 • 2 minutes to read • [Edit Online](#)

The Task Parallel Library (TPL) is a set of public types and APIs in the [System.Threading](#) and [System.Threading.Tasks](#) namespaces. The purpose of the TPL is to make developers more productive by simplifying the process of adding parallelism and concurrency to applications. The TPL scales the degree of concurrency dynamically to most efficiently use all the processors that are available. In addition, the TPL handles the partitioning of the work, the scheduling of threads on the [ThreadPool](#), cancellation support, state management, and other low-level details. By using TPL, you can maximize the performance of your code while focusing on the work that your program is designed to accomplish.

Starting with .NET Framework 4, the TPL is the preferred way to write multithreaded and parallel code. However, not all code is suitable for parallelization. For example, if a loop performs only a small amount of work on each iteration, or it doesn't run for many iterations, then the overhead of parallelization can cause the code to run more slowly. Furthermore, parallelization like any multithreaded code adds complexity to your program execution. Although the TPL simplifies multithreaded scenarios, we recommend that you have a basic understanding of threading concepts, for example, locks, deadlocks, and race conditions, so that you can use the TPL effectively.

Related articles

TITLE	DESCRIPTION
Data Parallelism	Describes how to create parallel <code>for</code> and <code>foreach</code> loops (<code>For</code> and <code>For Each</code> in Visual Basic).
Task-based Asynchronous Programming	Describes how to create and run tasks implicitly by using Parallel.Invoke or explicitly by using Task objects directly.
Dataflow	Describes how to use the dataflow components in the TPL Dataflow Library to handle multiple operations that must communicate with one another or to process data as it becomes available.
Potential Pitfalls in Data and Task Parallelism	Describes some common pitfalls and how to avoid them.
Parallel LINQ (PLINQ)	Describes how to achieve data parallelism with LINQ queries.
Parallel Programming	Top level node for .NET parallel programming.

See also

- [Samples for Parallel Programming with the .NET Core & .NET Standard](#)

Data Parallelism (Task Parallel Library)

9/20/2022 • 3 minutes to read • [Edit Online](#)

Data parallelism refers to scenarios in which the same operation is performed concurrently (that is, in parallel) on elements in a source collection or array. In data parallel operations, the source collection is partitioned so that multiple threads can operate on different segments concurrently.

The Task Parallel Library (TPL) supports data parallelism through the [System.Threading.Tasks.Parallel](#) class. This class provides method-based parallel implementations of `for` and `foreach` loops (`For` and `For Each` in Visual Basic). You write the loop logic for a `Parallel.For` or `Parallel.ForEach` loop much as you would write a sequential loop. You do not have to create threads or queue work items. In basic loops, you do not have to take locks. The TPL handles all the low-level work for you. For in-depth information about the use of `Parallel.For` and `Parallel.ForEach`, download the document [Patterns for Parallel Programming: Understanding and Applying Parallel Patterns with the .NET Framework 4](#). The following code example shows a simple `foreach` loop and its parallel equivalent.

NOTE

This documentation uses lambda expressions to define delegates in TPL. If you are not familiar with lambda expressions in C# or Visual Basic, see [Lambda Expressions in PLINQ and TPL](#).

```
// Sequential version
foreach (var item in sourceCollection)
{
    Process(item);
}

// Parallel equivalent
Parallel.ForEach(sourceCollection, item => Process(item));
```

```
' Sequential version
For Each item In sourceCollection
    Process(item)
Next

' Parallel equivalent
Parallel.ForEach(sourceCollection, Sub(item) Process(item))
```

When a parallel loop runs, the TPL partitions the data source so that the loop can operate on multiple parts concurrently. Behind the scenes, the Task Scheduler partitions the task based on system resources and workload. When possible, the scheduler redistributes work among multiple threads and processors if the workload becomes unbalanced.

NOTE

You can also supply your own custom partitioner or scheduler. For more information, see [Custom Partitioners for PLINQ and TPL](#) and [Task Schedulers](#).

Both the `Parallel.For` and `Parallel.ForEach` methods have several overloads that let you stop or break loop execution, monitor the state of the loop on other threads, maintain thread-local state, finalize thread-local

objects, control the degree of concurrency, and so on. The helper types that enable this functionality include [ParallelLoopState](#), [ParallelOptions](#), [ParallelLoopResult](#), [CancellationToken](#), and [CancellationTokenSource](#).

For more information, see [Patterns for Parallel Programming: Understanding and Applying Parallel Patterns with the .NET Framework 4](#).

Data parallelism with declarative, or query-like, syntax is supported by PLINQ. For more information, see [Parallel LINQ \(PLINQ\)](#).

Related Topics

TITLE	DESCRIPTION
How to: Write a Simple Parallel.For Loop	Describes how to write a <code>For</code> loop over any array or indexable <code>IEnumerable<T></code> source collection.
How to: Write a Simple Parallel.ForEach Loop	Describes how to write a <code>ForEach</code> loop over any <code>IEnumerable<T></code> source collection.
How to: Stop or Break from a Parallel.For Loop	Describes how to stop or break from a parallel loop so that all threads are informed of the action.
How to: Write a Parallel.For Loop with Thread-Local Variables	Describes how to write a <code>For</code> loop in which each thread maintains a private variable that is not visible to any other threads, and how to synchronize the results from all threads when the loop completes.
How to: Write a Parallel.ForEach Loop with Partition-Local Variables	Describes how to write a <code>ForEach</code> loop in which each thread maintains a private variable that is not visible to any other threads, and how to synchronize the results from all threads when the loop completes.
How to: Cancel a Parallel.For or ForEach Loop	Describes how to cancel a parallel loop by using a System.Threading.CancellationToken
How to: Speed Up Small Loop Bodies	Describes one way to speed up execution when a loop body is very small.
Task Parallel Library (TPL)	Provides an overview of the Task Parallel Library.
Parallel Programming	Introduces Parallel Programming in the .NET Framework.

See also

- [Parallel Programming](#)

How to: Write a Simple Parallel.For Loop

9/20/2022 • 9 minutes to read • [Edit Online](#)

This topic contains two examples that illustrate the [Parallel.For](#) method. The first uses the [Parallel.For\(Int64, Int64, Action<Int64>\)](#) method overload, and the second uses the [Parallel.For\(Int32, Int32, Action<Int32>\)](#) overload, the two simplest overloads of the [Parallel.For](#) method. You can use these two overloads of the [Parallel.For](#) method when you do not need to cancel the loop, break out of the loop iterations, or maintain any thread-local state.

NOTE

This documentation uses lambda expressions to define delegates in TPL. If you are not familiar with lambda expressions in C# or Visual Basic, see [Lambda Expressions in PLINQ and TPL](#).

The first example calculates the size of files in a single directory. The second computes the product of two matrices.

Directory size example

This example is a simple command-line utility that calculates the total size of files in a directory. It expects a single directory path as an argument, and reports the number and total size of the files in that directory. After verifying that the directory exists, it uses the [Parallel.For](#) method to enumerate the files in the directory and determine their file sizes. Each file size is then added to the `totalSize` variable. Note that the addition is performed by calling the [Interlocked.Add](#) so that the addition is performed as an atomic operation. Otherwise, multiple tasks could try to update the `totalSize` variable simultaneously.

```
using System;
using System.IO;
using System.Threading;
using System.Threading.Tasks;

public class Example
{
    public static void Main(string[] args)
    {
        long totalSize = 0;

        if (args.Length == 0) {
            Console.WriteLine("There are no command line arguments.");
            return;
        }
        if (! Directory.Exists(args[0])) {
            Console.WriteLine("The directory does not exist.");
            return;
        }

        String[] files = Directory.GetFiles(args[0]);
        Parallel.For(0, files.Length,
            index => { FileInfo fi = new FileInfo(files[index]);
                long size = fi.Length;
                Interlocked.Add(ref totalSize, size);
            } );
        Console.WriteLine("Directory '{0}':", args[0]);
        Console.WriteLine("{0:N0} files, {1:N0} bytes", files.Length, totalSize);
    }
}

// The example displays output like the following:
//      Directory 'c:\windows\':
//      32 files, 6,587,222 bytes
```

```

Imports System.IO
Imports System.Threading
Imports System.Threading.Tasks

Module Example
    Public Sub Main()
        Dim totalSize As Long = 0

        Dim args() As String = Environment.GetCommandLineArgs()
        If args.Length = 1 Then
            Console.WriteLine("There are no command line arguments.")
            Return
        End If
        If Not Directory.Exists(args(1))
            Console.WriteLine("The directory does not exist.")
            Return
        End If

        Dim files() As String = Directory.GetFiles(args(1))
        Parallel.For(0, files.Length,
            Sub(index As Integer)
                Dim fi As New FileInfo(files(index))
                Dim size As Long = fi.Length
                Interlocked.Add(totalSize, size)
            End Sub)
        Console.WriteLine("Directory '{0}':", args(1))
        Console.WriteLine("{0:N0} files, {1:N0} bytes", files.Length, totalSize)
    End Sub
End Module
' The example displays output like the following:
'   Directory 'c:\windows\':
'   32 files, 6,587,222 bytes

```

Matrix and stopwatch example

This example uses the [Parallel.ForEach](#) method to compute the product of two matrices. It also shows how to use the [System.Diagnostics.Stopwatch](#) class to compare the performance of a parallel loop with a non-parallel loop.

Note that, because it can generate a large volume of output, the example allows output to be redirected to a file.

```

using System;
using System.Diagnostics;
using System.Runtime.InteropServices;
using System.Threading.Tasks;

class MultiplyMatrices
{
    #region Sequential_Loop
    static void MultiplyMatricesSequential(double[,] matA, double[,] matB,
                                            double[,] result)
    {
        int matACols = matA.GetLength(1);
        int matBCols = matB.GetLength(1);
        int matARows = matA.GetLength(0);

        for (int i = 0; i < matARows; i++)
        {
            for (int j = 0; j < matBCols; j++)
            {
                double temp = 0;
                for (int k = 0; k < matACols; k++)
                {
                    temp += matA[i, k] * matB[k, j];
                }
                result[i, j] += temp;
            }
        }
    }
}

```

```

        }
    }
}

#endregion

#region Parallel_Loop
static void MultiplyMatricesParallel(double[,] matA, double[,] matB, double[,] result)
{
    int matACols = matA.GetLength(1);
    int matBCols = matB.GetLength(1);
    int matARows = matA.GetLength(0);

    // A basic matrix multiplication.
    // Parallelize the outer loop to partition the source array by rows.
    Parallel.For(0, matARows, i =>
    {
        for (int j = 0; j < matBCols; j++)
        {
            double temp = 0;
            for (int k = 0; k < matACols; k++)
            {
                temp += matA[i, k] * matB[k, j];
            }
            result[i, j] = temp;
        }
    });
}

#endregion

#region Main
static void Main(string[] args)
{
    // Set up matrices. Use small values to better view
    // result matrix. Increase the counts to see greater
    // speedup in the parallel loop vs. the sequential loop.
    int colCount = 180;
    int rowCount = 2000;
    int colCount2 = 270;
    double[,] m1 = InitializeMatrix(rowCount, colCount);
    double[,] m2 = InitializeMatrix(colCount, colCount2);
    double[,] result = new double[rowCount, colCount2];

    // First do the sequential version.
    Console.Error.WriteLine("Executing sequential loop...");
    Stopwatch stopwatch = new Stopwatch();
    stopwatch.Start();

    MultiplyMatricesSequential(m1, m2, result);
    stopwatch.Stop();
    Console.Error.WriteLine("Sequential loop time in milliseconds: {0}",
                           stopwatch.ElapsedMilliseconds);

    // For the skeptics.
    OfferToPrint(rowCount, colCount2, result);

    // Reset timer and results matrix.
    stopwatch.Reset();
    result = new double[rowCount, colCount2];

    // Do the parallel loop.
    Console.Error.WriteLine("Executing parallel loop...");
    stopwatch.Start();
    MultiplyMatricesParallel(m1, m2, result);
    stopwatch.Stop();
    Console.Error.WriteLine("Parallel loop time in milliseconds: {0}",
                           stopwatch.ElapsedMilliseconds);
    OfferToPrint(rowCount, colCount2, result);

    // Keep the console window open in debug mode.
}

```

```

        Console.Error.WriteLine("Press any key to exit.");
        Console.ReadKey();
    }
#endregion

#region Helper_Methods
static double[,] InitializeMatrix(int rows, int cols)
{
    double[,] matrix = new double[rows, cols];

    Random r = new Random();
    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < cols; j++)
        {
            matrix[i, j] = r.Next(100);
        }
    }
    return matrix;
}

private static void OfferToPrint(int rowCount, int colCount, double[,] matrix)
{
    Console.Error.Write("Computation complete. Print results (y/n)? ");
    char c = Console.ReadKey(true).KeyChar;
    Console.Error.WriteLine(c);
    if (Char.ToUpperInvariant(c) == 'Y')
    {
        if (!Console.IsOutputRedirected &&
            RuntimeInformation.IsOSPlatform(OSPlatform.Windows))
        {
            Console.WindowWidth = 180;
        }

        Console.WriteLine();
        for (int x = 0; x < rowCount; x++)
        {
            Console.WriteLine("ROW {0}: ", x);
            for (int y = 0; y < colCount; y++)
            {
                Console.Write("{0:#.##} ", matrix[x, y]);
            }
            Console.WriteLine();
        }
    }
}
#endregion
}

```

```

Imports System.Diagnostics
Imports System.Runtime.InteropServices
Imports System.Threading.Tasks

Module MultiplyMatrices
#Region "Sequential_Loop"
    Sub MultiplyMatricesSequential(ByVal matA As Double(), ByVal matB As Double(), ByVal result As Double())
        Dim matACols As Integer = matA.GetLength(1)
        Dim matBCols As Integer = matB.GetLength(1)
        Dim matARows As Integer = matA.GetLength(0)

        For i As Integer = 0 To matARows - 1
            For j As Integer = 0 To matBCols - 1
                Dim temp As Double = 0
                For k As Integer = 0 To matACols - 1
                    temp += matA(i, k) * matB(k, j)
                Next
            
```

```

        result(i, j) += temp
    Next
Next
End Sub
#End Region

#Region "Parallel_Loop"
Private Sub MultiplyMatricesParallel(ByVal matA As Double(,), ByVal matB As Double(,), ByVal result As Double(,))
    Dim matACols As Integer = matA.GetLength(1)
    Dim matBCols As Integer = matB.GetLength(1)
    Dim matARows As Integer = matA.GetLength(0)

    ' A basic matrix multiplication.
    ' Parallelize the outer loop to partition the source array by rows.
    Parallel.For(0, matARows, Sub(i)
        For j As Integer = 0 To matBCols - 1
            Dim temp As Double = 0
            For k As Integer = 0 To matACols - 1
                temp += matA(i, k) * matB(k, j)
            Next
            result(i, j) += temp
        Next
    End Sub)
End Sub
#End Region

#Region "Main"
Sub Main(ByVal args As String())
    ' Set up matrices. Use small values to better view
    ' result matrix. Increase the counts to see greater
    ' speedup in the parallel loop vs. the sequential loop.
    Dim colCount As Integer = 180
    Dim rowCount As Integer = 2000
    Dim colCount2 As Integer = 270
    Dim m1 As Double(,) = InitializeMatrix(rowCount, colCount)
    Dim m2 As Double(,) = InitializeMatrix(colCount, colCount2)
    Dim result As Double(,) = New Double(rowCount - 1, colCount2 - 1) {}

    ' First do the sequential version.
    Console.Error.WriteLine("Executing sequential loop...")
    Dim stopwatch As New Stopwatch()
    stopwatch.Start()

    MultiplyMatricesSequential(m1, m2, result)
    stopwatch.[Stop]()
    Console.Error.WriteLine("Sequential loop time in milliseconds: {0}", stopwatch.ElapsedMilliseconds)

    ' For the skeptics.
    OfferToPrint(rowCount, colCount2, result)

    ' Reset timer and results matrix.
    stopwatch.Reset()
    result = New Double(rowCount - 1, colCount2 - 1) {}

    ' Do the parallel loop.
    Console.Error.WriteLine("Executing parallel loop...")
    stopwatch.Start()
    MultiplyMatricesParallel(m1, m2, result)
    stopwatch.[Stop]()
    Console.Error.WriteLine("Parallel loop time in milliseconds: {0}", stopwatch.ElapsedMilliseconds)
    OfferToPrint(rowCount, colCount2, result)

    ' Keep the console window open in debug mode.
    Console.Error.WriteLine("Press any key to exit.")
    Console.ReadKey()
End Sub
#End Region

```

```

#Region "Helper_Methods"
    Function InitializeMatrix(ByVal rows As Integer, ByVal cols As Integer) As Double()
        Dim matrix As Double(,) = New Double(rows - 1, cols - 1) {}

        Dim r As New Random()
        For i As Integer = 0 To rows - 1
            For j As Integer = 0 To cols - 1
                matrix(i, j) = r.[Next](100)
            Next
        Next
        Return matrix
    End Function

    Sub OfferToPrint(ByVal rowCount As Integer, ByVal colCount As Integer, ByVal matrix As Double())
        Console.Error.WriteLine("Computation complete. Display results (y/n)? ")
        Dim c As Char = Console.ReadKey(True).KeyChar
        Console.Error.WriteLine(c)
        If Char.ToUpperInvariant(c) = "Y"c Then
            If Not Console.IsOutputRedirected AndAlso
                RuntimeInformation.OSPlatform = OSPlatform.Windows Then Console.WindowWidth = 168

            Console.WriteLine()
            For x As Integer = 0 To rowCount - 1
                Console.WriteLine("ROW {0}: ", x)
                For y As Integer = 0 To colCount - 1
                    Console.Write("{0:#.##} ", matrix(x, y))
                Next
                Console.WriteLine()
            Next
            End If
        End Sub
    #End Region
End Module

```

When parallelizing any code, including loops, one important goal is to utilize the processors as much as possible without over parallelizing to the point where the overhead for parallel processing negates any performance benefits. In this particular example, only the outer loop is parallelized because there is not very much work performed in the inner loop. The combination of a small amount of work and undesirable cache effects can result in performance degradation in nested parallel loops. Therefore, parallelizing the outer loop only is the best way to maximize the benefits of concurrency on most systems.

The Delegate

The third parameter of this overload of `For` is a delegate of type `Action<int>` in C# or `Action(Of Integer)` in Visual Basic. An `Action` delegate, whether it has zero, one or sixteen type parameters, always returns void. In Visual Basic, the behavior of an `Action` is defined with a `Sub`. The example uses a lambda expression to create the delegate, but you can create the delegate in other ways as well. For more information, see [Lambda Expressions in PLINQ and TPL](#).

The Iteration Value

The delegate takes a single input parameter whose value is the current iteration. This iteration value is supplied by the runtime and its starting value is the index of the first element on the segment (partition) of the source that is being processed on the current thread.

If you require more control over the concurrency level, use one of the overloads that takes a `System.Threading.Tasks.ParallelOptions` input parameter, such as: `Parallel.For(Int32, Int32, ParallelOptions, Action<Int32, ParallelLoopState>)`.

Return Value and Exception Handling

[For](#) returns use a [System.Threading.Tasks.ParallelLoopResult](#) object when all threads have completed. This return value is useful when you are stopping or breaking loop iteration manually, because the [ParallelLoopResult](#) stores information such as the last iteration that ran to completion. If one or more exceptions occur on one of the threads, a [System.AggregateException](#) will be thrown.

In the code in this example, the return value of [For](#) is not used.

Analysis and Performance

You can use the Performance Wizard to view CPU usage on your computer. As an experiment, increase the number of columns and rows in the matrices. The larger the matrices, the greater the performance difference between the parallel and sequential versions of the computation. When the matrix is small, the sequential version will run faster because of the overhead in setting up the parallel loop.

Synchronous calls to shared resources, like the Console or the File System, will significantly degrade the performance of a parallel loop. When measuring performance, try to avoid calls such as [Console.WriteLine](#) within the loop.

Compile the Code

Copy and paste this code into a Visual Studio project.

See also

- [For](#)
- [ForEach](#)
- [Data Parallelism](#)
- [Parallel Programming](#)

How to: Write a simple Parallel.ForEach loop

9/20/2022 • 3 minutes to read • [Edit Online](#)

This article shows how to use a [Parallel.ForEach](#) loop to enable data parallelism over any [System.Collections.IEnumerable](#) or [System.Collections.Generic.IEnumerable<T>](#) data source.

NOTE

This documentation uses lambda expressions to define delegates in PLINQ. If you aren't familiar with lambda expressions in C# or Visual Basic, see [Lambda expressions in PLINQ and TPL](#).

Example

This example demonstrates [Parallel.ForEach](#) for CPU-intensive operations. When you run the example, it randomly generates 2 million numbers and tries to filter to prime numbers. The first case iterates over the collection via a `for` loop. The second case iterates over the collection via [Parallel.ForEach](#). The resulting time taken by each iteration is displayed when the application is finished.

```
using System;
using System.Collections.Concurrent;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Threading.Tasks;

namespace ParallelExample
{
    class Program
    {
        static void Main()
        {
            // 2 million
            var limit = 2_000_000;
            var numbers = Enumerable.Range(0, limit).ToList();

            var watch = Stopwatch.StartNew();
            var primeNumbersFromForeach = GetPrimeList(numbers);
            watch.Stop();

            var watchForParallel = Stopwatch.StartNew();
            var primeNumbersFromParallelForeach = GetPrimeListWithParallel(numbers);
            watchForParallel.Stop();

            Console.WriteLine($"Classical foreach loop | Total prime numbers : {primeNumbersFromForeach.Count} | Time Taken : {watch.ElapsedMilliseconds} ms.");
            Console.WriteLine($"Parallel.ForEach loop | Total prime numbers : {primeNumbersFromParallelForeach.Count} | Time Taken : {watchForParallel.ElapsedMilliseconds} ms.");

            Console.WriteLine("Press any key to exit.");
            Console.ReadLine();
        }

        /// <summary>
        /// GetPrimeList returns Prime numbers by using sequential ForEach
        /// </summary>
        /// <param name="inputs"></param>
        /// <returns></returns>
        private static IList<int> GetPrimeList(IList<int> numbers) => numbers.Where(IsPrime).ToList();
    }
}
```

```
/// <summary>
/// GetPrimeListWithParallel returns Prime numbers by using Parallel.ForEach
/// </summary>
/// <param name="numbers"></param>
/// <returns></returns>
private static IList<int> GetPrimeListWithParallel(IList<int> numbers)
{
    var primeNumbers = new ConcurrentBag<int>();

    Parallel.ForEach(numbers, number =>
    {
        if (IsPrime(number))
        {
            primeNumbers.Add(number);
        }
    });
}

return primeNumbers.ToList();
}

/// <summary>
/// IsPrime returns true if number is Prime, else false.(https://en.wikipedia.org/wiki/Prime\_number)
/// </summary>
/// <param name="number"></param>
/// <returns></returns>
private static bool IsPrime(int number)
{
    if (number < 2)
    {
        return false;
    }

    for (var divisor = 2; divisor <= Math.Sqrt(number); divisor++)
    {
        if (number % divisor == 0)
        {
            return false;
        }
    }
    return true;
}
}
```

```

Imports System.Collections.Concurrent

Namespace ParallelExample
    Class Program
        Shared Sub Main()
            ' 2 million
            Dim limit = 2_000_000
            Dim numbers = Enumerable.Range(0, limit).ToList()

            Dim watch = Stopwatch.StartNew()
            Dim primeNumbersFromForeach = GetPrimeList(numbers)
            watch.Stop()

            Dim watchForParallel = Stopwatch.StartNew()
            Dim primeNumbersFromParallelForeach = GetPrimeListWithParallel(numbers)
            watchForParallel.Stop()

            Console.WriteLine($"Classical foreach loop | Total prime numbers :
{primeNumbersFromForeach.Count} | Time Taken : {watch.ElapsedMilliseconds} ms.")
            Console.WriteLine($"Parallel.ForEach loop | Total prime numbers :
{primeNumbersFromParallelForeach.Count} | Time Taken : {watchForParallel.ElapsedMilliseconds} ms.")

            Console.WriteLine("Press any key to exit.")
            Console.ReadLine()
        End Sub

        ' GetPrimeList returns Prime numbers by using sequential ForEach
        Private Shared Function GetPrimeList(numbers As IList(Of Integer)) As IList(Of Integer)
            Return numbers.Where(AddressOf IsPrime).ToList()
        End Function

        ' GetPrimeListWithParallel returns Prime numbers by using Parallel.ForEach
        Private Shared Function GetPrimeListWithParallel(numbers As IList(Of Integer)) As IList(Of Integer)
            Dim primeNumbers = New ConcurrentBag(Of Integer)()
            Parallel.ForEach(numbers, Sub(number)

                If IsPrime(number) Then
                    primeNumbers.Add(number)
                End If
            End Sub)
            Return primeNumbers.ToList()
        End Function

        ' IsPrime returns true if number is Prime, else false.(https://en.wikipedia.org/wiki/Prime\_number)
        Private Shared Function IsPrime(number As Integer) As Boolean
            If number < 2 Then
                Return False
            End If

            For divisor = 2 To Math.Sqrt(number)

                If number Mod divisor = 0 Then
                    Return False
                End If
            Next

            Return True
        End Function
    End Class
End Namespace

```

A [Parallel.ForEach](#) loop works like a [Parallel.ForEach](#) loop. The loop partitions the source collection and schedules the work on multiple threads based on the system environment. The more processors on the system, the faster the parallel method runs. For some source collections, a sequential loop might be faster, depending on the size of the source and the kind of work the loop performs. For more information about performance, see [Potential](#)

pitfalls in data and task parallelism.

For more information about parallel loops, see [How to: Write a simple Parallel.ForEach loop](#).

To use the `Parallel.ForEach` loop with a non-generic collection, you can use the `Enumerable.Cast` extension method to convert the collection to a generic collection, as shown in the following example:

```
Parallel.ForEach(nonGenericCollection.Cast<object>(),
    currentElement =>
{
});
```

```
Parallel.ForEach(nonGenericCollection.Cast(Of Object), _
    Sub(currentElement)
        ' ... work with currentElement
    End Sub)
```

You can also use Parallel LINQ (PLINQ) to parallelize the processing of `IEnumerable<T>` data sources. PLINQ enables you to use declarative query syntax to express the loop behavior. For more information, see [Parallel LINQ \(PLINQ\)](#).

Compile and run the code

You can compile the code as a console application for .NET Framework or as a console application for .NET Core.

In Visual Studio, there are Visual Basic and C# console application templates for Windows Desktop and .NET Core.

From the command line, you can use the .NET CLI commands (for example, `dotnet new console` or `dotnet new console -lang vb`) or create the file and use the command-line compiler for a .NET Framework application.

To run a .NET Core console application from the command line, use `dotnet run` from the folder that contains your application.

To run your console application from Visual Studio, press F5.

See also

- [Data parallelism](#)
- [Parallel programming](#)
- [Parallel LINQ \(PLINQ\)](#)

How to: Write a Parallel.For Loop with Thread-Local Variables

9/20/2022 • 3 minutes to read • [Edit Online](#)

This example shows how to use thread-local variables to store and retrieve state in each separate task that is created by a `For` loop. By using thread-local data, you can avoid the overhead of synchronizing a large number of accesses to shared state. Instead of writing to a shared resource on each iteration, you compute and store the value until all iterations for the task are complete. You can then write the final result once to the shared resource, or pass it to another method.

Example

The following example calls the `For<TLocal>(Int32, Int32, Func<TLocal>, Func<Int32,ParallelLoopState,TLocal,TLocal>, Action<TLocal>)` method to calculate the sum of the values in an array that contains one million elements. The value of each element is equal to its index.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading;
using System.Threading.Tasks;

class Test
{
    static void Main()
    {
        int[] nums = Enumerable.Range(0, 1000000).ToArray();
        long total = 0;

        // Use type parameter to make subtotal a long, not an int
        Parallel.For<long>(0, nums.Length, () => 0, (j, loop, subtotal) =>
        {
            subtotal += nums[j];
            return subtotal;
        },
        (x) => Interlocked.Add(ref total, x)
    );

        Console.WriteLine("The total is {0:N0}", total);
        Console.WriteLine("Press any key to exit");
        Console.ReadKey();
    }
}
```

```

'How to: Write a Parallel.For Loop That Has Thread-Local Variables

Imports System.Threading
Imports System.Threading.Tasks

Module ForWithThreadLocal

    Sub Main()
        Dim nums As Integer() = Enumerable.Range(0, 1000000).ToArray()
        Dim total As Long = 0

        ' Use type parameter to make subtotal a Long type. Function will overflow otherwise.
        Parallel.For(Of Long)(0, nums.Length, Function() 0, Function(j, [loop], subtotal)
            subtotal += nums(j)
            Return subtotal
        End Function, Function(x) Interlocked.Add(total,
x))

        Console.WriteLine("The total is {0:N0}", total)
        Console.WriteLine("Press any key to exit")
        Console.ReadKey()
    End Sub

End Module

```

The first two parameters of every `For` method specify the beginning and ending iteration values. In this overload of the method, the third parameter is where you initialize your local state. In this context, local state means a variable whose lifetime extends from just before the first iteration of the loop on the current thread, to just after the last iteration.

The type of the third parameter is a `Func<TResult>` where `TResult` is the type of the variable that will store the thread-local state. Its type is defined by the generic type argument supplied when calling the generic `For<TLocal>(Int32, Int32, Func<TLocal>, Func<Int32,ParallelLoopState,TLocal,TLocal>, Action<TLocal>)` method, which in this case is `Int64`. The type argument tells the compiler the type of the temporary variable that will be used to store the thread-local state. In this example, the expression `() => 0` (or `Function() 0` in Visual Basic) initializes the thread-local variable to zero. If the generic type argument is a reference type or user-defined value type, the expression would look like this:

```
(() => new MyClass())
```

```
Function() new MyClass()
```

The fourth parameter defines the loop logic. It must be a delegate or lambda expression whose signature is `Func<int, ParallelLoopState, long, long>` in C# or `Func(Of Integer, ParallelLoopState, Long, Long)` in Visual Basic. The first parameter is the value of the loop counter for that iteration of the loop. The second is a `ParallelLoopState` object that can be used to break out of the loop; this object is provided by the `Parallel` class to each occurrence of the loop. The third parameter is the thread-local variable. The last parameter is the return type. In this case, the type is `Int64` because that is the type we specified in the `For` type argument. That variable is named `subtotal` and is returned by the lambda expression. The return value is used to initialize `subtotal` on each subsequent iteration of the loop. You can also think of this last parameter as a value that is passed to each iteration, and then passed to the `localFinally` delegate when the last iteration is complete.

The fifth parameter defines the method that is called once, after all the iterations on a particular thread have completed. The type of the input argument again corresponds to the type argument of the `For<TLocal>(Int32, Int32, Func<TLocal>, Func<Int32,ParallelLoopState,TLocal,TLocal>, Action<TLocal>)` method and the type returned by the body lambda expression. In this example, the value is added to a variable at class scope in a

thread safe way by calling the [Interlocked.Add](#) method. By using a thread-local variable, we have avoided writing to this class variable on every iteration of the loop.

For more information about how to use lambda expressions, see [Lambda Expressions in PLINQ and TPL](#).

See also

- [Data Parallelism](#)
- [Parallel Programming](#)
- [Task Parallel Library \(TPL\)](#)
- [Lambda Expressions in PLINQ and TPL](#)

How to: Write a Parallel.ForEach loop with partition-local variables

9/20/2022 • 3 minutes to read • [Edit Online](#)

The following example shows how to write a [ForEach](#) method that uses partition-local variables. When a [ForEach](#) loop executes, it divides its source collection into multiple partitions. Each partition has its own copy of the partition-local variable. A partition-local variable is similar to a [thread-local variable](#), except that multiple partitions can run on a single thread.

The code and parameters in this example closely resemble the corresponding [For](#) method. For more information, see [How to: Write a Parallel.For Loop with Thread-Local Variables](#).

To use a partition-local variable in a [ForEach](#) loop, you must call one of the method overloads that takes two type parameters. The first type parameter, `TSource`, specifies the type of the source element, and the second type parameter, `TLocal`, specifies the type of the partition-local variable.

Example

The following example calls the `Parallel.ForEach<TSource,TLocal>(IEnumerable<TSource>, Func<TLocal>, Func<TSource,ParallelLoopState,TLocal,TLocal>, Action<TLocal>)` overload to compute the sum of an array of one million elements. This overload has four parameters:

- `source`, which is the data source. It must implement `IEnumerable<T>`. The data source in our example is the one million member `IEnumerable<Int32>` object returned by the [Enumerable.Range](#) method.
- `localInit`, or the function that initializes the partition-local variable. This function is called once for each partition in which the [Parallel.ForEach](#) operation executes. Our example initializes the partition-local variable to zero.
- `body`, a `Func<T1,T2,T3,TResult>` that is invoked by the parallel loop on each iteration of the loop. Its signature is `Func<TSource, ParallelLoopState, TLocal, TLocal>`. You supply the code for the delegate, and the loop passes in the input parameters, which are:
 - The current element of the `IEnumerable<T>`.
 - A `ParallelLoopState` variable that you can use in your delegate's code to examine the state of the loop.
 - The partition-local variable.

Your delegate returns the partition-local variable, which is then passed to the next iteration of the loop that executes in that particular partition. Each loop partition maintains a separate instance of this variable.

In the example, the delegate adds the value of each integer to the partition-local variable, which maintains a running total of the values of the integer elements in that partition.

- `localFinally`, an `Action<TLocal>` delegate that the [Parallel.ForEach](#) invokes when the looping operations in each partition have completed. The [Parallel.ForEach](#) method passes your `Action<TLocal>` delegate the final value of the partition-local variable for this loop partition, and you provide the code that performs the required action for combining the result from this partition with the results from the other partitions. This delegate can be invoked concurrently by multiple tasks. Because of this, the example uses the `Interlocked.Add(Int32, Int32)` method to synchronize access to the `total` variable. Because the delegate

type is an `Action<T>`, there is no return value.

```
using System;
using System.Linq;
using System.Threading;
using System.Threading.Tasks;

class Test
{
    static void Main()
    {
        int[] nums = Enumerable.Range(0, 1000000).ToArray();
        long total = 0;

        // First type parameter is the type of the source elements
        // Second type parameter is the type of the thread-local variable (partition subtotal)
        Parallel.ForEach<int, long>(nums, // source collection
            () => 0, // method to initialize the local variable
            (j, loop, subtotal) => // method invoked by the loop on each iteration
            {
                subtotal += j; //modify local variable
                return subtotal; // value to be passed to next iteration
            },
            // Method to be executed when each partition has completed.
            // finalResult is the final value of subtotal for a particular partition.
            (finalResult) => Interlocked.Add(ref total, finalResult)
        );

        Console.WriteLine("The total from Parallel.ForEach is {0:N0}", total);
    }
}

// The example displays the following output:
//      The total from Parallel.ForEach is 499,999,500,000
```

```
' How to: Write a Parallel.ForEach Loop That Has Thread-Local Variables

Imports System.Threading
Imports System.Threading.Tasks

Module ForEachThreadLocal
    Sub Main()

        Dim nums() As Integer = Enumerable.Range(0, 1000000).ToArray()
        Dim total As Long = 0

        ' First type paramemter is the type of the source elements
        ' Second type parameter is the type of the thread-local variable (partition subtotal)
        Parallel.ForEach(Of Integer, Long)(nums, Function() 0,
            Function(elem, loopState, subtotal)
                subtotal += elem
                Return subtotal
            End Function,
            Sub(finalResult)
                Interlocked.Add(total, finalResult)
            End Sub)

        Console.WriteLine("The result of Parallel.ForEach is {0:N0}", total)
    End Sub
End Module

' The example displays the following output:
'      The result of Parallel.ForEach is 499,999,500,000
```

See also

- Data Parallelism
- How to: Write a Parallel.For Loop with Thread-Local Variables
- Lambda Expressions in PLINQ and TPL

How to: Cancel a Parallel.For or ForEach Loop

9/20/2022 • 2 minutes to read • [Edit Online](#)

The [Parallel.For](#) and [Parallel.ForEach](#) methods support cancellation through the use of cancellation tokens. For more information about cancellation in general, see [Cancellation](#). In a parallel loop, you supply the [CancellationToken](#) to the method in the [ParallelOptions](#) parameter and then enclose the parallel call in a try-catch block.

Example

The following example shows how to cancel a call to [Parallel.ForEach](#). You can apply the same approach to a [Parallel.For](#) call.

```

namespace CancelParallelLoops
{
    using System;
    using System.Linq;
    using System.Threading;
    using System.Threading.Tasks;

    class Program
    {
        static void Main()
        {
            int[] nums = Enumerable.Range(0, 10000000).ToArray();
            CancellationTokenSource cts = new CancellationTokenSource();

            // Use ParallelOptions instance to store the CancellationToken
            ParallelOptions po = new ParallelOptions();
            po.CancellationToken = cts.Token;
            po.MaxDegreeOfParallelism = System.Environment.ProcessorCount;
            Console.WriteLine("Press any key to start. Press 'c' to cancel.");
            Console.ReadKey();

            // Run a task so that we can cancel from another thread.
            Task.Factory.StartNew(() =>
            {
                if (Console.ReadKey().KeyChar == 'c')
                    cts.Cancel();
                Console.WriteLine("press any key to exit");
            });

            try
            {
                Parallel.ForEach(nums, po, (num) =>
                {
                    double d = Math.Sqrt(num);
                    Console.WriteLine("{0} on {1}", d, Thread.CurrentThread.ManagedThreadId);
                });
            }
            catch (OperationCanceledException e)
            {
                Console.WriteLine(e.Message);
            }
            finally
            {
                cts.Dispose();
            }

            Console.ReadKey();
        }
    }
}

```

```

' How to: Cancel a Parallel.For or ForEach Loop

Imports System.Threading
Imports System.Threading.Tasks

Module CancelParallelLoops
    Sub Main()
        Dim nums() As Integer = Enumerable.Range(0, 10000000).ToArray()
        Dim cts As New CancellationTokenSource

        ' Use ParallelOptions instance to store the CancellationToken
        Dim po As New ParallelOptions
        po.CancellationToken = cts.Token
        po.MaxDegreeOfParallelism = System.Environment.ProcessorCount
        Console.WriteLine("Press any key to start. Press 'c' to cancel.")
        Console.ReadKey()

        ' Run a task so that we can cancel from another thread.
        Dim t As Task = Task.Factory.StartNew(Sub()
                                                If Console.ReadKey().KeyChar = "c"c Then
                                                    cts.Cancel()
                                                End If
                                                Console.WriteLine(vbCrLf & "Press any key to exit.")
                                            End Sub)

        Try

            ' The error "Exception is unhandled by user code" will appear if "Just My Code"
            ' is enabled. This error is benign. You can press F5 to continue, or disable Just My Code.
            Parallel.ForEach(nums, po, Sub(num)
                Dim d As Double = Math.Sqrt(num)
                Console.CursorLeft = 0
                Console.WriteLine("{0:##.##} on {1}", d,
Thread.CurrentThread.ManagedThreadId)
            End Sub)

            Catch e As OperationCanceledException
                Console.WriteLine(e.Message)
            Finally
                cts.Dispose()
            End Try

            Console.ReadKey()

        End Sub
    End Module

```

If the token that signals the cancellation is the same token that is specified in the [ParallelOptions](#) instance, then the parallel loop will throw a single [OperationCanceledException](#) on cancellation. If some other token causes cancellation, the loop will throw an [AggregateException](#) with an [OperationCanceledException](#) as an [InnerException](#).

See also

- [Data parallelism](#)
- [Lambda expressions in PLINQ and TPL](#)

How to: Handle Exceptions in Parallel Loops

9/20/2022 • 3 minutes to read • [Edit Online](#)

The `Parallel.For` and `Parallel.ForEach` overloads do not have any special mechanism to handle exceptions that might be thrown. In this respect, they resemble regular `for` and `foreach` loops (`For` and `For Each` in Visual Basic); an unhandled exception causes the loop to terminate as soon as all currently running iterations finish.

When you add your own exception-handling logic to parallel loops, handle the case in which similar exceptions might be thrown on multiple threads concurrently, and the case in which an exception thrown on one thread causes another exception to be thrown on another thread. You can handle both cases by wrapping all exceptions from the loop in a `System.AggregateException`. The following example shows one possible approach.

NOTE

When "Just My Code" is enabled, Visual Studio in some cases will break on the line that throws the exception and display an error message that says "exception not handled by user code." This error is benign. You can press F5 to continue from it, and see the exception-handling behavior that is demonstrated in the example below. To prevent Visual Studio from breaking on the first error, just uncheck the "Just My Code" checkbox under **Tools, Options, Debugging, General**.

Example

In this example, all exceptions are caught and then wrapped in an `System.AggregateException` which is thrown. The caller can decide which exceptions to handle.

```

public static partial class Program
{
    public static void ExceptionTwo()
    {
        // Create some random data to process in parallel.
        // There is a good probability this data will cause some exceptions to be thrown.
        byte[] data = new byte[5_000];
        Random r = Random.Shared;
        r.NextBytes(data);

        try
        {
            ProcessDataInParallel(data);
        }
        catch (AggregateException ae)
        {
            var ignoredExceptions = new List<Exception>();
            // This is where you can choose which exceptions to handle.
            foreach (var ex in ae.Flatten().InnerExceptions)
            {
                if (ex is ArgumentException) Console.WriteLine(ex.Message);
                else ignoredExceptions.Add(ex);
            }
            if (ignoredExceptions.Count > 0)
            {
                throw new AggregateException(ignoredExceptions);
            }
        }
    }

    Console.WriteLine("Press any key to exit.");
    Console.ReadKey();
}

private static void ProcessDataInParallel(byte[] data)
{
    // Use ConcurrentQueue to enable safe enqueueing from multiple threads.
    var exceptions = new ConcurrentQueue<Exception>();

    // Execute the complete loop and capture all exceptions.
    Parallel.ForEach(data, d =>
    {
        try
        {
            // Cause a few exceptions, but not too many.
            if (d < 3) throw new ArgumentException($"Value is {d}. Value must be greater than or equal
to 3.");
            else Console.Write(d + " ");
        }
        // Store the exception and continue with the loop.
        catch (Exception e)
        {
            exceptions.Enqueue(e);
        }
    });
    Console.WriteLine();

    // Throw the exceptions here after the loop completes.
    if (!exceptions.IsEmpty)
    {
        throw new AggregateException(exceptions);
    }
}
}

```

```

' How to: Handle Exceptions in Parallel Loops

Imports System.Collections.Concurrent
Imports System.Collections.Generic
Imports System.Threading.Tasks

Module ExceptionsInLoops

Sub Main()

    ' Create some random data to process in parallel.
    ' There is a good probability this data will cause some exceptions to be thrown.
    Dim data(1000) As Byte
    Dim r As New Random()
    r.NextBytes(data)

    Try
        ProcessDataInParallel(data)
    Catch ae As AggregateException
        Dim ignoredExceptions As New List(Of Exception)
        ' This is where you can choose which exceptions to handle.
        For Each ex As Exception In ae.Flatten().InnerExceptions
            If (TypeOf (ex) Is ArgumentException) Then
                Console.WriteLine(ex.Message)
            Else
                ignoredExceptions.Add(ex)
            End If
        Next
        If ignoredExceptions.Count > 0 Then
            Throw New AggregateException(ignoredExceptions)
        End If
    End Try
    Console.WriteLine("Press any key to exit.")
    Console.ReadKey()
End Sub

Sub ProcessDataInParallel(ByVal data As Byte())

    ' Use ConcurrentQueue to enable safe enqueueing from multiple threads.
    Dim exceptions As New ConcurrentQueue(Of Exception)

    ' Execute the complete loop and capture all exceptions.
    Parallel.ForEach(Of Byte)(data, Sub(d)
        Try
            ' Cause a few exceptions, but not too many.
            If d < 3 Then
                Throw New ArgumentException($"Value is {d}. Value must
be greater than or equal to 3")
            Else
                Console.Write(d & " ")
            End If
        Catch ex As Exception
            ' Store the exception and continue with the loop.
            exceptions.Enqueue(ex)
        End Try
    End Sub)

    Console.WriteLine()
    ' Throw the exceptions here after the loop completes.
    If exceptions.Count > 0 Then
        Throw New AggregateException(exceptions)
    End If
End Sub
End Module

```

See also

- Data Parallelism
- Lambda Expressions in PLINQ and TPL

How to: Speed Up Small Loop Bodies

9/20/2022 • 2 minutes to read • [Edit Online](#)

When a `Parallel.For` loop has a small body, it might perform more slowly than the equivalent sequential loop, such as the `for` loop in C# and the `For` loop in Visual Basic. Slower performance is caused by the overhead involved in partitioning the data and the cost of invoking a delegate on each loop iteration. To address such scenarios, the `Partitioner` class provides the `Partitioner.Create` method, which enables you to provide a sequential loop for the delegate body, so that the delegate is invoked only once per partition, instead of once per iteration. For more information, see [Custom Partitioners for PLINQ and TPL](#).

Example

```
using System;
using System.Collections.Concurrent;
using System.Linq;
using System.Threading;
using System.Threading.Tasks;

class Program
{
    static void Main()
    {

        // Source must be array or IList.
        var source = Enumerable.Range(0, 100000).ToArray();

        // Partition the entire source array.
        var rangePartitioner = Partitioner.Create(0, source.Length);

        double[] results = new double[source.Length];

        // Loop over the partitions in parallel.
        Parallel.ForEach(rangePartitioner, (range, loopState) =>
        {
            // Loop over each range element without a delegate invocation.
            for (int i = range.Item1; i < range.Item2; i++)
            {
                results[i] = source[i] * Math.PI;
            }
        });

        Console.WriteLine("Operation complete. Print results? y/n");
        char input = Console.ReadKey().KeyChar;
        if (input == 'y' || input == 'Y')
        {
            foreach(double d in results)
            {
                Console.Write("{0} ", d);
            }
        }
    }
}
```

```

Imports System.Threading.Tasks
Imports System.Collections.Concurrent

Module PartitionDemo

Sub Main()
    ' Source must be array or IList.
    Dim source = Enumerable.Range(0, 100000).ToArray()

    ' Partition the entire source array.
    ' Let the partitioner size the ranges.
    Dim rangePartitioner = Partitioner.Create(0, source.Length)

    Dim results(source.Length - 1) As Double

    ' Loop over the partitions in parallel. The Sub is invoked
    ' once per partition.
    Parallel.ForEach(rangePartitioner, Sub(range, loopState)

        ' Loop over each range element without a delegate invocation.
        For i As Integer = range.Item1 To range.Item2 - 1
            results(i) = source(i) * Math.PI
        Next
    End Sub)
    Console.WriteLine("Operation complete. Print results? y/n")
    Dim input As Char = Console.ReadKey().KeyChar
    If input = "y"c Or input = "Y"c Then
        For Each d As Double In results
            Console.Write("{0} ", d)
        Next
    End If

    End Sub
End Module

```

The approach demonstrated in this example is useful when the loop performs a minimal amount of work. As the work becomes more computationally expensive, you will probably get the same or better performance by using a [For](#) or [ForEach](#) loop with the default partitioner.

See also

- [Data Parallelism](#)
- [Custom Partitioners for PLINQ and TPL](#)
- [Iterators \(C#\)](#)
- [Iterators \(Visual Basic\)](#)
- [Lambda Expressions in PLINQ and TPL](#)

How to: Iterate File Directories with the Parallel Class

9/20/2022 • 6 minutes to read • [Edit Online](#)

In many cases, file iteration is an operation that can be easily parallelized. The topic [How to: Iterate File Directories with PLINQ](#) shows the easiest way to perform this task for many scenarios. However, complications can arise when your code has to deal with the many types of exceptions that can arise when accessing the file system. The following example shows one approach to the problem. It uses a stack-based iteration to traverse all files and folders under a specified directory, and it enables your code to catch and handle various exceptions. Of course, the way that you handle the exceptions is up to you.

Example

The following example iterates the directories sequentially, but processes the files in parallel. This is probably the best approach when you have a large file-to-directory ratio. It is also possible to parallelize the directory iteration, and access each file sequentially. It is probably not efficient to parallelize both loops unless you are specifically targeting a machine with a large number of processors. However, as in all cases, you should test your application thoroughly to determine the best approach.

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.IO;
using System.Security;
using System.Threading;
using System.Threading.Tasks;

class Program
{
    static void Main()
    {
        try {
            TraverseTreeParallelForEach(@"C:\Program Files", (f) =>
            {
                // Exceptions are no-ops.
                try {
                    // Do nothing with the data except read it.
                    byte[] data = File.ReadAllBytes(f);
                }
                catch (FileNotFoundException) {}
                catch (IOException) {}
                catch (UnauthorizedAccessException) {}
                catch (SecurityException) {}
                // Display the filename.
                Console.WriteLine(f);
            });
        }
        catch (ArgumentException) {
            Console.WriteLine(@"The directory 'C:\Program Files' does not exist.");
        }

        // Keep the console window open.
        Console.ReadKey();
    }

    public static void TraverseTreeParallelForEach(string root, Action<string> action)
    {
```

```

//Count of files traversed and timer for diagnostic output
int fileCount = 0;
var sw = Stopwatch.StartNew();

// Determine whether to parallelize file processing on each folder based on processor count.
int procCount = System.Environment.ProcessorCount;

// Data structure to hold names of subfolders to be examined for files.
Stack<string> dirs = new Stack<string>();

if (!Directory.Exists(root)) {
    throw new ArgumentException(
        "The given root directory doesn't exist.", nameof(root));
}
dirs.Push(root);

while (dirs.Count > 0) {
    string currentDir = dirs.Pop();
    string[] subDirs = {};
    string[] files = {};

    try {
        subDirs = Directory.GetDirectories(currentDir);
    }
    // Thrown if we do not have discovery permission on the directory.
    catch (UnauthorizedAccessException e) {
        Console.WriteLine(e.Message);
        continue;
    }
    // Thrown if another process has deleted the directory after we retrieved its name.
    catch (DirectoryNotFoundException e) {
        Console.WriteLine(e.Message);
        continue;
    }

    try {
        files = Directory.GetFiles(currentDir);
    }
    catch (UnauthorizedAccessException e) {
        Console.WriteLine(e.Message);
        continue;
    }
    catch (DirectoryNotFoundException e) {
        Console.WriteLine(e.Message);
        continue;
    }
    catch (IOException e) {
        Console.WriteLine(e.Message);
        continue;
    }

    // Execute in parallel if there are enough files in the directory.
    // Otherwise, execute sequentially. Files are opened and processed
    // synchronously but this could be modified to perform async I/O.
    try {
        if (files.Length < procCount) {
            foreach (var file in files) {
                action(file);
                fileCount++;
            }
        }
        else {
            Parallel.ForEach(files, () => 0, (file, loopState, localCount) =>
            {
                action(file);
                return (int) ++localCount;
            },
            (c) => {
                Interlocked.Add(ref fileCount, c);
            });
        }
    }
}

```

```

        }

    }

    catch (AggregateException ae) {
        ae.Handle((ex) => {
            if (ex is UnauthorizedAccessException) {
                // Here we just output a message and go on.
                Console.WriteLine(ex.Message);
                return true;
            }
            // Handle other exceptions here if necessary...
        });

        return false;
    }
}

// Push the subdirectories onto the stack for traversal.
// This could also be done before handing the files.
foreach (string str in subDirs)
    dirs.Push(str);
}

// For diagnostic purposes.
Console.WriteLine("Processed {0} files in {1} milliseconds", fileCount, sw.ElapsedMilliseconds);
}
}

```

```

Imports System.Collections.Generic
Imports System.Diagnostics
Imports System.IO
Imports System.Security
Imports System.Threading
Imports System.Threading.Tasks

Module Example
    Sub Main()
        Try
            TraverseTreeParallelForEach("C:\Program Files",
                Sub(f)
                    ' Exceptions are No-ops.
                    Try
                        ' Do nothing with the data except read it.
                        Dim data() As Byte = File.ReadAllBytes(f)
                        ' In the event the file has been deleted.
                    Catch e As FileNotFoundException

                        ' General I/O exception, especially if the file is in use.
                        Catch e As IOException

                            ' Lack of adequate permissions.
                            Catch e As UnauthorizedAccessException

                                ' Lack of adequate permissions.
                                Catch e As SecurityException

                            End Try
                            ' Display the filename.
                            Console.WriteLine(f)
                        End Sub)
                    Catch e As ArgumentException
                        Console.WriteLine("The directory 'C:\Program Files' does not exist.")
                    End Try
                    ' Keep the console window open.
                    Console.ReadKey()
                End Sub

            Public Sub TraverseTreeParallelForEach(ByVal root As String, ByVal action As Action(Of String))
                'Count of files traversed and timer for diagnostic output

```

```

Dim fileCount As Integer = 0
Dim sw As Stopwatch = Stopwatch.StartNew()

' Determine whether to parallelize file processing on each folder based on processor count.
Dim procCount As Integer = System.Environment.ProcessorCount

' Data structure to hold names of subfolders to be examined for files.
Dim dirs As New Stack(Of String)

If Not Directory.Exists(root) Then Throw New ArgumentException(
    "The given root directory doesn't exist.", nameof(root))

dirs.Push(root)

While (dirs.Count > 0)
    Dim currentDir As String = dirs.Pop()
    Dim subDirs() As String = Nothing
    Dim files() As String = Nothing

    Try
        subDirs = Directory.GetDirectories(currentDir)
        ' Thrown if we do not have discovery permission on the directory.
        Catch e As UnauthorizedAccessException
            Console.WriteLine(e.Message)
            Continue While
            ' Thrown if another process has deleted the directory after we retrieved its name.
        Catch e As DirectoryNotFoundException
            Console.WriteLine(e.Message)
            Continue While
    End Try

    Try
        files = Directory.GetFiles(currentDir)
        Catch e As UnauthorizedAccessException
            Console.WriteLine(e.Message)
            Continue While
        Catch e As DirectoryNotFoundException
            Console.WriteLine(e.Message)
            Continue While
        Catch e As IOException
            Console.WriteLine(e.Message)
            Continue While
    End Try

    ' Execute in parallel if there are enough files in the directory.
    ' Otherwise, execute sequentially.Files are opened and processed
    ' synchronously but this could be modified to perform async I/O.
    Try
        If files.Length < procCount Then
            For Each file In files
                action(file)
                fileCount += 1
            Next
        Else
            Parallel.ForEach(files, Function() 0, Function(file, loopState, localCount)
                action(file)
                localCount = localCount + 1
                Return localCount
            End Function,
            Sub(c)
                Interlocked.Add(fileCount, c)
            End Sub)
        End If
        Catch ae As AggregateException
            ae.Handle(Function(ex)
                If TypeOf (ex) Is UnauthorizedAccessException Then
                    ' Here we just output a message and go on.
                End If
            End Sub)
        End Try
    End Try
End Sub

```

```

        Console.WriteLine(ex.Message)
        Return True
    End If
    ' Handle other exceptions here if necessary...

    Return False
End Function)

End Try
' Push the subdirectories onto the stack for traversal.
' This could also be done before handing the files.
For Each str As String In subDirs
    dirs.Push(str)
Next

' For diagnostic purposes.
Console.WriteLine("Processed {0} files in {1} milliseconds", fileCount, sw.ElapsedMilliseconds)
End While
End Sub
End Module

```

In this example, the file I/O is performed synchronously. When dealing with large files or slow network connections, it might be preferable to access the files asynchronously. You can combine asynchronous I/O techniques with parallel iteration. For more information, see [TPL and Traditional .NET Asynchronous Programming](#).

The example uses the local `fileCount` variable to maintain a count of the total number of files processed. Because the variable might be accessed concurrently by multiple tasks, access to it is synchronized by calling the `Interlocked.Add` method.

Note that if an exception is thrown on the main thread, the threads that are started by the `ForEach` method might continue to run. To stop these threads, you can set a Boolean variable in your exception handlers, and check its value on each iteration of the parallel loop. If the value indicates that an exception has been thrown, use the `ParallelLoopState` variable to stop or break from the loop. For more information, see [How to: Stop or Break from a Parallel.ForEach Loop](#).

See also

- [Data Parallelism](#)

Task-based asynchronous programming

9/20/2022 • 35 minutes to read • [Edit Online](#)

The Task Parallel Library (TPL) is based on the concept of a *task*, which represents an asynchronous operation. In some ways, a task resembles a thread or [ThreadPool](#) work item but at a higher level of abstraction. The term *task parallelism* refers to one or more independent tasks running concurrently. Tasks provide two primary benefits:

- More efficient and more scalable use of system resources.

Behind the scenes, tasks are queued to the [ThreadPool](#), which has been enhanced with algorithms that determine and adjust to the number of threads. These algorithms provide load balancing to maximize throughput. This process makes tasks relatively lightweight, and you can create many of them to enable fine-grained parallelism.

- More programmatic control than is possible with a thread or work item.

Tasks and the framework built around them provide a rich set of APIs that support waiting, cancellation, continuations, robust exception handling, detailed status, custom scheduling, and more.

For both reasons, TPL is the preferred API for writing multi-threaded, asynchronous, and parallel code in .NET.

Creating and running tasks implicitly

The [Parallel.Invoke](#) method provides a convenient way to run any number of arbitrary statements concurrently. Just pass in an [Action](#) delegate for each item of work. The easiest way to create these delegates is to use lambda expressions. The lambda expression can either call a named method or provide the code inline. The following example shows a basic [Invoke](#) call that creates and starts two tasks that run concurrently. The first task is represented by a lambda expression that calls a method named `DoSomeWork`, and the second task is represented by a lambda expression that calls a method named `DoSomeOtherWork`.

NOTE

This documentation uses lambda expressions to define delegates in TPL. If you aren't familiar with lambda expressions in C# or Visual Basic, see [Lambda Expressions in PLINQ and TPL](#).

```
Parallel.Invoke(() => DoSomeWork(), () => DoSomeOtherWork());
```

```
Parallel.Invoke(Sub() DoSomeWork(), Sub() DoSomeOtherWork())
```

NOTE

The number of [Task](#) instances that are created behind the scenes by [Invoke](#) isn't necessarily equal to the number of delegates that are provided. The TPL might employ various optimizations, especially with large numbers of delegates.

For more information, see [How to: Use Parallel.Invoke to Execute Parallel Operations](#).

For greater control over task execution or to return a value from the task, you must work with [Task](#) objects more explicitly.

Creating and running tasks explicitly

A task that doesn't return a value is represented by the [System.Threading.Tasks.Task](#) class. A task that returns a value is represented by the [System.Threading.Tasks.Task<TResult>](#) class, which inherits from [Task](#). The task object handles the infrastructure details and provides methods and properties that are accessible from the calling thread throughout the lifetime of the task. For example, you can access the [Status](#) property of a task at any time to determine whether it has started running, ran to completion, was canceled, or has thrown an exception. The status is represented by a [TaskStatus](#) enumeration.

When you create a task, you give it a user delegate that encapsulates the code that the task will execute. The delegate can be expressed as a named delegate, an anonymous method, or a lambda expression. Lambda expressions can contain a call to a named method, as shown in the following example. The example includes a call to the [Task.Wait](#) method to ensure that the task completes execution before the console mode application ends.

```
using System;
using System.Threading;
using System.Threading.Tasks;

public class Example
{
    public static void Main()
    {
        Thread.CurrentThread.Name = "Main";

        // Create a task and supply a user delegate by using a lambda expression.
        Task taskA = new Task( () => Console.WriteLine("Hello from taskA."));
        // Start the task.
        taskA.Start();

        // Output a message from the calling thread.
        Console.WriteLine("Hello from thread '{0}'.",
                         Thread.CurrentThread.Name);
        taskA.Wait();
    }
}

// The example displays output like the following:
//      Hello from thread 'Main'.
//      Hello from taskA.
```

```
Imports System.Threading
Imports System.Threading.Tasks

Module Example
    Public Sub Main()
        Thread.CurrentThread.Name = "Main"

        ' Create a task and supply a user delegate by using a lambda expression.
        Dim taskA = New Task(Sub() Console.WriteLine("Hello from taskA."))
        ' Start the task.
        taskA.Start()

        ' Output a message from the calling thread.
        Console.WriteLine("Hello from thread '{0}'.",
                         Thread.CurrentThread.Name)
        taskA.Wait()
    End Sub
End Module

' The example displays output like the following:
'      Hello from thread 'Main'.
'      Hello from taskA.
```

You can also use the [Task.Run](#) methods to create and start a task in one operation. To manage the task, the [Run](#) methods use the default task scheduler, regardless of which task scheduler is associated with the current thread. The [Run](#) methods are the preferred way to create and start tasks when more control over the creation and scheduling of the task isn't needed.

```
using System;
using System.Threading;
using System.Threading.Tasks;

public class Example
{
    public static void Main()
    {
        Thread.CurrentThread.Name = "Main";

        // Define and run the task.
        Task taskA = Task.Run( () => Console.WriteLine("Hello from taskA."));

        // Output a message from the calling thread.
        Console.WriteLine("Hello from thread '{0}'.",
                          Thread.CurrentThread.Name);
        taskA.Wait();
    }
}

// The example displays output like the following:
//      Hello from thread 'Main'.
//      Hello from taskA.
```

```
Imports System.Threading
Imports System.Threading.Tasks

Module Example
    Public Sub Main()
        Thread.CurrentThread.Name = "Main"

        Dim taskA As Task = Task.Run(Sub() Console.WriteLine("Hello from taskA."))

        ' Output a message from the calling thread.
        Console.WriteLine("Hello from thread '{0}'.",
                          Thread.CurrentThread.Name)
        taskA.Wait()
    End Sub
End Module

' The example displays output like the following:
'      Hello from thread 'Main'.
'      Hello from taskA.
```

You can also use the [TaskFactory.StartNew](#) method to create and start a task in one operation. As shown in the following example, you can use this method when:

- Creation and scheduling don't have to be separated and you require additional task creation options or the use of a specific scheduler.
- You need to pass additional state into the task that you can retrieve through its [Task.AsyncState](#) property.

```
using System;
using System.Threading;
using System.Threading.Tasks;

class CustomData
{
    public long CreationTime;
    public int Name;
    public int ThreadNum;
}

public class Example
{
    public static void Main()
    {
        Task[] taskArray = new Task[10];
        for (int i = 0; i < taskArray.Length; i++) {
            taskArray[i] = Task.Factory.StartNew( (Object obj) => {
                CustomData data = obj as CustomData;
                if (data == null)
                    return;

                data.ThreadNum = Thread.CurrentThread.ManagedThreadId;
            },
            new CustomData() {Name = i, CreationTime =
DateTime.Now.Ticks} );
        }

        Task.WaitAll(taskArray);
        foreach (var task in taskArray) {
            var data = task.AsyncState as CustomData;
            if (data != null)
                Console.WriteLine("Task #{0} created at {1}, ran on thread #{2}.",
                    data.Name, data.CreationTime, data.ThreadNum);
        }
    }
}

// The example displays output like the following:
//      Task #0 created at 635116412924597583 on thread #3.
//      Task #1 created at 635116412924607584 on thread #4.
//      Task #3 created at 635116412924607584 on thread #4.
//      Task #4 created at 635116412924607584 on thread #4.
//      Task #2 created at 635116412924607584 on thread #3.
//      Task #6 created at 635116412924607584 on thread #3.
//      Task #5 created at 635116412924607584 on thread #4.
//      Task #8 created at 635116412924607584 on thread #4.
//      Task #7 created at 635116412924607584 on thread #3.
//      Task #9 created at 635116412924607584 on thread #4.
```

```

Imports System.Threading
Imports System.Threading.Tasks

Class CustomData
    Public CreationTime As Long
    Public Name As Integer
    Public ThreadNum As Integer
End Class

Module Example
    Public Sub Main()
        Dim taskArray(9) As Task
        For i As Integer = 0 To taskArray.Length - 1
            taskArray(i) = Task.Factory.StartNew(Sub(obj As Object)
                Dim data As CustomData = TryCast(obj, CustomData)
                If data Is Nothing Then Return

                data.ThreadNum = Thread.CurrentThread.ManagedThreadId
            End Sub,
            New CustomData With {.Name = i, .CreationTime =
DateTime.Now.Ticks})
        Next
        Task.WaitAll(taskArray)

        For Each task In taskArray
            Dim data = TryCast(task.AsyncState, CustomData)
            If data IsNot Nothing Then
                Console.WriteLine("Task #{0} created at {1}, ran on thread #{2}.",
data.Name, data.CreationTime, data.ThreadNum)
            End If
        Next
    End Sub
End Module
' The example displays output like the following:
'   Task #0 created at 635116451245250515, ran on thread #3, RanToCompletion
'   Task #1 created at 635116451245270515, ran on thread #4, RanToCompletion
'   Task #2 created at 635116451245270515, ran on thread #3, RanToCompletion
'   Task #3 created at 635116451245270515, ran on thread #3, RanToCompletion
'   Task #4 created at 635116451245270515, ran on thread #3, RanToCompletion
'   Task #5 created at 635116451245270515, ran on thread #3, RanToCompletion
'   Task #6 created at 635116451245270515, ran on thread #3, RanToCompletion
'   Task #7 created at 635116451245270515, ran on thread #3, RanToCompletion
'   Task #8 created at 635116451245270515, ran on thread #3, RanToCompletion
'   Task #9 created at 635116451245270515, ran on thread #3, RanToCompletion

```

[Task](#) and [Task<TResult>](#) each expose a static [Factory](#) property that returns a default instance of [TaskFactory](#), so that you can call the method as `Task.Factory.StartNew()`. Also, in the following example, because the tasks are of type [System.Threading.Tasks.Task<TResult>](#), they each have a public [Task<TResult>.Result](#) property that contains the result of the computation. The tasks run asynchronously and might complete in any order. If the [Result](#) property is accessed before the computation finishes, the property blocks the calling thread until the value is available.

```

using System;
using System.Threading.Tasks;

public class Example
{
    public static void Main()
    {
        Task<Double>[] taskArray = { Task<Double>.Factory.StartNew(() => DoComputation(1.0)),
                                    Task<Double>.Factory.StartNew(() => DoComputation(100.0)),
                                    Task<Double>.Factory.StartNew(() => DoComputation(1000.0)) };

        var results = new Double[taskArray.Length];
        Double sum = 0;

        for (int i = 0; i < taskArray.Length; i++) {
            results[i] = taskArray[i].Result;
            Console.WriteLine("{0:N1} {1}", results[i],
                i == taskArray.Length - 1 ? "=" : "+ ");
            sum += results[i];
        }
        Console.WriteLine("{0:N1}", sum);
    }

    private static Double DoComputation(Double start)
    {
        Double sum = 0;
        for (var value = start; value <= start + 10; value += .1)
            sum += value;

        return sum;
    }
}

// The example displays the following output:
//      606.0 + 10,605.0 + 100,495.0 = 111,706.0

```

```

Imports System.Threading.Tasks

Module Example
    Public Sub Main()
        Dim taskArray() = {Task(Of Double).Factory.StartNew(Function() DoComputation(1.0)),
                        Task(Of Double).Factory.StartNew(Function() DoComputation(100.0)),
                        Task(Of Double).Factory.StartNew(Function() DoComputation(1000.0))}

        Dim results(taskArray.Length - 1) As Double
        Dim sum As Double

        For i As Integer = 0 To taskArray.Length - 1
            results(i) = taskArray(i).Result
            Console.WriteLine("{0:N1} {1}", results(i),
                If(i = taskArray.Length - 1, "=", "+ "))
            sum += results(i)
        Next
        Console.WriteLine("{0:N1}", sum)
    End Sub

    Private Function DoComputation(start As Double) As Double
        Dim sum As Double
        For value As Double = start To start + 10 Step .1
            sum += value
        Next
        Return sum
    End Function
End Module

' The example displays the following output:
'      606.0 + 10,605.0 + 100,495.0 = 111,706.0

```

For more information, see [How to: Return a Value from a Task](#).

When you use a lambda expression to create a delegate, you have access to all the variables that are visible at that point in your source code. However, in some cases, most notably within loops, a lambda doesn't capture the variable as expected. It only captures the reference of the variable, not the value, as it mutates after each iteration. The following example illustrates the problem. It passes a loop counter to a lambda expression that instantiates a `CustomData` object and uses the loop counter as the object's identifier. As the output from the example shows, each `CustomData` object has an identical identifier.

```
using System;
using System.Threading;
using System.Threading.Tasks;

class CustomData
{
    public long CreationTime;
    public int Name;
    public int ThreadNum;
}

public class Example
{
    public static void Main()
    {
        // Create the task object by using an Action(Of Object) to pass in the loop
        // counter. This produces an unexpected result.
        Task[] taskArray = new Task[10];
        for (int i = 0; i < taskArray.Length; i++) {
            taskArray[i] = Task.Factory.StartNew( (Object obj) => {
                var data = new CustomData() {Name = i, CreationTime =
DateTime.Now.Ticks};
                data.ThreadNum = Thread.CurrentThread.ManagedThreadId;
                Console.WriteLine("Task #{0} created at {1} on thread #
{2}.",
data.Name, data.CreationTime,
data.ThreadNum);
            },
            i );
        }
        Task.WaitAll(taskArray);
    }
}

// The example displays output like the following:
//      Task #10 created at 635116418427727841 on thread #4.
//      Task #10 created at 635116418427737842 on thread #4.
//      Task #10 created at 635116418427737842 on thread #4.
//      Task #10 created at 635116418427737842 on thread #4.
//      Task #10 created at 635116418427737842 on thread #4.
//      Task #10 created at 635116418427737842 on thread #4.
//      Task #10 created at 635116418427727841 on thread #3.
//      Task #10 created at 635116418427747843 on thread #3.
//      Task #10 created at 635116418427747843 on thread #3.
//      Task #10 created at 635116418427737842 on thread #4.
```

```

Imports System.Threading
Imports System.Threading.Tasks

Class CustomData
    Public CreationTime As Long
    Public Name As Integer
    Public ThreadNum As Integer
End Class

Module Example
    Public Sub Main()
        ' Create the task object by using an Action(Of Object) to pass in the loop
        ' counter. This produces an unexpected result.
        Dim taskArray(9) As Task
        For i As Integer = 0 To taskArray.Length - 1
            taskArray(i) = Task.Factory.StartNew(Sub(obj As Object)
                Dim data As New CustomData With {.Name = i,
                .CreationTime = DateTime.Now.Ticks}
                data.ThreadNum = Thread.CurrentThread.ManagedThreadId
                Console.WriteLine("Task #{0} created at {1} on thread #
{2}.",
                data.Name, data.CreationTime,
                data.ThreadNum)
            End Sub,
            i)
        Next
        Task.WaitAll(taskArray)
    End Sub
End Module
' The example displays output like the following:
'   Task #10 created at 635116418427727841 on thread #4.
'   Task #10 created at 635116418427737842 on thread #4.
'   Task #10 created at 635116418427737842 on thread #4.
'   Task #10 created at 635116418427737842 on thread #4.
'   Task #10 created at 635116418427737842 on thread #4.
'   Task #10 created at 635116418427737842 on thread #4.
'   Task #10 created at 635116418427737842 on thread #4.
'   Task #10 created at 635116418427727841 on thread #3.
'   Task #10 created at 635116418427747843 on thread #3.
'   Task #10 created at 635116418427747843 on thread #3.
'   Task #10 created at 635116418427737842 on thread #4.

```

You can access the value on each iteration by providing a state object to a task through its constructor. The following example modifies the previous example by using the loop counter when creating the `CustomData` object, which, in turn, is passed to the lambda expression. As the output from the example shows, each `CustomData` object now has a unique identifier based on the value of the loop counter at the time the object was instantiated.

```

using System;
using System.Threading;
using System.Threading.Tasks;

class CustomData
{
    public long CreationTime;
    public int Name;
    public int ThreadNum;
}

public class Example
{
    public static void Main()
    {
        // Create the task object by using an Action(Of Object) to pass in custom data
        // to the Task constructor. This is useful when you need to capture outer variables
        // from within a loop.
        Task[] taskArray = new Task[10];
        for (int i = 0; i < taskArray.Length; i++) {
            taskArray[i] = Task.Factory.StartNew( (Object obj) => {
                CustomData data = obj as CustomData;
                if (data == null)
                    return;

                data.ThreadNum = Thread.CurrentThread.ManagedThreadId;
                Console.WriteLine("Task #{0} created at {1} on thread #
{2}.",
                    data.Name, data.CreationTime,
                    data.ThreadNum);
            },
            new CustomData() {Name = i, CreationTime =
DateTime.Now.Ticks} );
        }
        Task.WaitAll(taskArray);
    }
}

// The example displays output like the following:
//      Task #0 created at 635116412924597583 on thread #3.
//      Task #1 created at 635116412924607584 on thread #4.
//      Task #3 created at 635116412924607584 on thread #4.
//      Task #4 created at 635116412924607584 on thread #4.
//      Task #2 created at 635116412924607584 on thread #3.
//      Task #6 created at 635116412924607584 on thread #3.
//      Task #5 created at 635116412924607584 on thread #4.
//      Task #8 created at 635116412924607584 on thread #4.
//      Task #7 created at 635116412924607584 on thread #3.
//      Task #9 created at 635116412924607584 on thread #4.

```

```

Imports System.Threading
Imports System.Threading.Tasks

Class CustomData
    Public CreationTime As Long
    Public Name As Integer
    Public ThreadNum As Integer
End Class

Module Example
    Public Sub Main()
        ' Create the task object by using an Action(Of Object) to pass in custom data
        ' to the Task constructor. This is useful when you need to capture outer variables
        ' from within a loop.
        Dim taskArray(9) As Task
        For i As Integer = 0 To taskArray.Length - 1
            taskArray(i) = Task.Factory.StartNew(Sub(obj As Object)
                Dim data As CustomData = TryCast(obj, CustomData)
                If data Is Nothing Then Return

                data.ThreadNum = Thread.CurrentThread.ManagedThreadId
                Console.WriteLine("Task #{0} created at {1} on thread #
{2}.",
                    data.Name, data.CreationTime,
                    data.ThreadNum)
            End Sub,
            New CustomData With {.Name = i, .CreationTime =
DateTime.Now.Ticks})
        Next
        Task.WaitAll(taskArray)
    End Sub
End Module
' The example displays output like the following:
'     Task #0 created at 635116412924597583 on thread #3.
'     Task #1 created at 635116412924607584 on thread #4.
'     Task #3 created at 635116412924607584 on thread #4.
'     Task #4 created at 635116412924607584 on thread #4.
'     Task #2 created at 635116412924607584 on thread #3.
'     Task #6 created at 635116412924607584 on thread #3.
'     Task #5 created at 635116412924607584 on thread #4.
'     Task #8 created at 635116412924607584 on thread #4.
'     Task #7 created at 635116412924607584 on thread #3.
'     Task #9 created at 635116412924607584 on thread #4.

```

This state is passed as an argument to the task delegate, and it can be accessed from the task object by using the [Task.AsyncState](#) property. The following example is a variation on the previous example. It uses the [AsyncState](#) property to display information about the `CustomData` objects passed to the lambda expression.

```
using System;
using System.Threading;
using System.Threading.Tasks;

class CustomData
{
    public long CreationTime;
    public int Name;
    public int ThreadNum;
}

public class Example
{
    public static void Main()
    {
        Task[] taskArray = new Task[10];
        for (int i = 0; i < taskArray.Length; i++) {
            taskArray[i] = Task.Factory.StartNew( (Object obj) => {
                CustomData data = obj as CustomData;
                if (data == null)
                    return;

                data.ThreadNum = Thread.CurrentThread.ManagedThreadId;
            },
            new CustomData() {Name = i, CreationTime =
DateTime.Now.Ticks} );
        }

        Task.WaitAll(taskArray);
        foreach (var task in taskArray) {
            var data = task.AsyncState as CustomData;
            if (data != null)
                Console.WriteLine("Task #{0} created at {1}, ran on thread #{2}.",
                    data.Name, data.CreationTime, data.ThreadNum);
        }
    }
}

// The example displays output like the following:
//      Task #0 created at 635116412924597583 on thread #3.
//      Task #1 created at 635116412924607584 on thread #4.
//      Task #3 created at 635116412924607584 on thread #4.
//      Task #4 created at 635116412924607584 on thread #4.
//      Task #2 created at 635116412924607584 on thread #3.
//      Task #6 created at 635116412924607584 on thread #3.
//      Task #5 created at 635116412924607584 on thread #4.
//      Task #8 created at 635116412924607584 on thread #4.
//      Task #7 created at 635116412924607584 on thread #3.
//      Task #9 created at 635116412924607584 on thread #4.
```

```

Imports System.Threading
Imports System.Threading.Tasks

Class CustomData
    Public CreationTime As Long
    Public Name As Integer
    Public ThreadNum As Integer
End Class

Module Example
    Public Sub Main()
        Dim taskArray(9) As Task
        For i As Integer = 0 To taskArray.Length - 1
            taskArray(i) = Task.Factory.StartNew(Sub(obj As Object)
                Dim data As CustomData = TryCast(obj, CustomData)
                If data Is Nothing Then Return

                data.ThreadNum = Thread.CurrentThread.ManagedThreadId
            End Sub,
            New CustomData With {.Name = i, .CreationTime =
DateTime.Now.Ticks})
        Next
        Task.WaitAll(taskArray)

        For Each task In taskArray
            Dim data = TryCast(task.AsyncState, CustomData)
            If data IsNot Nothing Then
                Console.WriteLine("Task #{0} created at {1}, ran on thread #{2}.",
data.Name, data.CreationTime, data.ThreadNum)
            End If
        Next
    End Sub
End Module
' The example displays output like the following:
'   Task #0 created at 635116451245250515, ran on thread #3, RanToCompletion
'   Task #1 created at 635116451245270515, ran on thread #4, RanToCompletion
'   Task #2 created at 635116451245270515, ran on thread #3, RanToCompletion
'   Task #3 created at 635116451245270515, ran on thread #3, RanToCompletion
'   Task #4 created at 635116451245270515, ran on thread #3, RanToCompletion
'   Task #5 created at 635116451245270515, ran on thread #3, RanToCompletion
'   Task #6 created at 635116451245270515, ran on thread #3, RanToCompletion
'   Task #7 created at 635116451245270515, ran on thread #3, RanToCompletion
'   Task #8 created at 635116451245270515, ran on thread #3, RanToCompletion
'   Task #9 created at 635116451245270515, ran on thread #3, RanToCompletion

```

Task ID

Every task receives an integer ID that uniquely identifies it in an application domain and can be accessed by using the [Task.Id](#) property. The ID is useful for viewing task information in the Visual Studio debugger **Parallel Stacks** and **Tasks** windows. The ID is created lazily, which means it isn't created until it's requested. Therefore, a task might have a different ID every time the program is run. For more information about how to view task IDs in the debugger, see [Using the Tasks Window](#) and [Using the Parallel Stacks Window](#).

Task creation options

Most APIs that create tasks provide overloads that accept a [TaskCreationOptions](#) parameter. By specifying one or more of these options, you tell the task scheduler how to schedule the task on the thread pool. Options might be combined by using a bitwise OR operation.

The following example shows a task that has the [LongRunning](#) and [PreferFairness](#) options:

```
var task3 = new Task(() => MyLongRunningMethod(),
    TaskCreationOptions.LongRunning | TaskCreationOptions.PreferFairness);
task3.Start();
```

```
Dim task3 = New Task(Sub() MyLongRunningMethod(),
    TaskCreationOptions.LongRunning Or TaskCreationOptions.PreferFairness)
task3.Start()
```

Tasks, threads, and culture

Each thread has an associated culture and UI culture, which are defined by the [Thread.CurrentCulture](#) and [Thread.CurrentUICulture](#) properties, respectively. A thread's culture is used in operations such as, formatting, parsing, sorting, and string comparison operations. A thread's UI culture is used in resource lookup.

The system culture defines the default culture and UI culture of a thread. However, you can specify a default culture for all the threads in an application domain by using the [CultureInfo.DefaultThreadCurrentCulture](#) and [CultureInfo.DefaultThreadCurrentUICulture](#) properties. If you explicitly set a thread's culture and launch a new thread, the new thread doesn't inherit the culture of the calling thread; instead, its culture is the default system culture. However, in task-based programming, tasks use the calling thread's culture, even if the task runs asynchronously on a different thread.

The following example provides a simple illustration. It changes the app's current culture to French (France). If French (France) is already the current culture, it changes to English (United States). It then invokes a delegate named `formatDelegate` that returns some numbers formatted as currency values in the new culture. Whether the delegate is invoked by a task either synchronously or asynchronously, the task uses the culture of the calling thread.

```
using System;
using System.Globalization;
using System.Threading;
using System.Threading.Tasks;

public class Example
{
    public static void Main()
    {
        decimal[] values = { 163025412.32m, 18905365.59m };
        string formatString = "C2";
        Func<String> formatDelegate = () => { string output = String.Format("Formatting using the {0} culture
on thread {1}.\n",
                                         CultureInfo.CurrentCulture.Name,
                                         Thread.CurrentThread.ManagedThreadId);
                                                foreach (var value in values)
                                                    output += String.Format("{0}    ",
value.ToString(formatString));
                                                output += Environment.NewLine;
                                                return output;
                                            };

        Console.WriteLine("The example is running on thread {0}",
                         Thread.CurrentThread.ManagedThreadId);
        // Make the current culture different from the system culture.
        Console.WriteLine("The current culture is {0}",
                         CultureInfo.CurrentCulture.Name);
        if (CultureInfo.CurrentCulture.Name == "fr-FR")
            Thread.CurrentThread.CurrentCulture = new CultureInfo("en-US");
        else
```

```

else
    Thread.CurrentThread.CurrentCulture = new CultureInfo("fr-FR");

Console.WriteLine("Changed the current culture to {0}.\n",
                 CultureInfo.CurrentCulture.Name);

// Execute the delegate synchronously.
Console.WriteLine("Executing the delegate synchronously:");
Console.WriteLine(formatDelegate());

// Call an async delegate to format the values using one format string.
Console.WriteLine("Executing a task asynchronously:");
var t1 = Task.Run(formatDelegate);
Console.WriteLine(t1.Result);

Console.WriteLine("Executing a task synchronously:");
var t2 = new Task<String>(formatDelegate);
t2.RunSynchronously();
Console.WriteLine(t2.Result);
}

}

// The example displays the following output:
//      The example is running on thread 1
//      The current culture is en-US
//      Changed the current culture to fr-FR.
//
//      Executing the delegate synchronously:
//      Formatting using the fr-FR culture on thread 1.
//      163 025 412,32 €  18 905 365,59 €
//
//      Executing a task asynchronously:
//      Formatting using the fr-FR culture on thread 3.
//      163 025 412,32 €  18 905 365,59 €
//
//      Executing a task synchronously:
//      Formatting using the fr-FR culture on thread 1.
//      163 025 412,32 €  18 905 365,59 €

```

```

Imports System.Globalization
Imports System.Threading

Module Example
    Public Sub Main()
        Dim values() As Decimal = {163025412.32D, 18905365.59D}
        Dim formatString As String = "C2"
        Dim formatDelegate As Func(Of String) = Function()
            Dim output As String = String.Format("Formatting using
the {0} culture on thread {1}.",
                                         CultureInfo.CurrentCulture.Name,
                                         Thread.CurrentThread.ManagedThreadId)
            output += Environment.NewLine
            For Each value In values
                output += String.Format("{0}    ",
                                         value.ToString(formatString))
            Next
            output += Environment.NewLine
            Return output
        End Function

        Console.WriteLine("The example is running on thread {0}",
                         Thread.CurrentThread.ManagedThreadId)
        ' Make the current culture different from the system culture.
        Console.WriteLine("The current culture is {0}",
                         CultureInfo.CurrentCulture.Name)
        If CultureInfo.CurrentCulture.Name = "fr-FR" Then
            Thread.CurrentThread.CurrentCulture = New CultureInfo("en-US")

```

```

Else
    Thread.CurrentThread.CurrentCulture = New CultureInfo("fr-FR")
End If
Console.WriteLine("Changed the current culture to {0}.",
                  CultureInfo.CurrentCulture.Name)
Console.WriteLine()

' Execute the delegate synchronously.
Console.WriteLine("Executing the delegate synchronously:")
Console.WriteLine(formatDelegate())

' Call an async delegate to format the values using one format string.
Console.WriteLine("Executing a task asynchronously:")
Dim t1 = Task.Run(formatDelegate)
Console.WriteLine(t1.Result)

Console.WriteLine("Executing a task synchronously:")
Dim t2 = New Task(Of String)(formatDelegate)
t2.RunSynchronously()
Console.WriteLine(t2.Result)
End Sub
End Module

' The example displays the following output:
'
'   The example is running on thread 1
'   The current culture is en-US
'   Changed the current culture to fr-FR.
'
'   Executing the delegate synchronously:
'   Formatting Imports the fr-FR culture on thread 1.
'   163 025 412,32 €  18 905 365,59 €
'
'   Executing a task asynchronously:
'   Formatting Imports the fr-FR culture on thread 3.
'   163 025 412,32 €  18 905 365,59 €
'
'   Executing a task synchronously:
'   Formatting Imports the fr-FR culture on thread 1.
'   163 025 412,32 €  18 905 365,59 €

```

NOTE

In versions of .NET Framework earlier than .NET Framework 4.6, a task's culture is determined by the culture of the thread on which it runs, not the culture of the calling thread. For asynchronous tasks, the culture used by the task could be different from the calling thread's culture.

For more information on asynchronous tasks and culture, see the "Culture and asynchronous task-based operations" section in the [CultureInfo](#) article.

Creating task continuations

The [Task.ContinueWith](#) and [Task<TResult>.ContinueWith](#) methods let you specify a task to start when the *antecedent task* finishes. The delegate of the continuation task is passed a reference to the antecedent task so that it can examine the antecedent task's status. And by retrieving the value of the [Task<TResult>.Result](#) property, you can use the output of the antecedent as input for the continuation.

In the following example, the `getData` task is started by a call to the [TaskFactory.StartNew<TResult>\(Func<TResult>\)](#) method. The `processData` task is started automatically when `getData` finishes, and `displayData` is started when `processData` finishes. `getData` produces an integer array, which is accessible to the `processData` task through the `getData` task's [Task<TResult>.Result](#) property. The `processData` task

processes that array and returns a result whose type is inferred from the return type of the lambda expression passed to the `Task<TResult>.ContinueWith<TNewResult>(Func<Task<TResult>,TNewResult>)` method. The `displayData` task executes automatically when `processData` finishes, and the `Tuple<T1,T2,T3>` object returned by the `processData` lambda expression is accessible to the `displayData` task through the `processData` task's `Task<TResult>.Result` property. The `displayData` task takes the result of the `processData` task. It produces a result whose type is inferred in a similar manner, and which is made available to the program in the `Result` property.

```
using System;
using System.Threading.Tasks;

public class Example
{
    public static void Main()
    {
        var getData = Task.Factory.StartNew(() => {
            Random rnd = new Random();
            int[] values = new int[100];
            for (int ctr = 0; ctr <= values.GetUpperBound(0); ctr++)
                values[ctr] = rnd.Next();

            return values;
        });

        var processData = getData.ContinueWith((x) => {
            int n = x.Result.Length;
            long sum = 0;
            double mean;

            for (int ctr = 0; ctr <= x.Result.GetUpperBound(0); ctr++)
                sum += x.Result[ctr];

            mean = sum / (double) n;
            return Tuple.Create(n, sum, mean);
        });

        var displayData = processData.ContinueWith((x) => {
            return String.Format("N={0:N0}, Total = {1:N0}, Mean = {2:N2}",
                x.Result.Item1, x.Result.Item2,
                x.Result.Item3);
        });

        Console.WriteLine(displayData.Result);
    }
}

// The example displays output similar to the following:
//      N=100, Total = 110,081,653,682, Mean = 1,100,816,536.82
```

```

Imports System.Threading.Tasks

Module Example
    Public Sub Main()
        Dim getData = Task.Factory.StartNew(Function()
            Dim rnd As New Random()
            Dim values(99) As Integer
            For ctr = 0 To values.GetUpperBound(0)
                values(ctr) = rnd.Next()
            Next
            Return values
        End Function)
        Dim processData = getData.ContinueWith(Function(x)
            Dim n As Integer = x.Result.Length
            Dim sum As Long
            Dim mean As Double

            For ctr = 0 To x.Result.GetUpperBound(0)
                sum += x.Result(ctr)
            Next
            mean = sum / n
            Return Tuple.Create(n, sum, mean)
        End Function)
        Dim displayData = processData.ContinueWith(Function(x)
            Return String.Format("N={0:N0}, Total = {1:N0}, Mean
= {2:N2}",
                x.Result.Item1, x.Result.Item2,
                x.Result.Item3)
        End Function)
        Console.WriteLine(displayData.Result)
    End Sub
End Module
' The example displays output like the following:
'   N=100, Total = 110,081,653,682, Mean = 1,100,816,536.82

```

Because [Task.ContinueWith](#) is an instance method, you can chain method calls together instead of instantiating a [Task<TResult>](#) object for each antecedent task. The following example is functionally identical to the previous one, except that it chains together calls to the [Task.ContinueWith](#) method. The [Task<TResult>](#) object returned by the chain of method calls is the final continuation task.

```
using System;
using System.Threading.Tasks;

public class Example
{
    public static void Main()
    {
        var displayData = Task.Factory.StartNew(() => {
            Random rnd = new Random();
            int[] values = new int[100];
            for (int ctr = 0; ctr <= values.GetUpperBound(0); ctr++)
                values[ctr] = rnd.Next();

            return values;
        }).
        ContinueWith((x) => {
            int n = x.Result.Length;
            long sum = 0;
            double mean;

            for (int ctr = 0; ctr <= x.Result.GetUpperBound(0); ctr++)
                sum += x.Result[ctr];

            mean = sum / (double) n;
            return Tuple.Create(n, sum, mean);
        }).
        ContinueWith((x) => {
            return String.Format("N={0:N0}, Total = {1:N0}, Mean = {2:N2}",
                x.Result.Item1, x.Result.Item2,
                x.Result.Item3);
        });
        Console.WriteLine(displayData.Result);
    }
}

// The example displays output similar to the following:
//      N=100, Total = 110,081,653,682, Mean = 1,100,816,536.82
```

```

Imports System.Threading.Tasks

Module Example
    Public Sub Main()
        Dim displayData = Task.Factory.StartNew(Function()
            Dim rnd As New Random()
            Dim values(99) As Integer
            For ctr = 0 To values.GetUpperBound(0)
                values(ctr) = rnd.Next()
            Next
            Return values
        End Function). _
        ContinueWith(Function(x)
            Dim n As Integer = x.Result.Length
            Dim sum As Long
            Dim mean As Double

            For ctr = 0 To x.Result.GetUpperBound(0)
                sum += x.Result(ctr)
            Next
            mean = sum / n
            Return Tuple.Create(n, sum, mean)
        End Function). _
        ContinueWith(Function(x)
            Return String.Format("N={0:N0}, Total = {1:N0}, Mean = {2:N2}",
                x.Result.Item1, x.Result.Item2,
                x.Result.Item3)
        End Function)
        Console.WriteLine(displayData.Result)
    End Sub
End Module
' The example displays output like the following:
'   N=100, Total = 110,081,653,682, Mean = 1,100,816,536.82

```

The [ContinueWhenAll](#) and [ContinueWhenAny](#) methods enable you to continue from multiple tasks.

For more information, see [Chaining Tasks by Using Continuation Tasks](#).

Creating detached child tasks

When user code that's running in a task creates a new task and doesn't specify the [AttachedToParent](#) option, the new task isn't synchronized with the parent task in any special way. This type of non-synchronized task is called a *detached nested task* or *detached child task*. The following example shows a task that creates one detached child task:

```

var outer = Task.Factory.StartNew(() =>
{
    Console.WriteLine("Outer task beginning.");

    var child = Task.Factory.StartNew(() =>
    {
        Thread.Sleep(5000000);
        Console.WriteLine("Detached task completed.");
    });
});

outer.Wait();
Console.WriteLine("Outer task completed.");
// The example displays the following output:
//   Outer task beginning.
//   Outer task completed.
//   Detached task completed.

```

```
Dim outer = Task.Factory.StartNew(Sub()
    Console.WriteLine("Outer task beginning.")
    Dim child = Task.Factory.StartNew(Sub()
        Thread.SpinWait(5000000)
        Console.WriteLine("Detached task
completed.")
    End Sub)
    outer.Wait()
    Console.WriteLine("Outer task completed.")
    ' The example displays the following output:
    '   Outer task beginning.
    '   Outer task completed.
    '   Detached child completed.
```

NOTE

The parent task doesn't wait for the detached child task to finish.

Creating child tasks

When user code that's running in a task creates a task with the [AttachedToParent](#) option, the new task is known as an *attached child task* of the parent task. You can use the [AttachedToParent](#) option to express structured task parallelism because the parent task implicitly waits for all attached child tasks to finish. The following example shows a parent task that creates 10 attached child tasks. The example calls the [Task.Wait](#) method to wait for the parent task to finish. It doesn't have to explicitly wait for the attached child tasks to complete.

```
using System;
using System.Threading;
using System.Threading.Tasks;

public class Example
{
    public static void Main()
    {
        var parent = Task.Factory.StartNew(() => {
            Console.WriteLine("Parent task beginning.");
            for (int ctr = 0; ctr < 10; ctr++) {
                int taskNo = ctr;
                Task.Factory.StartNew((x) => {
                    Thread.SpinWait(5000000);
                    Console.WriteLine("Attached child #{0} completed.", x);
                },
                taskNo, TaskCreationOptions.AttachedToParent);
            }
        });

        parent.Wait();
        Console.WriteLine("Parent task completed.");
    }
}

// The example displays output like the following:
// Parent task beginning.
// Attached child #9 completed.
// Attached child #0 completed.
// Attached child #8 completed.
// Attached child #1 completed.
// Attached child #7 completed.
// Attached child #2 completed.
// Attached child #6 completed.
// Attached child #3 completed.
// Attached child #5 completed.
// Attached child #4 completed.
// Parent task completed.
```

```

Imports System.Threading
Imports System.Threading.Tasks

Module Example
    Public Sub Main()
        Dim parent = Task.Factory.StartNew(Sub()
            Console.WriteLine("Parent task beginning.")
            For ctr As Integer = 0 To 9
                Dim taskNo As Integer = ctr
                Task.Factory.StartNew(Sub(x)
                    Thread.SpinWait(5000000)
                    Console.WriteLine("Attached
child #{0} completed.",

x)
                End Sub,
                taskNo,
                TaskCreationOptions.AttachedToParent)
            Next
        End Sub)
        parent.Wait()
        Console.WriteLine("Parent task completed.")
    End Sub
End Module
' The example displays output like the following:
'   Parent task beginning.
'   Attached child #9 completed.
'   Attached child #0 completed.
'   Attached child #8 completed.
'   Attached child #1 completed.
'   Attached child #7 completed.
'   Attached child #2 completed.
'   Attached child #6 completed.
'   Attached child #3 completed.
'   Attached child #5 completed.
'   Attached child #4 completed.
'   Parent task completed.

```

A parent task can use the [TaskCreationOptions.DenyChildAttach](#) option to prevent other tasks from attaching to the parent task. For more information, see [Attached and Detached Child Tasks](#).

Waiting for tasks to finish

The [System.Threading.Tasks.Task](#) and [System.Threading.Tasks.Task<TResult>](#) types provide several overloads of the [Task.Wait](#) methods that enable you to wait for a task to finish. In addition, overloads of the static [Task.WaitAll](#) and [Task.WaitAny](#) methods let you wait for any or all of an array of tasks to finish.

Typically, you would wait for a task for one of these reasons:

- The main thread depends on the final result computed by a task.
- You have to handle exceptions that might be thrown from the task.
- The application might terminate before all tasks have completed execution. For example, console applications will terminate after all synchronous code in `Main` (the application entry point) has executed.

The following example shows the basic pattern that doesn't involve exception handling:

```

Task[] tasks = new Task[3]
{
    Task.Factory.StartNew(() => MethodA()),
    Task.Factory.StartNew(() => MethodB()),
    Task.Factory.StartNew(() => MethodC())
};

//Block until all tasks complete.
Task.WaitAll(tasks);

// Continue on this thread...

```

```

Dim tasks() =
{
    Task.Factory.StartNew(Sub() MethodA()),
    Task.Factory.StartNew(Sub() MethodB()),
    Task.Factory.StartNew(Sub() MethodC())
}

' Block until all tasks complete.
Task.WaitAll(tasks)

' Continue on this thread...

```

For an example that shows exception handling, see [Exception Handling](#).

Some overloads let you specify a time-out, and others take an additional [CancellationToken](#) as an input parameter so that the wait itself can be canceled either programmatically or in response to user input.

When you wait for a task, you implicitly wait for all children of that task that were created by using the [TaskCreationOptions.AttachedToParent](#) option. [Task.Wait](#) returns immediately if the task has already completed. A [Task.Wait](#) method will throw any exceptions raised by a task, even if the [Task.Wait](#) method was called after the task completed.

Composing tasks

The [Task](#) and [Task<TResult>](#) classes provide several methods to help you compose multiple tasks. These methods implement common patterns and make better use of the asynchronous language features that are provided by C#, Visual Basic, and F#. This section describes the [WhenAll](#), [WhenAny](#), [Delay](#), and [FromResult](#) methods.

Task.WhenAll

The [Task.WhenAll](#) method asynchronously waits for multiple [Task](#) or [Task<TResult>](#) objects to finish. It provides overloaded versions that enable you to wait for non-uniform sets of tasks. For example, you can wait for multiple [Task](#) and [Task<TResult>](#) objects to complete from one method call.

Task.WhenAny

The [Task.WhenAny](#) method asynchronously waits for one of multiple [Task](#) or [Task<TResult>](#) objects to finish. As in the [Task.WhenAll](#) method, this method provides overloaded versions that enable you to wait for non-uniform sets of tasks. The [WhenAny](#) method is especially useful in the following scenarios:

- **Redundant operations:** Consider an algorithm or operation that can be performed in many ways. You can use the [WhenAny](#) method to select the operation that finishes first and then cancel the remaining operations.
- **Interleaved operations:** You can start multiple operations that must finish and use the [WhenAny](#) method to process results as each operation finishes. After one operation finishes, you can start one or

more tasks.

- **Throttled operations:** You can use the [WhenAny](#) method to extend the previous scenario by limiting the number of concurrent operations.
- **Expired operations:** You can use the [WhenAny](#) method to select between one or more tasks and a task that finishes after a specific time, such as a task that's returned by the [Delay](#) method. The [Delay](#) method is described in the following section.

Task.Delay

The [Task.Delay](#) method produces a [Task](#) object that finishes after the specified time. You can use this method to build loops that poll for data, to specify time-outs, to delay the handling of user input, and so on.

Task(T).FromResult

By using the [Task.FromResult](#) method, you can create a [Task<TResult>](#) object that holds a pre-computed result. This method is useful when you perform an asynchronous operation that returns a [Task<TResult>](#) object, and the result of that [Task<TResult>](#) object is already computed. For an example that uses [FromResult](#) to retrieve the results of asynchronous download operations that are held in a cache, see [How to: Create Pre-Computed Tasks](#).

Handling exceptions in tasks

When a task throws one or more exceptions, the exceptions are wrapped in an [AggregateException](#) exception. That exception is propagated back to the thread that joins with the task. Typically, it's the thread waiting for the task to finish or the thread accessing the [Result](#) property. This behavior enforces the .NET Framework policy that all unhandled exceptions by default should terminate the process. The calling code can handle the exceptions by using any of the following in a `try / catch` block:

- The [Wait](#) method
- The [WaitAll](#) method
- The [WaitAny](#) method
- The [Result](#) property

The joining thread can also handle exceptions by accessing the [Exception](#) property before the task is garbage-collected. By accessing this property, you prevent the unhandled exception from triggering the exception propagation behavior that terminates the process when the object is finalized.

For more information about exceptions and tasks, see [Exception Handling](#).

Cancelling tasks

The [Task](#) class supports cooperative cancellation and is fully integrated with the [System.Threading.CancellationTokenSource](#) and [System.Threading.CancellationToken](#) classes, which were introduced in the .NET Framework 4. Many of the constructors in the [System.Threading.Tasks.Task](#) class take a [CancellationToken](#) object as an input parameter. Many of the [StartNew](#) and [Run](#) overloads also include a [CancellationToken](#) parameter.

You can create the token and issue the cancellation request at some later time, by using the [CancellationTokenSource](#) class. Pass the token to the [Task](#) as an argument, and also reference the same token in your user delegate, which does the work of responding to a cancellation request.

For more information, see [Task Cancellation](#) and [How to: Cancel a Task and Its Children](#).

The TaskFactory class

The [TaskFactory](#) class provides static methods that encapsulate common patterns for creating and starting tasks and continuation tasks.

- The most common pattern is [StartNew](#), which creates and starts a task in one statement.
- When you create continuation tasks from multiple antecedents, use the [ContinueWhenAll](#) method or [ContinueWhenAny](#) method or their equivalents in the [Task<TResult>](#) class. For more information, see [Chaining Tasks by Using Continuation Tasks](#).
- To encapsulate Asynchronous Programming Model [BeginX](#) and [EndX](#) methods in a [Task](#) or [Task<TResult>](#) instance, use the [FromAsync](#) methods. For more information, see [TPL and Traditional .NET Framework Asynchronous Programming](#).

The default [TaskFactory](#) can be accessed as a static property on the [Task](#) class or [Task<TResult>](#) class. You can also instantiate a [TaskFactory](#) directly and specify various options that include a [CancellationToken](#), a [TaskCreationOptions](#) option, a [TaskContinuationOptions](#) option, or a [TaskScheduler](#). Whatever options are specified when you create the task factory will be applied to all tasks that it creates unless the [Task](#) is created by using the [TaskCreationOptions](#) enumeration, in which case the task's options override those of the task factory.

Tasks without delegates

In some cases, you might want to use a [Task](#) to encapsulate some asynchronous operation that's performed by an external component instead of your user delegate. If the operation is based on the Asynchronous Programming Model Begin/End pattern, you can use the [FromAsync](#) methods. If that's not the case, you can use the [TaskCompletionSource<TResult>](#) object to wrap the operation in a task and thereby gain some of the benefits of [Task](#) programmability. For example, support for exception propagation and continuations. For more information, see [TaskCompletionSource<TResult>](#).

Custom schedulers

Most application or library developers don't care which processor the task runs on, how it synchronizes its work with other tasks, or how it's scheduled on the [System.Threading.ThreadPool](#). They only require that it execute as efficiently as possible on the host computer. If you require more fine-grained control over the scheduling details, the TPL lets you configure some settings on the default task scheduler, and even lets you supply a custom scheduler. For more information, see [TaskScheduler](#).

Related data structures

The TPL has several new public types that are useful in parallel and sequential scenarios. These include several thread-safe, fast, and scalable collection classes in the [System.Collections.Concurrent](#) namespace and several new synchronization types. For example, [System.Threading.Semaphore](#) and [System.Threading.ManualResetEventSlim](#), which are more efficient than their predecessors for specific kinds of workloads. Other new types in the .NET Framework 4, for example, [System.Threading.Barrier](#) and [System.Threading.SpinLock](#), provide functionality that wasn't available in earlier releases. For more information, see [Data Structures for Parallel Programming](#).

Custom task types

We recommend that you don't inherit from [System.Threading.Tasks.Task](#) or [System.Threading.Tasks.Task<TResult>](#). Instead, we recommend that you use the [AsyncState](#) property to associate additional data or state with a [Task](#) or [Task<TResult>](#) object. You can also use extension methods to extend the functionality of the [Task](#) and [Task<TResult>](#) classes. For more information about extension methods, see [Extension Methods](#) and [Extension Methods](#).

If you must inherit from [Task](#) or [Task<TResult>](#), you can't use [Run](#) or the [System.Threading.Tasks.TaskFactory](#),

`System.Threading.Tasks.TaskFactory<TResult>`, or `System.Threading.Tasks.TaskCompletionSource<TResult>` classes to create instances of your custom task type. You can't use them because these classes create only `Task` and `Task<TResult>` objects. In addition, you can't use the task continuation mechanisms that are provided by `Task`, `Task<TResult>`, `TaskFactory`, and `TaskFactory<TResult>` to create instances of your custom task type. You can't use them because these classes also create only `Task` and `Task<TResult>` objects.

Related sections

TITLE	DESCRIPTION
Chaining Tasks by Using Continuation Tasks	Describes how continuations work.
Attached and Detached Child Tasks	Describes the difference between attached and detached child tasks.
Task Cancellation	Describes the cancellation support that's built into the <code>Task</code> object.
Exception Handling	Describes how exceptions on concurrent threads are handled.
How to: Use Parallel.Invoke to Execute Parallel Operations	Describes how to use <code>Invoke</code> .
How to: Return a Value from a Task	Describes how to return values from tasks.
How to: Cancel a Task and Its Children	Describes how to cancel tasks.
How to: Create Pre-Computed Tasks	Describes how to use the <code>Task.FromResult</code> method to retrieve the results of asynchronous download operations that are held in a cache.
How to: Traverse a Binary Tree with Parallel Tasks	Describes how to use tasks to traverse a binary tree.
How to: Unwrap a Nested Task	Demonstrates how to use the <code>Unwrap</code> extension method.
Data Parallelism	Describes how to use <code>For</code> and <code>ForEach</code> to create parallel loops over data.
Parallel Programming	Top-level node for .NET Framework parallel programming.

See also

- [Parallel Programming](#)
- [Samples for Parallel Programming with the .NET Core & .NET Standard](#)

Chaining tasks using continuation tasks

9/20/2022 • 28 minutes to read • [Edit Online](#)

In asynchronous programming, it's common for one asynchronous operation, on completion, to invoke a second operation. Continuations allow descendant operations to consume the results of the first operation. Traditionally, continuations have been done by using callback methods. In the Task Parallel Library, the same functionality is provided by *continuation tasks*. A continuation task (also known just as a continuation) is an asynchronous task that's invoked by another task, known as the *antecedent*, when the antecedent finishes.

Continuations are relatively easy to use, but are nevertheless powerful and flexible. For example, you can:

- Pass data from the antecedent to the continuation.
- Specify the precise conditions under which the continuation will be invoked or not invoked.
- Cancel a continuation either before it starts or cooperatively as it is running.
- Provide hints about how the continuation should be scheduled.
- Invoke multiple continuations from the same antecedent.
- Invoke one continuation when all or any one of multiple antecedents complete.
- Chain continuations one after another to any arbitrary length.
- Use a continuation to handle exceptions thrown by the antecedent.

About continuations

A continuation is a task that is created in the [WaitingForActivation](#) state. It is activated automatically when its antecedent task or tasks complete. Calling [Task.Start](#) on a continuation in user code throws an [System.InvalidOperationException](#) exception.

A continuation is itself a [Task](#) and does not block the thread on which it is started. Call the [Task.Wait](#) method to block until the continuation task finishes.

Create a continuation for a single antecedent

You create a continuation that executes when its antecedent has completed by calling the [Task.ContinueWith](#) method. The following example shows the basic pattern (for clarity, exception handling is omitted). It executes an antecedent task, `taskA`, that returns a [DayOfWeek](#) object that indicates the name of the current day of the week. When the antecedent completes, the continuation task, `continuation`, is passed the antecedent and displays a string that includes its result.

NOTE

The C# samples in this article make use of the `async` modifier on the `Main` method. That feature is available in C# 7.1 and later. Previous versions generate `CS5001` when compiling this sample code. You'll need to set the language version to C# 7.1 or newer. You can learn how to configure the language version in the article on [configure language version](#).

```

using System;
using System.Threading.Tasks;

public class SimpleExample
{
    public static async Task Main()
    {
        // Declare, assign, and start the antecedent task.
        Task<DayOfWeek> taskA = Task.Run(() => DateTime.Today.DayOfWeek);

        // Execute the continuation when the antecedent finishes.
        await taskA.ContinueWith(antecedent => Console.WriteLine($"Today is {antecedent.Result}."));

    }
}

// The example displays the following output:
//     Today is Monday.

```

```

Imports System.Threading.Tasks

Module Example
    Public Sub Main()
        ' Execute the antecedent.
        Dim taskA As Task(Of DayOfWeek) = Task.Run(Function() DateTime.Today.DayOfWeek)

        ' Execute the continuation when the antecedent finishes.
        Dim continuation As Task = taskA.ContinueWith(Sub(antecedent)
                                                       Console.WriteLine("Today is {0}.",
antecedent.Result))
                                                       End Sub)
        continuation.Wait()
    End Sub
End Module

' The example displays output like the following output:
'     Today is Monday.

```

Create a continuation for multiple antecedents

You can also create a continuation that will run when any or all of a group of tasks has completed. To execute a continuation when all antecedent tasks have completed, you call the static (`Shared` in Visual Basic) [Task.WhenAll](#) method or the instance [TaskFactory.ContinueWhenAll](#) method. To execute a continuation when any of the antecedent tasks has completed, you call the static (`Shared` in Visual Basic) [Task.WhenAny](#) method or the instance [TaskFactory.ContinueWhenAny](#) method.

Calls to the [Task.WhenAll](#) and [Task.WhenAny](#) overloads do not block the calling thread. However, you typically call all but the [Task.WhenAll\(IEnumerable<Task>\)](#) and [Task.WhenAll\(Task\[\]\)](#) methods to retrieve the returned [Task<TResult>.Result](#) property, which does block the calling thread.

The following example calls the [Task.WhenAll\(IEnumerable<Task>\)](#) method to create a continuation task that reflects the results of its 10 antecedent tasks. Each antecedent task squares an index value that ranges from one to 10. If the antecedents complete successfully (their [Task.Status](#) property is [TaskStatus.RanToCompletion](#)), the [Task<TResult>.Result](#) property of the continuation is an array of the [Task<TResult>.Result](#) values returned by each antecedent. The example adds them to compute the sum of squares for all numbers between one and 10.

```

using System.Collections.Generic;
using System;
using System.Threading.Tasks;

public class WhenAllExample
{
    public static async Task Main()
    {
        var tasks = new List<Task<int>>();
        for (int ctr = 1; ctr <= 10; ctr++)
        {
            int baseValue = ctr;
            tasks.Add(Task.Factory.StartNew(b => (int)b! * (int)b, baseValue));
        }

        var results = await Task.WhenAll(tasks);

        int sum = 0;
        for (int ctr = 0; ctr <= results.Length - 1; ctr++)
        {
            var result = results[ctr];
            Console.Write($"{result} {((ctr == results.Length - 1) ? "=" : "+")}");
            sum += result;
        }

        Console.WriteLine(sum);
    }
}

// The example displays the similar output:
//      1 + 4 + 9 + 16 + 25 + 36 + 49 + 64 + 81 + 100 = 385

```

```

Imports System.Collections.Generic
Imports System.Threading.Tasks

Module Example
    Public Sub Main()
        Dim tasks As New List(Of Task(Of Integer))()
        For ctr As Integer = 1 To 10
            Dim baseValue As Integer = ctr
            tasks.Add(Task.Factory.StartNew(Function(b)
                Dim i As Integer = CInt(b)
                Return i * i
            End Function, baseValue))
        Next
        Dim continuation = Task.WhenAll(tasks)

        Dim sum As Long = 0
        For ctr As Integer = 0 To continuation.Result.Length - 1
            Console.Write("{0} {1} ", continuation.Result(ctr),
                If(ctr = continuation.Result.Length - 1, "=", "+"))
            sum += continuation.Result(ctr)
        Next
        Console.WriteLine(sum)
    End Sub
End Module
' The example displays the following output:
'      1 + 4 + 9 + 16 + 25 + 36 + 49 + 64 + 81 + 100 = 385

```

Continuation options

When you create a single-task continuation, you can use a [ContinueWith](#) overload that takes a [System.Threading.Tasks.TaskContinuationOptions](#) enumeration value to specify the conditions under which the continuation starts. For example, you can specify that the continuation is to run only if the antecedent completes

successfully, or only if it completes in a faulted state. If the condition is not true when the antecedent is ready to invoke the continuation, the continuation transitions directly to the [TaskStatus.Canceled](#) state and subsequently cannot be started.

A number of multi-task continuation methods, such as overloads of the [TaskFactory.ContinueWhenAll](#) method, also include a [System.Threading.Tasks.TaskContinuationOptions](#) parameter. Only a subset of all [System.Threading.Tasks.TaskContinuationOptions](#) enumeration members are valid, however. You can specify [System.Threading.Tasks.TaskContinuationOptions](#) values that have counterparts in the [System.Threading.Tasks.TaskCreationOptions](#) enumeration, such as [TaskContinuationOptions.AttachedToParent](#), [TaskContinuationOptions.LongRunning](#), and [TaskContinuationOptions.PreferFairness](#). If you specify any of the [NotOn](#) or [OnlyOn](#) options with a multi-task continuation, an [ArgumentOutOfRangeException](#) exception will be thrown at run time.

For more information on task continuation options, see the [TaskContinuationOptions](#) topic.

Pass data to a continuation

The [Task.ContinueWith](#) method passes a reference to the antecedent to the user delegate of the continuation as an argument. If the antecedent is a [System.Threading.Tasks.Task<TResult>](#) object, and the task ran until it was completed, then the continuation can access the [Task<TResult>.Result](#) property of the task.

The [Task<TResult>.Result](#) property blocks until the task has completed. However, if the task was canceled or faulted, attempting to access the [Result](#) property throws an [AggregateException](#) exception. You can avoid this problem by using the [OnlyOnRanToCompletion](#) option, as shown in the following example.

```
using System;
using System.Threading.Tasks;

public class ResultExample
{
    public static async Task Main()
    {
        var task = Task.Run(
            () =>
            {
                DateTime date = DateTime.Now;
                return date.Hour > 17
                    ? "evening"
                    : date.Hour > 12
                        ? "afternoon"
                        : "morning";
            });

        await task.ContinueWith(
            antecedent =>
            {
                Console.WriteLine($"Good {antecedent.Result}!");
                Console.WriteLine($"And how are you this fine {antecedent.Result}?");
            }, TaskContinuationOptions.OnlyOnRanToCompletion);
    }
}

// The example displays the similar output:
//      Good afternoon!
//      And how are you this fine afternoon?
```

```

Imports System.Threading.Tasks

Module Example
    Public Sub Main()
        Dim t = Task.Run(Function()
            Dim dat As DateTime = DateTime.Now
            If dat = DateTime.MinValue Then
                Throw New ArgumentException("The clock is not working.")
            End If

            If dat.Hour > 17 Then
                Return "evening"
            Else If dat.Hour > 12 Then
                Return "afternoon"
            Else
                Return "morning"
            End If
        End Function)
        Dim c = t.ContinueWith(Sub(antecedent)
            Console.WriteLine("Good {0}!",
                antecedent.Result)
            Console.WriteLine("And how are you this fine {0}?",
                antecedent.Result)
        End Sub, TaskContinuationOptions.OnlyOnRanToCompletion)
        c.Wait()
    End Sub
End Module
' The example displays output like the following:
'     Good afternoon!
'     And how are you this fine afternoon?

```

If you want the continuation to run even if the antecedent did not run to successful completion, you must guard against the exception. One approach is to test the [Task.Status](#) property of the antecedent, and only attempt to access the [Result](#) property if the status is not [Faulted](#) or [Canceled](#). You can also examine the [Exception](#) property of the antecedent. For more information, see [Exception Handling](#). The following example modifies the previous example to access antecedent's [Task<TResult>.Result](#) property only if its status is [TaskStatus.RanToCompletion](#).

```
using System;
using System.Threading.Tasks;

public class ResultTwoExample
{
    public static async Task Main() =>
        await Task.Run(
            () =>
            {
                DateTime date = DateTime.Now;
                return date.Hour > 17
                    ? "evening"
                    : date.Hour > 12
                        ? "afternoon"
                        : "morning";
            })
            .ContinueWith(
                antecedent =>
                {
                    if (antecedent.Status == TaskStatus.RanToCompletion)
                    {
                        Console.WriteLine($"Good {antecedent.Result}!");
                        Console.WriteLine($"And how are you this fine {antecedent.Result}?");
                    }
                    else if (antecedent.Status == TaskStatus.Faulted)
                    {
                        Console.WriteLine(antecedent.Exception!.GetBaseException().Message);
                    }
                });
}

// The example displays output like the following:
//      Good afternoon!
//      And how are you this fine afternoon?
```

```

Imports System.Threading.Tasks

Module Example
    Public Sub Main()
        Dim t = Task.Run(Function()
            Dim dat As DateTime = DateTime.Now
            If dat = DateTime.MinValue Then
                Throw New ArgumentException("The clock is not working.")
            End If

            If dat.Hour > 17 Then
                Return "evening"
            Else If dat.Hour > 12 Then
                Return "afternoon"
            Else
                Return "morning"
            End If
        End Function)
        Dim c = t.ContinueWith(Sub(antecedent)
            If t.Status = TaskStatus.RanToCompletion Then
                Console.WriteLine("Good {0}!",
                    antecedent.Result)
                Console.WriteLine("And how are you this fine {0}?",
                    antecedent.Result)
            Else If t.Status = TaskStatus.Faulted Then
                Console.WriteLine(t.Exception.GetBaseException().Message)
            End If
        End Sub)
    End Sub
End Module
' The example displays output like the following:
'     Good afternoon!
'     And how are you this fine afternoon?

```

Cancel a continuation

The [Task.Status](#) property of a continuation is set to [TaskStatus.Canceled](#) in the following situations:

- It throws an [OperationCanceledException](#) exception in response to a cancellation request. As with any task, if the exception contains the same token that was passed to the continuation, it is treated as an acknowledgment of cooperative cancellation.
- The continuation is passed a [System.Threading.CancellationToken](#) whose [IsCancellationRequested](#) property is `true`. In this case, the continuation does not start, and it transitions to the [TaskStatus.Canceled](#) state.
- The continuation never runs because the condition set by its [TaskContinuationOptions](#) argument was not met. For example, if an antecedent goes into a [TaskStatus.Faulted](#) state, its continuation that was passed the [TaskContinuationOptions.NotOnFaulted](#) option will not run but will transition to the [Canceled](#) state.

If a task and its continuation represent two parts of the same logical operation, you can pass the same cancellation token to both tasks, as shown in the following example. It consists of an antecedent that generates a list of integers that are divisible by 33, which it passes to the continuation. The continuation in turn displays the list. Both the antecedent and the continuation pause regularly for random intervals. In addition, a [System.Threading.Timer](#) object is used to execute the `Elapsed` method after a five-second timeout interval. This example calls the [CancellationTokenSource.Cancel](#) method, which causes the currently executing task to call the [CancellationToken.ThrowIfCancellationRequested](#) method. Whether the [CancellationTokenSource.Cancel](#) method is called when the antecedent or its continuation is executing depends on the duration of the randomly generated pauses. If the antecedent is canceled, the continuation will not start. If the antecedent is not canceled, the token can still be used to cancel the continuation.

```
using System;
```

```

using System.Collections.Generic;
using System.Linq;
using System.Threading;
using System.Threading.Tasks;

public class CancellationExample
{
    static readonly Random s_random = new Random((int)DateTime.Now.Ticks);

    public static async Task Main()
    {
        using var cts = new CancellationTokenSource();
        CancellationToken token = cts.Token;
        var timer = new Timer(Elapsed, cts, 5000, Timeout.Infinite);

        var task = Task.Run(
            async () =>
            {
                var product33 = new List<int>();
                for (int index = 1; index < short.MaxValue; index++)
                {
                    if (token.IsCancellationRequested)
                    {
                        Console.WriteLine("\nCancellation requested in antecedent...\n");
                        token.ThrowIfCancellationRequested();
                    }
                    if (index % 2000 == 0)
                    {
                        int delay = s_random.Next(16, 501);
                        await Task.Delay(delay);
                    }
                    if (index % 33 == 0)
                    {
                        product33.Add(index);
                    }
                }

                return product33.ToArray();
            }, token);

        Task<double> continuation = task.ContinueWith(
            async antecedent =>
            {
                Console.WriteLine("Multiples of 33:\n");
                int[] array = antecedent.Result;
                for (int index = 0; index < array.Length; index++)
                {
                    if (token.IsCancellationRequested)
                    {
                        Console.WriteLine("\nCancellation requested in continuation...\n");
                        token.ThrowIfCancellationRequested();
                    }
                    if (index % 100 == 0)
                    {
                        int delay = s_random.Next(16, 251);
                        await Task.Delay(delay);
                    }

                    Console.Write($"{array[index]}:{N0}{{(index != array.Length - 1 ? ", " : "")}}");

                    if (Console.CursorLeft >= 74)
                    {
                        Console.WriteLine();
                    }
                }
                Console.WriteLine();
                return array.Average();
            }, token).Unwrap();
    }
}

```

```

try
{
    await task;
    double result = await continuation;
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
}

Console.WriteLine("\nAntecedent Status: {0}", task.Status);
Console.WriteLine("Continuation Status: {0}", continuation.Status);
}

static void Elapsed(object? state)
{
    if (state is CancellationTokenSource cts)
    {
        cts.Cancel();
        Console.WriteLine("\nCancellation request issued...\n");
    }
}

// The example displays the similar output:
// Multiples of 33:
//
//      33, 66, 99, 132, 165, 198, 231, 264, 297, 330, 363, 396, 429, 462, 495, 528,
//      561, 594, 627, 660, 693, 726, 759, 792, 825, 858, 891, 924, 957, 990, 1,023,
//      1,056, 1,089, 1,122, 1,155, 1,188, 1,221, 1,254, 1,287, 1,320, 1,353, 1,386,
//      1,419, 1,452, 1,485, 1,518, 1,551, 1,584, 1,617, 1,650, 1,683, 1,716, 1,749,
//      1,782, 1,815, 1,848, 1,881, 1,914, 1,947, 1,980, 2,013, 2,046, 2,079, 2,112,
//      2,145, 2,178, 2,211, 2,244, 2,277, 2,310, 2,343, 2,376, 2,409, 2,442, 2,475,
//      2,508, 2,541, 2,574, 2,607, 2,640, 2,673, 2,706, 2,739, 2,772, 2,805, 2,838,
//      2,871, 2,904, 2,937, 2,970, 3,003, 3,036, 3,069, 3,102, 3,135, 3,168, 3,201,
//      3,234, 3,267, 3,300, 3,333, 3,366, 3,399, 3,432, 3,465, 3,498, 3,531, 3,564,
//      3,597, 3,630, 3,663, 3,696, 3,729, 3,762, 3,795, 3,828, 3,861, 3,894, 3,927,
//      3,960, 3,993, 4,026, 4,059, 4,092, 4,125, 4,158, 4,191, 4,224, 4,257, 4,290,
//      4,323, 4,356, 4,389, 4,422, 4,455, 4,488, 4,521, 4,554, 4,587, 4,620, 4,653,
//      4,686, 4,719, 4,752, 4,785, 4,818, 4,851, 4,884, 4,917, 4,950, 4,983, 5,016,
//      5,049, 5,082, 5,115, 5,148, 5,181, 5,214, 5,247, 5,280, 5,313, 5,346, 5,379,
//      5,412, 5,445, 5,478, 5,511, 5,544, 5,577, 5,610, 5,643, 5,676, 5,709, 5,742,
//      Cancellation request issued...

//
//      5,775,
//      Cancellation requested in continuation...

//
//      The operation was canceled.

//
//      Antecedent Status: RanToCompletion
//      Continuation Status: Canceled

```

```
Imports System.Collections.Generic
Imports System.Threading
Imports System.Threading.Tasks

Module Example
    Public Sub Main()
        Dim rnd As New Random()
        Dim lockObj As New Object()
        Dim cts As New CancellationTokenSource()
        Dim token As CancellationToken = cts.Token
        Dim timer As New Timer(AddressOf Elapsed, cts, 5000, Timeout.Infinite)

        Dim t = Task.Run(Function()
                            Dim product33 As New List(Of Integer)()
                            For ctr As Integer = 1 To Int16.MaxValue
                                ' Check for cancellation.
                                If token.IsCancellationRequested Then
                                    Return
                                End If
                                product33.Add(rnd.Next())
                            Next
                            Return product33
                        End Function)
        timer.Start()
        Dim result = t.Result
        For Each item In result
            Console.WriteLine(item)
        Next
    End Sub
End Module
```

```

        If token.IsCancellationRequested Then
            Console.WriteLine("\nCancellation requested in antecedent...\n")
            token.ThrowIfCancellationRequested()
        End If
        ' Introduce a delay.
        If ctr Mod 2000 = 0 Then
            Dim delay As Integer
            SyncLock lockObj
                delay = rnd.Next(16, 501)
            End SyncLock
            Thread.Sleep(delay)
        End If

        ' Determine if this is a multiple of 33.
        If ctr Mod 33 = 0 Then product33.Add(ctr)
    Next
    Return product33.ToArray()
End Function, token)

Dim continuation = t.ContinueWith(Sub(antecedent)
    Console.WriteLine("Multiples of 33:" + vbCrLf)
    Dim arr = antecedent.Result
    For ctr As Integer = 0 To arr.Length - 1
        If token.IsCancellationRequested Then
            Console.WriteLine("{0}Cancellation requested in
continuation...{0}", vbCrLf)
            token.ThrowIfCancellationRequested()
        End If

        If ctr Mod 100 = 0 Then
            Dim delay As Integer
            SyncLock lockObj
                delay = rnd.Next(16, 251)
            End SyncLock
            Thread.Sleep(delay)
        End If
        Console.Write("{0:N0}{1}", arr(ctr),
                    If(ctr <> arr.Length - 1, ", ", ""))
        If Console.CursorLeft >= 74 Then Console.WriteLine()
    Next
    Console.WriteLine()
End Sub, token)

Try
    continuation.Wait()
Catch e As AggregateException
    For Each ie In e.InnerExceptions
        Console.WriteLine("{0}: {1}", ie.GetType().Name,
                        ie.Message)
    Next
Finally
    cts.Dispose()
End Try

Console.WriteLine(vbCrLf + "Antecedent Status: {0}", t.Status)
Console.WriteLine("Continuation Status: {0}", continuation.Status)
End Sub

Private Sub Elapsed(state As Object)
    Dim cts As CancellationTokenSource = TryCast(state, CancellationTokenSource)
    If cts Is Nothing Then return

    cts.Cancel()
    Console.WriteLine("{0}Cancellation request issued...{0}", vbCrLf)
End Sub
End Module
' The example displays output like the following:
'     Multiples of 33:

```

```
' 33, 66, 99, 132, 165, 198, 231, 264, 297, 330, 363, 396, 429, 462, 495, 528,
' 561, 594, 627, 660, 693, 726, 759, 792, 825, 858, 891, 924, 957, 990, 1,023,
' 1,056, 1,089, 1,122, 1,155, 1,188, 1,221, 1,254, 1,287, 1,320, 1,353, 1,386,
' 1,419, 1,452, 1,485, 1,518, 1,551, 1,584, 1,617, 1,650, 1,683, 1,716, 1,749,
' 1,782, 1,815, 1,848, 1,881, 1,914, 1,947, 1,980, 2,013, 2,046, 2,079, 2,112,
' 2,145, 2,178, 2,211, 2,244, 2,277, 2,310, 2,343, 2,376, 2,409, 2,442, 2,475,
' 2,508, 2,541, 2,574, 2,607, 2,640, 2,673, 2,706, 2,739, 2,772, 2,805, 2,838,
' 2,871, 2,904, 2,937, 2,970, 3,003, 3,036, 3,069, 3,102, 3,135, 3,168, 3,201,
' 3,234, 3,267, 3,300, 3,333, 3,366, 3,399, 3,432, 3,465, 3,498, 3,531, 3,564,
' 3,597, 3,630, 3,663, 3,696, 3,729, 3,762, 3,795, 3,828, 3,861, 3,894, 3,927,
' 3,960, 3,993, 4,026, 4,059, 4,092, 4,125, 4,158, 4,191, 4,224, 4,257, 4,290,
' 4,323, 4,356, 4,389, 4,422, 4,455, 4,488, 4,521, 4,554, 4,587, 4,620, 4,653,
' 4,686, 4,719, 4,752, 4,785, 4,818, 4,851, 4,884, 4,917, 4,950, 4,983, 5,016,
' 5,049, 5,082, 5,115, 5,148, 5,181, 5,214, 5,247, 5,280, 5,313, 5,346, 5,379,
' 5,412, 5,445, 5,478, 5,511, 5,544, 5,577, 5,610, 5,643, 5,676, 5,709, 5,742,
' 5,775, 5,808, 5,841, 5,874, 5,907, 5,940, 5,973, 6,006, 6,039, 6,072, 6,105,
' 6,138, 6,171, 6,204, 6,237, 6,270, 6,303, 6,336, 6,369, 6,402, 6,435, 6,468,
' 6,501, 6,534, 6,567, 6,600, 6,633, 6,666, 6,699, 6,732, 6,765, 6,798, 6,831,
' 6,864, 6,897, 6,930, 6,963, 6,996, 7,029, 7,062, 7,095, 7,128, 7,161, 7,194,
' 7,227, 7,260, 7,293, 7,326, 7,359, 7,392, 7,425, 7,458, 7,491, 7,524, 7,557,
' 7,590, 7,623, 7,656, 7,689, 7,722, 7,755, 7,788, 7,821, 7,854, 7,887, 7,920,
' 7,953, 7,986, 8,019, 8,052, 8,085, 8,118, 8,151, 8,184, 8,217, 8,250, 8,283,
' 8,316, 8,349, 8,382, 8,415, 8,448, 8,481, 8,514, 8,547, 8,580, 8,613, 8,646,
' 8,679, 8,712, 8,745, 8,778, 8,811, 8,844, 8,877, 8,910, 8,943, 8,976, 9,009,
' 9,042, 9,075, 9,108, 9,141, 9,174, 9,207, 9,240, 9,273, 9,306, 9,339, 9,372,
' 9,405, 9,438, 9,471, 9,504, 9,537, 9,570, 9,603, 9,636, 9,669, 9,702, 9,735,
' 9,768, 9,801, 9,834, 9,867, 9,900, 9,933, 9,966, 9,999, 10,032, 10,065, 10,098,
' 10,131, 10,164, 10,197, 10,230, 10,263, 10,296, 10,329, 10,362, 10,395, 10,428,
' 10,461, 10,494, 10,527, 10,560, 10,593, 10,626, 10,659, 10,692, 10,725, 10,758,
' 10,791, 10,824, 10,857, 10,890, 10,923, 10,956, 10,989, 11,022, 11,055, 11,088,
' 11,121, 11,154, 11,187, 11,220, 11,253, 11,286, 11,319, 11,352, 11,385, 11,418,
' 11,451, 11,484, 11,517, 11,550, 11,583, 11,616, 11,649, 11,682, 11,715, 11,748,
' 11,781, 11,814, 11,847, 11,880, 11,913, 11,946, 11,979, 12,012, 12,045, 12,078,
' 12,111, 12,144, 12,177, 12,210, 12,243, 12,276, 12,309, 12,342, 12,375, 12,408,
' 12,441, 12,474, 12,507, 12,540, 12,573, 12,606, 12,639, 12,672, 12,705, 12,738,
' 12,771, 12,804, 12,837, 12,870, 12,903, 12,936, 12,969, 13,002, 13,035, 13,068,
' 13,101, 13,134, 13,167, 13,200, 13,233, 13,266,
' Cancellation requested in continuation...
```

```
' Cancellation request issued...
```

```
' TaskCanceledException: A task was canceled.
```

```
' Antecedent Status: RanToCompletion
```

```
' Continuation Status: Canceled
```

You can also prevent a continuation from executing if its antecedent is canceled without supplying the continuation a cancellation token by specifying the [TaskContinuationOptions.NotOnCanceled](#) option when you create the continuation. The following is a simple example.

```
using System;
using System.Threading;
using System.Threading.Tasks;

public class CancellationTwoExample
{
    public static async Task Main()
    {
        using var cts = new CancellationTokenSource();
        CancellationToken token = cts.Token;
        cts.Cancel();

        var task = Task.FromCanceled(token);
        Task continuation =
            task.ContinueWith(
                antecedent => Console.WriteLine("The continuation is running."),
                TaskContinuationOptions.NotOnCanceled);

        try
        {
            await task;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"{ex.GetType().Name}: {ex.Message}");
            Console.WriteLine();
        }

        Console.WriteLine($"Task {task.Id}: {task.Status:G}");
        Console.WriteLine($"Task {continuation.Id}: {continuation.Status:G}");
    }
}

// The example displays the similar output:
//     TaskCanceledException: A task was canceled.
//
//     Task 1: Canceled
//     Task 2: Canceled
```

```

Imports System.Threading
Imports System.Threading.Tasks

Module Example
    Public Sub Main()
        Dim cts As New CancellationTokenSource()
        Dim token As CancellationToken = cts.Token
        cts.Cancel()

        Dim t As Task = Task.FromCanceled(token)
        Dim continuation As Task = t.ContinueWith(Sub(antecedent)
                                                    Console.WriteLine("The continuation is running."))
                                                End Sub, TaskContinuationOptions.NotOnCanceled)

        Try
            t.Wait()
        Catch e As AggregateException
            For Each ie In e.InnerExceptions
                Console.WriteLine("{0}: {1}", ie.GetType().Name, ie.Message)
            Next
            Console.WriteLine()
        Finally
            cts.Dispose()
        End Try

        Console.WriteLine("Task {0}: {1:G}", t.Id, t.Status)
        Console.WriteLine("Task {0}: {1:G}", continuation.Id,
                         continuation.Status)
    End Sub
End Module
' The example displays the following output:
'     TaskCanceledException: A task was canceled.
'
'     Task 1: Canceled
'     Task 2: Canceled

```

After a continuation goes into the [Canceled](#) state, it may affect continuations that follow, depending on the [TaskContinuationOptions](#) that were specified for those continuations.

Continuations that are disposed will not start.

Continuations and child tasks

A continuation does not run until the antecedent and all of its attached child tasks have completed. The continuation does not wait for detached child tasks to finish. The following two examples illustrate child tasks that are attached to and detached from an antecedent that creates a continuation. In the following example, the continuation runs only after all child tasks have completed, and running the example multiple times produces identical output each time. The example launches the antecedent by calling the [TaskFactory.StartNew](#) method, since by default the [Task.Run](#) method creates a parent task whose default task creation option is [TaskCreationOptions.DenyChildAttach](#).

```
using System;
using System.Threading.Tasks;

public class AttachedExample
{
    public static async Task Main()
    {
        await Task.Factory
            .StartNew(
                () =>
            {
                Console.WriteLine($"Running antecedent task {Task.CurrentId}...");
                Console.WriteLine("Launching attached child tasks...");
                for (int ctr = 1; ctr <= 5; ctr++)
                {
                    int index = ctr;
                    Task.Factory.StartNew(async value =>
                    {
                        Console.WriteLine($"    Attached child task #{value} running");
                        await Task.Delay(1000);
                    }, index, TaskCreationOptions.AttachedToParent);
                }
                Console.WriteLine("Finished launching attached child tasks...");
            }).ContinueWith(
                antecedent =>
                Console.WriteLine($"Executing continuation of Task {antecedent.Id}"));
    }
}

// The example displays the similar output:
//     Running antecedent task 1...
//     Launching attached child tasks...
//     Finished launching attached child tasks...
//         Attached child task #1 running
//         Attached child task #5 running
//         Attached child task #3 running
//         Attached child task #2 running
//         Attached child task #4 running
//     Executing continuation of Task 1
```

```

Imports System.Threading
Imports System.Threading.Tasks

Public Module Example
    Public Sub Main()
        Dim t = Task.Factory.StartNew(Sub()
            Console.WriteLine("Running antecedent task {0}...",
                Task.CurrentId)
            Console.WriteLine("Launching attached child tasks...")
            For ctr As Integer = 1 To 5
                Dim index As Integer = ctr
                Task.Factory.StartNew(Sub(value)
                    Console.WriteLine(" Attached child
task #{0} running",
                        value)
                    Thread.Sleep(1000)
                End Sub, index,
                TaskCreationOptions.AttachedToParent)
            Next
            Console.WriteLine("Finished launching attached child tasks...")
        End Sub)
        Dim continuation = t.ContinueWith(Sub(antecedent)
            Console.WriteLine("Executing continuation of Task {0}",
                antecedent.Id)
        End Sub)
        continuation.Wait()
    End Sub
End Module
' The example displays the following output:
'   Running antecedent task 1...
'   Launching attached child tasks...
'   Finished launching attached child tasks...
'       Attached child task #5 running
'       Attached child task #1 running
'       Attached child task #2 running
'       Attached child task #3 running
'       Attached child task #4 running
'   Executing continuation of Task 1

```

If child tasks are detached from the antecedent, however, the continuation runs as soon as the antecedent has terminated, regardless of the state of the child tasks. As a result, multiple runs of the following example can produce variable output that depends on how the task scheduler handled each child task.

```
using System;
using System.Threading.Tasks;

public class DetachedExample
{
    public static async Task Main()
    {
        Task task =
            Task.Factory.StartNew(
                () =>
                {
                    Console.WriteLine($"Running antecedent task {Task.CurrentId}...");
                    Console.WriteLine("Launching attached child tasks...");
                    for (int ctr = 1; ctr <= 5; ctr++)
                    {
                        int index = ctr;
                        Task.Factory.StartNew(
                            async value =>
                            {
                                Console.WriteLine($"    Attached child task #{value} running");
                                await Task.Delay(1000);
                            }, index);
                    }
                    Console.WriteLine("Finished launching detached child tasks...");
                }, TaskCreationOptions.DenyChildAttach);

        Task continuation =
            task.ContinueWith(
                antecedent =>
                Console.WriteLine($"Executing continuation of Task {antecedent.Id}"));

        await continuation;

        Console.ReadLine();
    }
}

// The example displays the similar output:
// Running antecedent task 1...
// Launching attached child tasks...
// Finished launching detached child tasks...
// Executing continuation of Task 1
//     Attached child task #1 running
//     Attached child task #5 running
//     Attached child task #2 running
//     Attached child task #3 running
//     Attached child task #4 running
```

```

Imports System.Threading
Imports System.Threading.Tasks

Public Module Example
    Public Sub Main()
        Dim t = Task.Factory.StartNew(Sub()
            Console.WriteLine("Running antecedent task {0}...", 
                Task.CurrentId)
            Console.WriteLine("Launching attached child tasks...")
            For ctr As Integer = 1 To 5
                Dim index As Integer = ctr
                Task.Factory.StartNew(Sub(value)
                    Console.WriteLine(" Attached child
task #{0} running",
                        value)
                    Thread.Sleep(1000)
                End Sub, index)
            Next
            Console.WriteLine("Finished launching detached child tasks...")
        End Sub, TaskCreationOptions.DenyChildAttach)

        Dim continuation = t.ContinueWith(Sub(antecedent)
            Console.WriteLine("Executing continuation of Task {0}",
                antecedent.Id)
        End Sub)

        continuation.Wait()
    End Sub
End Module

' The example displays output like the following:
'   Running antecedent task 1...
'   Launching attached child tasks...
'   Finished launching detached child tasks...
'       Attached child task #1 running
'       Attached child task #2 running
'       Attached child task #5 running
'       Attached child task #3 running
'   Executing continuation of Task 1
'       Attached child task #4 running

```

The final status of the antecedent task depends on the final status of any attached child tasks. The status of detached child tasks does not affect the parent. For more information, see [Attached and Detached Child Tasks](#).

Associate state with continuations

You can associate arbitrary state with a task continuation. The [ContinueWith](#) method provides overloaded versions that each take an [Object](#) value that represents the state of the continuation. You can later access this state object by using the [Task.AsyncState](#) property. This state object is `null` if you do not provide a value.

Continuation state is useful when you convert existing code that uses the [Asynchronous Programming Model \(APM\)](#) to use the TPL. In the APM, you typically provide object state in the [BeginMethod](#) method and later access that state by using the [IAsyncResult.AsyncState](#) property. By using the [ContinueWith](#) method, you can preserve this state when you convert code that uses the APM to use the TPL.

Continuation state can also be useful when you work with [Task](#) objects in the Visual Studio debugger. For example, in the **Parallel Tasks** window, the **Task** column displays the string representation of the state object for each task. For more information about the **Parallel Tasks** window, see [Using the Tasks Window](#).

The following example shows how to use continuation state. It creates a chain of continuation tasks. Each task provides the current time, a [DateTime](#) object, for the `state` parameter of the [ContinueWith](#) method. Each [DateTime](#) object represents the time at which the continuation task is created. Each task produces as its result a second [DateTime](#) object that represents the time at which the task finishes. After all tasks finish, this example displays the creation time and the time at which each continuation task finishes.

```
using System;
using System.Collections.Generic;
using System.Threading;
using System.Threading.Tasks;

class ContinuationStateExample
{
    static DateTime DoWork()
    {
        Thread.Sleep(2000);

        return DateTime.Now;
    }

    static async Task Main()
    {
        Task<DateTime> task = Task.Run(() => DoWork());

        var continuations = new List<Task<DateTime>>();
        for (int i = 0; i < 5; i++)
        {
            task = task.ContinueWith((antecedent, _) => DoWork(), DateTime.Now);
            continuations.Add(task);
        }

        await task;

        foreach (Task<DateTime> continuation in continuations)
        {
            DateTime start = (DateTime)continuation.AsyncState!;
            DateTime end = continuation.Result;

            Console.WriteLine($"Task was created at {start.TimeOfDay} and finished at {end.TimeOfDay}.");
        }

        Console.ReadLine();
    }
}

// The example displays the similar output:
//      Task was created at 10:56:21.1561762 and finished at 10:56:25.1672062.
//      Task was created at 10:56:21.1610677 and finished at 10:56:27.1707646.
//      Task was created at 10:56:21.1610677 and finished at 10:56:29.1743230.
//      Task was created at 10:56:21.1610677 and finished at 10:56:31.1779883.
//      Task was created at 10:56:21.1610677 and finished at 10:56:33.1837083.
```

```

Imports System.Collections.Generic
Imports System.Threading
Imports System.Threading.Tasks

' Demonstrates how to associate state with task continuations.
Public Module ContinuationState
    ' Simulates a lengthy operation and returns the time at which
    ' the operation completed.
    Public Function DoWork() As Date
        ' Simulate work by suspending the current thread
        ' for two seconds.
        Thread.Sleep(2000)

        ' Return the current time.
        Return Date.Now
    End Function

    Public Sub Main()
        ' Start a root task that performs work.
        Dim t As Task(Of Date) = Task(Of Date).Run(Function() DoWork())

        ' Create a chain of continuation tasks, where each task is
        ' followed by another task that performs work.
        Dim continuations As New List(Of Task(Of DateTime))()
        For i As Integer = 0 To 4
            ' Provide the current time as the state of the continuation.
            t = t.ContinueWith(Function(antecedent, state) DoWork(), DateTime.Now)
            continuations.Add(t)
        Next

        ' Wait for the last task in the chain to complete.
        t.Wait()

        ' Display the creation time of each continuation (the state object)
        ' and the completion time (the result of that task) to the console.
        For Each continuation In continuations
            Dim start As DateTime = CDate(continuation.AsyncState)
            Dim [end] As DateTime = continuation.Result

            Console.WriteLine("Task was created at {0} and finished at {1}.",
                start.TimeOfDay, [end].TimeOfDay)
        Next
    End Sub
End Module
' The example displays output like the following:
'     Task was created at 10:56:21.1561762 and finished at 10:56:25.1672062.
'     Task was created at 10:56:21.1610677 and finished at 10:56:27.1707646.
'     Task was created at 10:56:21.1610677 and finished at 10:56:29.1743230.
'     Task was created at 10:56:21.1610677 and finished at 10:56:31.1779883.
'     Task was created at 10:56:21.1610677 and finished at 10:56:33.1837083.

```

Continuations that return Task types

Sometimes you may need to chain a continuation that returns a [Task](#) type. These are referred to as nested tasks, and they are common. When a parent task calls [Task<TResult>.ContinueWith](#), and provides a `continuationFunction` that is task returning you call [Unwrap](#) to create a proxy task that represents the asynchronous operation of the `<Task<Task<T>>` or `Task(Of Task(Of T))` (Visual Basic).

The following example shows how to use continuations that wrap additional task returning functions. Each continuation can be unwrapped, exposing the inner task that was wrapped.

```

using System;
using System.Threading;
using System.Threading.Tasks;

public class UnwrapExample
{
    public static async Task Main()
    {
        Task<int> taskOne = RemoteIncrement(0);
        Console.WriteLine("Started RemoteIncrement(0)");

        Task<int> taskTwo = RemoteIncrement(4)
            .ContinueWith(t => RemoteIncrement(t.Result))
            .Unwrap().ContinueWith(t => RemoteIncrement(t.Result))
            .Unwrap().ContinueWith(t => RemoteIncrement(t.Result))
            .Unwrap();

        Console.WriteLine("Started RemoteIncrement(...(RemoteIncrement(RemoteIncrement(4))..."));

        try
        {
            await taskOne;
            Console.WriteLine("Finished RemoteIncrement(0)");

            await taskTwo;
            Console.WriteLine("Finished RemoteIncrement(...(RemoteIncrement(RemoteIncrement(4))..."));
        }
        catch (Exception e)
        {
            Console.WriteLine($"A task has thrown the following (unexpected) exception:\n{e}");
        }
    }

    static Task<int> RemoteIncrement(int number) =>
        Task<int>.Factory.StartNew(
            obj =>
            {
                Thread.Sleep(1000);

                int x = (int)(obj!);
                Console.WriteLine("Thread={0}, Next={1}", Thread.CurrentThread.ManagedThreadId, ++x);
                return x;
            },
            number);
    }

    // The example displays the similar output:
    //     Started RemoteIncrement(0)
    //     Started RemoteIncrement(...(RemoteIncrement(RemoteIncrement(4))...))
    //     Thread=4, Next=1
    //     Finished RemoteIncrement(0)
    //     Thread=5, Next=5
    //     Thread=6, Next=6
    //     Thread=6, Next=7
    //     Thread=6, Next=8
    //     Finished RemoteIncrement(...(RemoteIncrement(RemoteIncrement(4))...)
}

```

```

Imports System.Threading

Module UnwrapExample
    Sub Main()
        Dim taskOne As Task(Of Integer) = RemoteIncrement(0)
        Console.WriteLine("Started RemoteIncrement(0)")

        Dim taskTwo As Task(Of Integer) = RemoteIncrement(4).
            ContinueWith(Function(t) RemoteIncrement(t.Result)).
            Unwrap().ContinueWith(Function(t) RemoteIncrement(t.Result)).
            Unwrap().ContinueWith(Function(t) RemoteIncrement(t.Result)).
            Unwrap()

        Console.WriteLine("Started RemoteIncrement(...(RemoteIncrement(RemoteIncrement(4))..."))

        Try
            taskOne.Wait()
            Console.WriteLine("Finished RemoteIncrement(0)")

            taskTwo.Wait()
            Console.WriteLine("Finished RemoteIncrement(...(RemoteIncrement(RemoteIncrement(4))..."))
        Catch e As AggregateException
            Console.WriteLine($"A task has thrown the following (unexpected) exception:{vbLf}{e}")
        End Try
    End Sub

    Function RemoteIncrement(ByVal number As Integer) As Task(Of Integer)
        Return Task(Of Integer).Factory.StartNew(
            Function(obj)
                Thread.Sleep(1000)

                Dim x As Integer = CInt(obj)
                Console.WriteLine("Thread={0}, Next={1}", Thread.CurrentThread.ManagedThreadId,
Interlocked.Increment(x))
                Return x
            End Function, number)
    End Function
End Module

' The example displays the similar output:
' Started RemoteIncrement(0)
' Started RemoteIncrement(...(RemoteIncrement(RemoteIncrement(4))...
' Thread=4, Next=1
' Finished RemoteIncrement(0)
' Thread=5, Next=5
' Thread=6, Next=6
' Thread=6, Next=7
' Thread=6, Next=8
' Finished RemoteIncrement(...(RemoteIncrement(RemoteIncrement(4))...

```

For more information on using [Unwrap](#), see [How to: Unwrap a nested Task](#).

Handle exceptions thrown from continuations

An antecedent-continuation relationship is not a parent-child relationship. Exceptions thrown by continuations are not propagated to the antecedent. Therefore, handle exceptions thrown by continuations as you would handle them in any other task, as follows:

- You can use the [Wait](#), [WaitAll](#), or [WaitAny](#) method, or its generic counterpart, to wait on the continuation. You can wait for an antecedent and its continuations in the same `try` statement, as shown in the following example.

```
using System;
using System.Threading.Tasks;

public class ExceptionExample
{
    public static async Task Main()
    {
        Task<int> task = Task.Run(
            () =>
            {
                Console.WriteLine($"Executing task {Task.CurrentId}");
                return 54;
            });

        var continuation = task.ContinueWith(
            antecedent =>
            {
                Console.WriteLine($"Executing continuation task {Task.CurrentId}");
                Console.WriteLine($"Value from antecedent: {antecedent.Result}");

                throw new InvalidOperationException();
            });

        try
        {
            await task;
            await continuation;
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
        }
    }
}

// The example displays the similar output:
//      Executing task 1
//      Executing continuation task 2
//      Value from antecedent: 54
//      Operation is not valid due to the current state of the object.
```

```

Imports System.Threading.Tasks

Module Example
    Public Sub Main()
        Dim task1 = Task(Of Integer).Run(Function()
            Console.WriteLine("Executing task {0}",
                Task.CurrentId)
            Return 54
        End Function)

        Dim continuation = task1.ContinueWith(Sub(antecedent)
            Console.WriteLine("Executing continuation task {0}",
                Task.CurrentId)
            Console.WriteLine("Value from antecedent: {0}",
                antecedent.Result)
            Throw New InvalidOperationException()
        End Sub)

        Try
            task1.Wait()
            continuation.Wait()
        Catch ae As AggregateException
            For Each ex In ae.InnerExceptions
                Console.WriteLine(ex.Message)
            Next
        End Try
    End Sub
End Module

' The example displays the following output:
'   Executing task 1
'   Executing continuation task 2
'   Value from antecedent: 54
'   Operation is not valid due to the current state of the object.

```

- You can use a second continuation to observe the [Exception](#) property of the first continuation. In the following example, a task attempts to read from a non-existent file. The continuation then displays information about the exception in the antecedent task.

```

using System;
using System.IO;
using System.Linq;
using System.Threading.Tasks;

public class ExceptionTwoExample
{
    public static async Task Main()
    {
        var task = Task.Run(
            () =>
            {
                string fileText = File.ReadAllText(@"C:\NonexistentFile.txt");
                return fileText;
            });
    }

    Task continuation = task.ContinueWith(
        antecedent =>
    {
        var fileNotFound =
            antecedent.Exception
                ?.InnerExceptions
                ?.FirstOrDefault(e => e is FileNotFoundException) as FileNotFoundException;

        if (fileNotFound != null)
        {
            Console.WriteLine(fileNotFound.Message);
        }
    }, TaskContinuationOptions.OnlyOnFaulted);

    await continuation;

    Console.ReadLine();
}
}

// The example displays the following output:
// Could not find file 'C:\NonexistentFile.txt'.

```

```

Imports System.IO
Imports System.Threading.Tasks

Module Example
    Public Sub Main()
        Dim t = Task.Run(Function()
                            Dim s As String = File.ReadAllText("C:\NonexistentFile.txt")
                            Return s
                        End Function)

        Dim c = t.ContinueWith(Sub(antecedent)
                                ' Get the antecedent's exception information.
                                For Each ex In antecedent.Exception.InnerException
                                    If TypeOf ex Is FileNotFoundException
                                        Console.WriteLine(ex.Message)
                                    End If
                                Next
                            End Sub, TaskContinuationOptions.OnlyOnFaulted)

        c.Wait()
    End Sub
End Module
'
```

' The example displays the following output:
' Could not find file 'C:\NonexistentFile.txt'.

Because it was run with the [TaskContinuationOptions.OnlyOnFaulted](#) option, the continuation executes only if an

exception occurs in the antecedent, and therefore it can assume that the antecedent's [Exception](#) property is not `null`. If the continuation executes whether or not an exception is thrown in the antecedent, it would have to check whether the antecedent's [Exception](#) property is not `null` before attempting to handle the exception, as the following code fragment shows.

```
var fileNotFound =
    antecedent.Exception
        ?.InnerExceptions
        ?.FirstOrDefault(e => e is FileNotFoundException) as FileNotFoundException;

if (fileNotFound != null)
{
    Console.WriteLine(fileNotFound.Message);
}
```

```
' Determine whether an exception occurred.
If antecedent.Exception IsNot Nothing Then
    ' Get the antecedent's exception information.
    For Each ex In antecedent.Exception.InnerExceptions
        If TypeOf ex Is FileNotFoundException
            Console.WriteLine(ex.Message)
        End If
    Next
End If
```

For more information, see [Exception Handling](#).

- If the continuation is an attached child task that was created by using the [TaskContinuationOptions.AttachedToParent](#) option, its exceptions will be propagated by the parent back to the calling thread, as is the case in any other attached child. For more information, see [Attached and Detached Child Tasks](#).

See also

- [Task Parallel Library \(TPL\)](#)

Attached and Detached Child Tasks

9/20/2022 • 8 minutes to read • [Edit Online](#)

A *child task* (or *nested task*) is a [System.Threading.Tasks.Task](#) instance that is created in the user delegate of another task, which is known as the *parent task*. A child task can be either detached or attached. A *detached child task* is a task that executes independently of its parent. An *attached child task* is a nested task that is created with the [TaskCreationOptions.AttachedToParent](#) option whose parent does not explicitly or by default prohibit it from being attached. A task may create any number of attached and detached child tasks, limited only by system resources.

The following table lists the basic differences between the two kinds of child tasks.

CATEGORY	DETACHED CHILD TASKS	ATTACHED CHILD TASKS
Parent waits for child tasks to complete.	No	Yes
Parent propagates exceptions thrown by child tasks.	No	Yes
Status of parent depends on status of child.	No	Yes

In most scenarios, we recommend that you use detached child tasks, because their relationships with other tasks are less complex. That is why tasks created inside parent tasks are detached by default, and you must explicitly specify the [TaskCreationOptions.AttachedToParent](#) option to create an attached child task.

Detached child tasks

Although a child task is created by a parent task, by default it is independent of the parent task. In the following example, a parent task creates one simple child task. If you run the example code multiple times, you may notice that the output from the example differs from that shown, and also that the output may change each time you run the code. This occurs because the parent task and child tasks execute independently of each other; the child is a detached task. The example waits only for the parent task to complete, and the child task may not execute or complete before the console app terminates.

```

using System;
using System.Threading;
using System.Threading.Tasks;

public class Example
{
    public static void Main()
    {
        var parent = Task.Factory.StartNew(() => {
            Console.WriteLine("Outer task executing.");

            var child = Task.Factory.StartNew(() => {
                Console.WriteLine("Nested task starting.");
                Thread.Sleep(50000);
                Console.WriteLine("Nested task completing.");
            });
        });

        parent.Wait();
        Console.WriteLine("Outer has completed.");
    }
}

// The example produces output like the following:
//      Outer task executing.
//      Nested task starting.
//      Outer has completed.
//      Nested task completing.

```

```

Imports System.Threading
Imports System.Threading.Tasks

Module Example
    Public Sub Main()
        Dim parent = Task.Factory.StartNew(Sub()
            Console.WriteLine("Outer task executing.")
            Dim child = Task.Factory.StartNew(Sub()
                Console.WriteLine("Nested task starting.")
                Thread.Sleep(50000)

                Console.WriteLine("Nested task completing.")
            End Sub)
            parent.Wait()
            Console.WriteLine("Outer task has completed.")
        End Sub)
    End Module

    ' The example produces output like the following:
    '      Outer task executing.
    '      Nested task starting.
    '      Outer task has completed.
    '      Nested task completing.

```

If the child task is represented by a [Task<TResult>](#) object rather than a [Task](#) object, you can ensure that the parent task will wait for the child to complete by accessing the [Task<TResult>.Result](#) property of the child even if it is a detached child task. The [Result](#) property blocks until its task completes, as the following example shows.

```

using System;
using System.Threading;
using System.Threading.Tasks;

class Example
{
    static void Main()
    {
        var outer = Task<int>.Factory.StartNew(() => {
            Console.WriteLine("Outer task executing.");

            var nested = Task<int>.Factory.StartNew(() => {
                Console.WriteLine("Nested task starting.");
                Thread.Sleep(5000000);
                Console.WriteLine("Nested task completing.");
                return 42;
            });

            // Parent will wait for this detached child.
            return nested.Result;
        });

        Console.WriteLine("Outer has returned {0}.", outer.Result);
    }
}

// The example displays the following output:
//      Outer task executing.
//      Nested task starting.
//      Nested task completing.
//      Outer has returned 42.

```

```

Imports System.Threading
Imports System.Threading.Tasks

Module Example
    Public Sub Main()
        Dim parent = Task(Of Integer).Factory.StartNew(Function()
            Console.WriteLine("Outer task executing.")
            Dim child = Task(Of
Integer).Factory.StartNew(Function()

Console.WriteLine("Nested task starting.")

Thread.Sleep(5000000)

Console.WriteLine("Nested task completing.")

Return 42
End Function)
        Return child.Result
    End Function

    End Sub
End Module

' The example displays the following output:
'      Outer task executing.
'      Nested task starting.
'      Detached task completing.
'      Outer has returned 42

```

Attached child tasks

Unlike detached child tasks, attached child tasks are closely synchronized with the parent. You can change the detached child task in the previous example to an attached child task by using the [TaskCreationOptions.AttachedToParent](#) option in the task creation statement, as shown in the following example. In this code, the attached child task completes before its parent. As a result, the output from the example is the same each time you run the code.

```
using System;
using System.Threading;
using System.Threading.Tasks;

public class Example
{
    public static void Main()
    {
        var parent = Task.Factory.StartNew(() => {
            Console.WriteLine("Parent task executing.");
            var child = Task.Factory.StartNew(() => {
                Console.WriteLine("Attached child starting.");
                Thread.Sleep(5000000);
                Console.WriteLine("Attached child completing.");
            }, TaskCreationOptions.AttachedToParent);
        });
        parent.Wait();
        Console.WriteLine("Parent has completed.");
    }
}
// The example displays the following output:
//      Parent task executing.
//      Attached child starting.
//      Attached child completing.
//      Parent has completed.
```

```
Imports System.Threading
Imports System.Threading.Tasks

Module Example
    Public Sub Main()
        Dim parent = Task.Factory.StartNew(Sub()
            Console.WriteLine("Parent task executing")
            Dim child = Task.Factory.StartNew(Sub()

                Console.WriteLine("Attached child starting.")

                Thread.Sleep(5000000)

                Console.WriteLine("Attached child completing.")
            End Sub,
            TaskCreationOptions.AttachedToParent)
        End Sub)
        parent.Wait()
        Console.WriteLine("Parent has completed.")
    End Sub
End Module
' The example displays the following output:
'      Parent task executing.
'      Attached child starting.
'      Attached child completing.
'      Parent has completed.
```

You can use attached child tasks to create tightly synchronized graphs of asynchronous operations.

However, a child task can attach to its parent only if its parent does not prohibit attached child tasks. Parent tasks can explicitly prevent child tasks from attaching to them by specifying the [TaskCreationOptions.DenyChildAttach](#) option in the parent task's class constructor or the [TaskFactory.StartNew](#) method. Parent tasks implicitly prevent child tasks from attaching to them if they are created by calling the [Task.Run](#) method. The following example illustrates this. It is identical to the previous example, except that the parent task is created by calling the [Task.Run\(Action\)](#) method rather than the [TaskFactory.StartNew\(Action\)](#) method. Because the child task is not able to attach to its parent, the output from the example is unpredictable. Because the default task creation options for the [Task.Run](#) overloads include [TaskCreationOptions.DenyChildAttach](#), this example is functionally equivalent to the first example in the "Detached child tasks" section.

```
using System;
using System.Threading;
using System.Threading.Tasks;

public class Example
{
    public static void Main()
    {
        var parent = Task.Run(() => {
            Console.WriteLine("Parent task executing.");
            var child = Task.Factory.StartNew(() => {
                Console.WriteLine("Attached child starting.");
                Thread.Sleep(5000000);
                Console.WriteLine("Attached child completing.");
            }, TaskCreationOptions.AttachedToParent);
        });
        parent.Wait();
        Console.WriteLine("Parent has completed.");
    }
}
// The example displays output like the following:
//      Parent task executing.
//      Parent has completed.
//      Attached child starting.
```

```
Imports System.Threading
Imports System.Threading.Tasks

Module Example
    Public Sub Main()
        Dim parent = Task.Run(Sub()
            Console.WriteLine("Parent task executing.")
            Dim child = Task.Factory.StartNew(Sub()
                Console.WriteLine("Attached child
starting.")
                Thread.Sleep(5000000)
                Console.WriteLine("Attached child
completing.")
            End Sub,
            TaskCreationOptions.AttachedToParent)
        End Sub)
        parent.Wait()
        Console.WriteLine("Parent has completed.")
    End Sub
End Module
' The example displays output like the following:
'      Parent task executing.
'      Parent has completed.
'      Attached child starting.
```

Exceptions in child tasks

If a detached child task throws an exception, that exception must be observed or handled directly in the parent task just as with any non-nested task. If an attached child task throws an exception, the exception is automatically propagated to the parent task and back to the thread that waits or tries to access the task's [Task<TResult>.Result](#) property. Therefore, by using attached child tasks, you can handle all exceptions at just one point in the call to [Task.Wait](#) on the calling thread. For more information, see [Exception Handling](#).

Cancellation and child tasks

Task cancellation is cooperative. That is, to be cancelable, every attached or detached child task must monitor the status of the cancellation token. If you want to cancel a parent and all its children by using one cancellation request, you pass the same token as an argument to all tasks and provide in each task the logic to respond to the request in each task. For more information, see [Task Cancellation](#) and [How to: Cancel a Task and Its Children](#).

When the parent cancels

If a parent cancels itself before its child task is started, the child never starts. If a parent cancels itself after its child task has already started, the child runs to completion unless it has its own cancellation logic. For more information, see [Task Cancellation](#).

When a detached child task cancels

If a detached child task cancels itself by using the same token that was passed to the parent, and the parent does not wait for the child task, no exception is propagated, because the exception is treated as benign cooperation cancellation. This behavior is the same as that of any top-level task.

When an attached child task cancels

When an attached child task cancels itself by using the same token that was passed to its parent task, a [TaskCanceledException](#) is propagated to the joining thread inside an [AggregateException](#). You must wait for the parent task so that you can handle all benign exceptions in addition to all faulting exceptions that are propagated up through a graph of attached child tasks.

For more information, see [Exception Handling](#).

Preventing a child task from attaching to its parent

An unhandled exception that is thrown by a child task is propagated to the parent task. You can use this behavior to observe all child task exceptions from one root task instead of traversing a tree of tasks. However, exception propagation can be problematic when a parent task does not expect attachment from other code. For example, consider an app that calls a third-party library component from a [Task](#) object. If the third-party library component also creates a [Task](#) object and specifies [TaskCreationOptions.AttachedToParent](#) to attach it to the parent task, any unhandled exceptions that occur in the child task propagate to the parent. This could lead to unexpected behavior in the main app.

To prevent a child task from attaching to its parent task, specify the [TaskCreationOptions.DenyChildAttach](#) option when you create the parent [Task](#) or [Task<TResult>](#) object. When a task tries to attach to its parent and the parent specifies the [TaskCreationOptions.DenyChildAttach](#) option, the child task will not be able to attach to a parent and will execute just as if the [TaskCreationOptions.AttachedToParent](#) option was not specified.

You might also want to prevent a child task from attaching to its parent when the child task does not finish in a timely manner. Because a parent task does not finish until all child tasks finish, a long-running child task can cause the overall app to perform poorly. For an example that shows how to improve app performance by preventing a task from attaching to its parent task, see [How to: Prevent a Child Task from Attaching to its Parent](#).

See also

- [Parallel Programming](#)

- Data Parallelism

Task cancellation

9/20/2022 • 3 minutes to read • [Edit Online](#)

The [System.Threading.Tasks.Task](#) and [System.Threading.Tasks.Task<TResult>](#) classes support cancellation by using cancellation tokens. For more information, see [Cancellation in Managed Threads](#). In the Task classes, cancellation involves cooperation between the user delegate, which represents a cancelable operation, and the code that requested the cancellation. A successful cancellation involves the requesting code calling the [CancellationTokenSource.Cancel](#) method and the user delegate terminating the operation in a timely manner. You can terminate the operation by using one of these options:

- By returning from the delegate. In many scenarios, this option is sufficient. However, a task instance that's canceled in this way transitions to the [TaskStatus.RanToCompletion](#) state, not to the [TaskStatus.Canceled](#) state.
- By throwing an [OperationCanceledException](#) and passing it the token on which cancellation was requested. The preferred way to perform is to use the [ThrowIfCancellationRequested](#) method. A task that's canceled in this way transitions to the Canceled state, which the calling code can use to verify that the task responded to its cancellation request.

The following example shows the basic pattern for task cancellation that throws the exception:

NOTE

The token is passed to the user delegate and the task instance.

```
using System;
using System.Threading;
using System.Threading.Tasks;

class Program
{
    static async Task Main()
    {
        var tokenSource2 = new CancellationTokenSource();
        CancellationToken ct = tokenSource2.Token;

        var task = Task.Run(() =>
        {
            // Were we already canceled?
            ct.ThrowIfCancellationRequested();

            bool moreToDo = true;
            while (moreToDo)
            {
                // Poll on this property if you have to do
                // other cleanup before throwing.
                if (ct.IsCancellationRequested)
                {
                    // Clean up here, then...
                    ct.ThrowIfCancellationRequested();
                }
            }
        }, tokenSource2.Token); // Pass same token to Task.Run.

        tokenSource2.Cancel();

        // Just continue on this thread, or await with try-catch:
        try
        {
            await task;
        }
        catch (OperationCanceledException e)
        {
            Console.WriteLine($"{nameof(OperationCanceledException)} thrown with message: {e.Message}");
        }
        finally
        {
            tokenSource2.Dispose();
        }

        Console.ReadKey();
    }
}
```

```

Imports System.Threading
Imports System.Threading.Tasks

Module Test
    Sub Main()
        Dim tokenSource2 As New CancellationTokenSource()
        Dim ct As CancellationToken = tokenSource2.Token

        Dim t2 = Task.Factory.StartNew(Sub()
            ' Were we already canceled?
            ct.ThrowIfCancellationRequested()

            Dim moreToDo As Boolean = True
            While moreToDo = True
                ' Poll on this property if you have to do
                ' other cleanup before throwing.
                If ct.IsCancellationRequested Then

                    ' Clean up here, then...
                    ct.ThrowIfCancellationRequested()
                End If

                End While
            End Sub _
        , tokenSource2.Token) ' Pass same token to StartNew.

        ' Cancel the task.
        tokenSource2.Cancel()

        ' Just continue on this thread, or Wait/WaitAll with try-catch:
        Try
            t2.Wait()

            Catch e As AggregateException

                For Each item In e.InnerExceptions
                    Console.WriteLine(e.Message & " " & item.Message)
                Next
            Finally
                tokenSource2.Dispose()
            End Try

            Console.ReadKey()
        End Sub
    End Module

```

For a complete example, see [How to: Cancel a Task and Its Children](#).

When a task instance observes an [OperationCanceledException](#) thrown by the user code, it compares the exception's token to its associated token (the one that was passed to the API that created the Task). If the tokens are same and the token's [IsCancellationRequested](#) property returns `true`, the task interprets this as acknowledging cancellation and transitions to the [Canceled](#) state. If you don't use a [Wait](#) or [WaitAll](#) method to wait for the task, then the task just sets its status to [Canceled](#).

If you're waiting on a Task that transitions to the [Canceled](#) state, a [System.Threading.Tasks.TaskCanceledException](#) exception (wrapped in an [AggregateException](#) exception) is thrown. This exception indicates successful cancellation instead of a faulty situation. Therefore, the task's [Exception](#) property returns `null`.

If the token's [IsCancellationRequested](#) property returns `false` or if the exception's token doesn't match the Task's token, the [OperationCanceledException](#) is treated like a normal exception, causing the Task to transition to the [Faulted](#) state. The presence of other exceptions will also cause the Task to transition to the [Faulted](#) state. You can get the status of the completed task in the [Status](#) property.

It's possible that a task might continue to process some items after cancellation is requested.

See also

- [Cancellation in Managed Threads](#)
- [How to: Cancel a Task and Its Children](#)

Exception handling (Task Parallel Library)

9/20/2022 • 14 minutes to read • [Edit Online](#)

Unhandled exceptions that are thrown by user code that is running inside a task are propagated back to the calling thread, except in certain scenarios that are described later in this topic. Exceptions are propagated when you use one of the static or instance `Task.Wait` methods, and you handle them by enclosing the call in a `try / catch` statement. If a task is the parent of attached child tasks, or if you are waiting on multiple tasks, multiple exceptions could be thrown.

To propagate all the exceptions back to the calling thread, the Task infrastructure wraps them in an `AggregateException` instance. The `AggregateException` exception has an `InnerExceptions` property that can be enumerated to examine all the original exceptions that were thrown, and handle (or not handle) each one individually. You can also handle the original exceptions by using the `AggregateException.Handle` method.

Even if only one exception is thrown, it is still wrapped in an `AggregateException` exception, as the following example shows.

```
public static partial class Program
{
    public static void HandleThree()
    {
        var task = Task.Run(
            () => throw new CustomException("This exception is expected!"));

        try
        {
            task.Wait();
        }
        catch (AggregateException ae)
        {
            foreach (var ex in ae.InnerExceptions)
            {
                // Handle the custom exception.
                if (ex is CustomException)
                {
                    Console.WriteLine(ex.Message);
                }
                // Rethrow any other exception.
                else
                {
                    throw ex;
                }
            }
        }
    }
}

// The example displays the following output:
//      This exception is expected!
```

```

Imports System.Threading.Tasks

Module Example
    Public Sub Main()
        Dim task1 = Task.Run(Sub() Throw New CustomException("This exception is expected!"))

        Try
            task1.Wait()
        Catch ae As AggregateException
            For Each ex In ae.InnerExceptions
                ' Handle the custom exception.
                If TypeOf ex Is CustomException Then
                    Console.WriteLine(ex.Message)
                    ' Rethrow any other exception.
                Else
                    Throw ex
                End If
            Next
        End Try
    End Sub
End Module

Class CustomException : Inherits Exception
    Public Sub New(s As String)
        MyBase.New(s)
    End Sub
End Class
' The example displays the following output:
'      This exception is expected!

```

You could avoid an unhandled exception by just catching the [AggregateException](#) and not observing any of the inner exceptions. However, we recommend that you do not do this because it is analogous to catching the base [Exception](#) type in non-parallel scenarios. To catch an exception without taking specific actions to recover from it can leave your program in an indeterminate state.

If you do not want to call the [Task.Wait](#) method to wait for a task's completion, you can also retrieve the [AggregateException](#) exception from the task's [Exception](#) property, as the following example shows. For more information, see the [Observing exceptions by using the Task.Exception property](#) section in this topic.

```

public static partial class Program
{
    public static void HandleFour()
    {
        var task = Task.Run(
            () => throw new CustomException("This exception is expected!"));

        while (!task.IsCompleted) { }

        if (task.Status == TaskStatus.Faulted)
        {
            foreach (var ex in task.Exception?.InnerExceptions ?? new(Array.Empty<Exception>()))
            {
                // Handle the custom exception.
                if (ex is CustomException)
                {
                    Console.WriteLine(ex.Message);
                }
                // Rethrow any other exception.
                else
                {
                    throw ex;
                }
            }
        }
    }
}

// The example displays the following output:
//      This exception is expected!

```

```

Imports System.Threading.Tasks

Module Example
    Public Sub Main()
        Dim task1 = Task.Run(Sub() Throw New CustomException("This exception is expected!"))

        While Not task1.IsCompleted
            End While

        If task1.Status = TaskStatus.Faulted Then
            For Each ex In task1.Exception.InnerExceptions
                ' Handle the custom exception.
                If TypeOf ex Is CustomException Then
                    Console.WriteLine(ex.Message)
                    ' Rethrow any other exception.
                Else
                    Throw ex
                End If
            Next
        End If
    End Sub
End Module

Class CustomException : Inherits Exception
    Public Sub New(s As String)
        MyBase.New(s)
    End Sub
End Class

' The example displays the following output:
'      This exception is expected!

```

Caution

The preceding example code includes a `while` loop that polls the task's `Task.IsCompleted` property to determine

when the task has completed. This should never be done in production code as it is very inefficient.

If you do not wait on a task that propagates an exception, or access its [Exception](#) property, the exception is escalated according to the .NET exception policy when the task is garbage-collected.

When exceptions are allowed to bubble up back to the joining thread, it is possible that a task may continue to process some items after the exception is raised.

NOTE

When "Just My Code" is enabled, Visual Studio in some cases will break on the line that throws the exception and display an error message that says "exception not handled by user code." This error is benign. You can press F5 to continue and see the exception-handling behavior that is demonstrated in these examples. To prevent Visual Studio from breaking on the first error, just uncheck the **Enable Just My Code** checkbox under **Tools, Options, Debugging, General**.

Attached child tasks and nested AggregateExceptions

If a task has an attached child task that throws an exception, that exception is wrapped in an [AggregateException](#) before it is propagated to the parent task, which wraps that exception in its own [AggregateException](#) before it propagates it back to the calling thread. In such cases, the [InnerExceptions](#) property of the [AggregateException](#) exception that is caught at the [Task.Wait](#), [WaitAny](#), or [WaitAll](#) method contains one or more [AggregateException](#) instances, not the original exceptions that caused the fault. To avoid having to iterate over nested [AggregateException](#) exceptions, you can use the [Flatten](#) method to remove all the nested [AggregateException](#) exceptions, so that the [AggregateException.InnerExceptions](#) property contains the original exceptions. In the following example, nested [AggregateException](#) instances are flattened and handled in just one loop.

```
public static partial class Program
{
    public static void FlattenTwo()
    {
        var task = Task.Factory.StartNew(() =>
        {
            var child = Task.Factory.StartNew(() =>
            {
                var grandChild = Task.Factory.StartNew(() =>
                {
                    // This exception is nested inside three AggregateExceptions.
                    throw new CustomException("Attached child2 faulted.");
                }, TaskCreationOptions.AttachedToParent);

                    // This exception is nested inside two AggregateExceptions.
                    throw new CustomException("Attached child1 faulted.");
                }, TaskCreationOptions.AttachedToParent);
            });
        });

        try
        {
            task.Wait();
        }
        catch (AggregateException ae)
        {
            foreach (var ex in ae.Flatten().InnerExceptions)
            {
                if (ex is CustomException)
                {
                    Console.WriteLine(ex.Message);
                }
                else
                {
                    throw;
                }
            }
        }
    }
}

// The example displays the following output:
//      Attached child1 faulted.
//      Attached child2 faulted.
```

```

Imports System.Threading.Tasks

Module Example
    Public Sub Main()
        Dim task1 = Task.Factory.StartNew(Sub()
            Dim child1 = Task.Factory.StartNew(Sub()
                Dim child2 =
Task.Factory.StartNew(Sub()

Throw New CustomException("Attached child2 faulted.")

End Sub,

TaskCreationOptions.AttachedToParent)
Throw New
CustomException("Attached child1 faulted.")
End Sub,

TaskCreationOptions.AttachedToParent)
End Sub)

Try
    task1.Wait()
Catch ae As AggregateException
    For Each ex In ae.Flatten().InnerExceptions
        If TypeOf ex Is CustomException Then
            Console.WriteLine(ex.Message)
        Else
            Throw
        End If
    Next
End Try
End Sub
End Module

Class CustomException : Inherits Exception
    Public Sub New(s As String)
        MyBase.New(s)
    End Sub
End Class
' The example displays the following output:
'     Attached child1 faulted.
'     Attached child2 faulted.

```

You can also use the [AggregateException.Flatten](#) method to rethrow the inner exceptions from multiple [AggregateException](#) instances thrown by multiple tasks in a single [AggregateException](#) instance, as the following example shows.

```

public static partial class Program
{
    public static void TaskExceptionTwo()
    {
        try
        {
            ExecuteTasks();
        }
        catch (AggregateException ae)
        {
            foreach (var e in ae.InnerExceptions)
            {
                Console.WriteLine(
                    "{0}:\n {1}", e.GetType().Name, e.Message);
            }
        }
    }

    static void ExecuteTasks()
    {
        // Assume this is a user-entered String.
        string path = @"C:\";

        List<Task> tasks = new();

        tasks.Add(Task.Run(() =>
        {
            // This should throw an UnauthorizedAccessException.
            return Directory.GetFiles(
                path, "*.*",
                SearchOption.AllDirectories);
        }));

        tasks.Add(Task.Run(() =>
        {
            if (path == @"C:\")
            {
                throw new ArgumentException(
                    "The system root is not a valid path.");
            }
            return new string[] { ".txt", ".dll", ".exe", ".bin", ".dat" };
        }));

        tasks.Add(Task.Run(() =>
        {
            throw new NotImplementedException(
                "This operation has not been implemented.");
        }));
    }

    try
    {
        Task.WaitAll(tasks.ToArray());
    }
    catch (AggregateException ae)
    {
        throw ae.Flatten();
    }
}

// The example displays the following output:
//      UnauthorizedAccessException:
//          Access to the path 'C:\Documents and Settings' is denied.
//      ArgumentException:
//          The system root is not a valid path.
//      NotImplementedException:
//          This operation has not been implemented.

```

```

Imports System.Collections.Generic
Imports System.IO
Imports System.Threading.Tasks

Module Example
    Public Sub Main()
        Try
            ExecuteTasks()
        Catch ae As AggregateException
            For Each e In ae.InnerExceptions
                Console.WriteLine("{0}:{1}", e.GetType().Name, e.Message,
                    vbCrLf)
            Next
        End Try
    End Sub

    Sub ExecuteTasks()
        ' Assume this is a user-entered String.
        Dim path = "C:\"
        Dim tasks As New List(Of Task)

        tasks.Add(Task.Run(Function()
            ' This should throw an UnauthorizedAccessException.
            Return Directory.GetFiles(path, "*.txt",
                SearchOption.AllDirectories)
        End Function))

        tasks.Add(Task.Run(Function()
            If path = "C:\" Then
                Throw New ArgumentException("The system root is not a valid path.")
            End If
            Return {"*.txt", ".dll", ".exe", ".bin", ".dat"}
        End Function))

        tasks.Add(Task.Run(Sub()
            Throw New NotImplementedException("This operation has not been implemented.")
        End Sub))

        Try
            Task.WaitAll(tasks.ToArray)
        Catch ae As AggregateException
            Throw ae.Flatten()
        End Try
    End Sub
End Module
' The example displays the following output:
'     UnauthorizedAccessException:
'         Access to the path 'C:\Documents and Settings' is denied.
'     ArgumentException:
'         The system root is not a valid path.
'     NotImplementedException:
'         This operation has not been implemented.

```

Exceptions from detached child tasks

By default, child tasks are created as detached. Exceptions thrown from detached tasks must be handled or rethrown in the immediate parent task; they are not propagated back to the calling thread in the same way as attached child tasks propagated back. The topmost parent can manually rethrow an exception from a detached child to cause it to be wrapped in an [AggregateException](#) and propagated back to the calling thread.

```
public static partial class Program
{
    public static void DetachedTwo()
    {
        var task = Task.Run(() =>
        {
            var nestedTask = Task.Run(
                () => throw new CustomException("Detached child task faulted."));

            // Here the exception will be escalated back to the calling thread.
            // We could use try/catch here to prevent that.
            nestedTask.Wait();
        });

        try
        {
            task.Wait();
        }
        catch (AggregateException ae)
        {
            foreach (var e in ae.Flatten().InnerExceptions)
            {
                if (e is CustomException)
                {
                    Console.WriteLine(e.Message);
                }
            }
        }
    }
}

// The example displays the following output:
//     Detached child task faulted.
```

```

Imports System.Threading.Tasks

Module Example
    Public Sub Main()
        Dim task1 = Task.Run(Sub()
            Dim nestedTask1 = Task.Run(Sub()
                Throw New CustomException("Detached child
task faulted.")
            End Sub)
            ' Here the exception will be escalated back to joining thread.
            ' We could use try/catch here to prevent that.
            nestedTask1.Wait()
        End Sub)

        Try
            task1.Wait()
        Catch ae As AggregateException
            For Each ex In ae.Flatten().InnerExceptions
                If TypeOf ex Is CustomException Then
                    ' Recover from the exception. Here we just
                    ' print the message for demonstration purposes.
                    Console.WriteLine(ex.Message)
                End If
            Next
        End Try
    End Sub
End Module

Class CustomException : Inherits Exception
    Public Sub New(s As String)
        MyBase.New(s)
    End Sub
End Class
' The example displays the following output:
'     Detached child task faulted.

```

Even if you use a continuation to observe an exception in a child task, the exception still must be observed by the parent task.

Exceptions that indicate cooperative cancellation

When user code in a task responds to a cancellation request, the correct procedure is to throw an [OperationCanceledException](#) passing in the cancellation token on which the request was communicated. Before it attempts to propagate the exception, the task instance compares the token in the exception to the one that was passed to it when it was created. If they are the same, the task propagates a [TaskCanceledException](#) wrapped in the [AggregateException](#), and it can be seen when the inner exceptions are examined. However, if the calling thread is not waiting on the task, this specific exception will not be propagated. For more information, see [Task Cancellation](#).

```

var tokenSource = new CancellationTokenSource();
var token = tokenSource.Token;
var task = Task.Factory.StartNew(() =>
{
    CancellationToken ct = token;
    while (someCondition)
    {
        // Do some work...
        Thread.SpinWait(50_000);
        ct.ThrowIfCancellationRequested();
    }
},
token);

// No waiting required.
tokenSource.Dispose();

```

```

Dim someCondition As Boolean = True
Dim tokenSource = New CancellationTokenSource()
Dim token = tokenSource.Token

Dim task1 = Task.Factory.StartNew(Sub()
    Dim ct As CancellationToken = token
    While someCondition = True
        ' Do some work...
        Thread.SpinWait(500000)
        ct.ThrowIfCancellationRequested()
    End While
End Sub,
token)

```

Using the handle method to filter inner exceptions

You can use the [AggregateException.Handle](#) method to filter out exceptions that you can treat as "handled" without using any further logic. In the user delegate that is supplied to the [AggregateException.Handle\(Func<Exception,Boolean>\)](#) method, you can examine the exception type, its [Message](#) property, or any other information about it that will let you determine whether it is benign. Any exceptions for which the delegate returns `false` are rethrown in a new [AggregateException](#) instance immediately after the [AggregateException.Handle](#) method returns.

The following example is functionally equivalent to the first example in this topic, which examines each exception in the [AggregateException.InnerExceptions](#) collection. Instead, this exception handler calls the [AggregateException.Handle](#) method object for each exception, and only rethrows exceptions that are not `CustomException` instances.

```

public static partial class Program
{
    public static void HandleMethodThree()
    {
        var task = Task.Run(
            () => throw new CustomException("This exception is expected!"));

        try
        {
            task.Wait();
        }
        catch (AggregateException ae)
        {
            // Call the Handle method to handle the custom exception,
            // otherwise rethrow the exception.
            ae.Handle(ex =>
            {
                if (ex is CustomException)
                {
                    Console.WriteLine(ex.Message);
                }
                return ex is CustomException;
            });
        }
    }
}

// The example displays the following output:
//      This exception is expected!

```

```

Imports System.Threading.Tasks

Module Example
    Public Sub Main()
        Dim task1 = Task.Run(Sub() Throw New CustomException("This exception is expected!"))

        Try
            task1.Wait()
        Catch ae As AggregateException
            ' Call the Handle method to handle the custom exception,
            ' otherwise rethrow the exception.
            ae.Handle(Function(e)
                If TypeOf e Is CustomException Then
                    Console.WriteLine(e.Message)
                End If
                Return TypeOf e Is CustomException
            End Function)
        End Try
    End Sub
End Module

Class CustomException : Inherits Exception
    Public Sub New(s As String)
        MyBase.New(s)
    End Sub
End Class

' The example displays the following output:
'      This exception is expected!

```

The following is a more complete example that uses the [AggregateException.Handle](#) method to provide special handling for an [UnauthorizedAccessException](#) exception when enumerating files.

```

public static partial class Program
{

```

```

public static void TaskException()
{
    // This should throw an UnauthorizedAccessException.
    try
    {
        if (GetAllFiles(@"C:\") is { Length: > 0 } files)
        {
            foreach (var file in files)
            {
                Console.WriteLine(file);
            }
        }
    }
    catch (AggregateException ae)
    {
        foreach (var ex in ae.InnerExceptions)
        {
            Console.WriteLine(
                "{0}: {1}", ex.GetType().Name, ex.Message);
        }
    }
    Console.WriteLine();

    // This should throw an ArgumentException.
    try
    {
        foreach (var s in GetAllFiles(""))
        {
            Console.WriteLine(s);
        }
    }
    catch (AggregateException ae)
    {
        foreach (var ex in ae.InnerExceptions)
            Console.WriteLine(
                "{0}: {1}", ex.GetType().Name, ex.Message);
    }
}

static string[] GetAllFiles(string path)
{
    var task1 =
        Task.Run(() => Directory.GetFiles(
            path, "*.*",
            SearchOption.AllDirectories));

    try
    {
        return task1.Result;
    }
    catch (AggregateException ae)
    {
        ae.Handle(x =>
        {
            // Handle an UnauthorizedAccessException
            if (x is UnauthorizedAccessException)
            {
                Console.WriteLine(
                    "You do not have permission to access all folders in this path.");
                Console.WriteLine(
                    "See your network administrator or try another path.");
            }
            return x is UnauthorizedAccessException;
        });
        return Array.Empty<string>();
    }
}
// The example displays the following output:

```

```
//      You do not have permission to access all folders in this path.  
//      See your network administrator or try another path.  
//  
//      ArgumentException: The path is not of a legal form.
```

```
Imports System.IO  
Imports System.Threading.Tasks  
  
Module Example  
    Public Sub Main()  
        ' This should throw an UnauthorizedAccessException.  
        Try  
            Dim files = GetAllFiles("C:\")  
            If files IsNot Nothing Then  
                For Each file In files  
                    Console.WriteLine(file)  
                Next  
            End If  
        Catch ae As AggregateException  
            For Each ex In ae.InnerExceptions  
                Console.WriteLine("{0}: {1}", ex.GetType().Name, ex.Message)  
            Next  
        End Try  
        Console.WriteLine()  
  
        ' This should throw an ArgumentException.  
        Try  
            For Each s In GetAllFiles("")  
                Console.WriteLine(s)  
            Next  
        Catch ae As AggregateException  
            For Each ex In ae.InnerExceptions  
                Console.WriteLine("{0}: {1}", ex.GetType().Name, ex.Message)  
            Next  
        End Try  
        Console.WriteLine()  
    End Sub  
  
    Function GetAllFiles(ByVal path As String) As String()  
        Dim task1 = Task.Run(Function()  
            Return Directory.GetFiles(path, "*.*",  
                SearchOption.AllDirectories)  
        End Function)  
        Try  
            Return task1.Result  
        Catch ae As AggregateException  
            ae.Handle(Function(x)  
                ' Handle an UnauthorizedAccessException  
                If TypeOf x Is UnauthorizedAccessException Then  
                    Console.WriteLine("You do not have permission to access all folders in this  
path.")  
                    Console.WriteLine("See your network administrator or try another path.")  
                End If  
                Return TypeOf x Is UnauthorizedAccessException  
            End Function)  
        End Try  
        Return Array.Empty(Of String)()  
    End Function  
End Module  
' The example displays the following output:  
'      You do not have permission to access all folders in this path.  
'      See your network administrator or try another path.  
'  
'      ArgumentException: The path is not of a legal form.
```

Observing exceptions by using the Task.Exception property

If a task completes in the `TaskStatus.Faulted` state, its `Exception` property can be examined to discover which specific exception caused the fault. A good way to observe the `Exception` property is to use a continuation that runs only if the antecedent task faults, as shown in the following example.

```
public static partial class Program
{
    public static void ExceptionPropagationTwo()
    {
        _ = Task.Run(
            () => throw new CustomException("task1 faulted."))
            .ContinueWith(_ =>
        {
            if (_.Exception?.InnerException is { } inner)
            {
                Console.WriteLine("{0}: {1}",
                    inner.GetType().Name,
                    inner.Message);
            }
        },
        TaskContinuationOptions.OnlyOnFaulted);

        Thread.Sleep(500);
    }
}

// The example displays output like the following:
//      CustomException: task1 faulted.
```

```
Imports System.Threading
Imports System.Threading.Tasks

Module Example
    Public Sub Main()
        Dim task1 = Task.Factory.StartNew(Sub()
            Throw New CustomException("task1 faulted.")
        End Sub).
            ContinueWith(Sub(t)
                Console.WriteLine("{0}: {1}",
                    t.Exception.InnerException.GetType().Name,
                    t.Exception.InnerException.Message)
            End Sub, TaskContinuationOptions.OnlyOnFaulted)

        Thread.Sleep(500)
    End Sub
End Module

Class CustomException : Inherits Exception
    Public Sub New(s As String)
        MyBase.New(s)
    End Sub
End Class

' The example displays output like the following:
'      CustomException: task1 faulted.
```

In a meaningful application, the continuation delegate could log detailed information about the exception and possibly spawn new tasks to recover from the exception. If a task faults, the following expressions throw the exception:

- `await task`
- `task.Wait()`

- `task.Result`
- `task.GetAwaiter().GetResult()`

Use a `try-catch` statement to handle and observe thrown exceptions. Alternatively, observe the exception by accessing the [Task.Exception](#) property.

IMPORTANT

The [AggregateException](#) cannot be explicitly caught when using the following expressions:

- `await task`
- `task.GetAwaiter().GetResult()`

UnobservedTaskException event

In some scenarios, such as when hosting untrusted plug-ins, benign exceptions might be common, and it might be too difficult to manually observe them all. In these cases, you can handle the [TaskScheduler.UnobservedTaskException](#) event. The [System.Threading.Tasks.UnobservedTaskExceptionEventArgs](#) instance that is passed to your handler can be used to prevent the unobserved exception from being propagated back to the joining thread.

See also

- [Task Parallel Library \(TPL\)](#)

How to: Use Parallel.Invoke to Execute Parallel Operations

9/20/2022 • 4 minutes to read • [Edit Online](#)

This example shows how to parallelize operations by using [Invoke](#) in the Task Parallel Library. Three operations are performed on a shared data source. The operations can be executed in parallel in a straightforward manner, because none of them modifies the source.

NOTE

This documentation uses lambda expressions to define delegates in TPL. If you aren't familiar with lambda expressions in C# or Visual Basic, see [Lambda Expressions in PLINQ and TPL](#).

Example

```
namespace ParallelTasks
{
    using System;
    using System.IO;
    using System.Linq;
    using System.Text;
    using System.Threading;
    using System.Threading.Tasks;
    using System.Net;

    class ParallelInvoke
    {
        static void Main()
        {
            // Retrieve Goncharov's "Oblomov" from Gutenberg.org.
            string[] words = CreateWordArray(@"http://www.gutenberg.org/files/54700/54700-0.txt");

            #region ParallelTasks
            // Perform three tasks in parallel on the source array
            Parallel.Invoke(() =>
            {
                Console.WriteLine("Begin first task...");
                GetLongestWord(words);
            }, // close first Action

            () =>
            {
                Console.WriteLine("Begin second task...");
                GetMostCommonWords(words);
            }, //close second Action

            () =>
            {
                Console.WriteLine("Begin third task...");
                GetCountForWord(words, "sleep");
            } //close third Action
        ); //close parallel.invoke

        Console.WriteLine("Returned from Parallel.Invoke");
        #endregion

        Console.WriteLine("Press any key to exit");
    }
}
```

```

        Console.ReadKey();
    }

    #region HelperMethods
    private static void GetCountForWord(string[] words, string term)
    {
        var findWord = from word in words
                      where word.ToUpper().Contains(term.ToUpper())
                      select word;

        Console.WriteLine($"Task 3 -- The word "{term}" occurs {findWord.Count()} times.");
    }

    private static void GetMostCommonWords(string[] words)
    {
        var frequencyOrder = from word in words
                              where word.Length > 6
                              group word by word into g
                              orderby g.Count() descending
                              select g.Key;

        var commonWords = frequencyOrder.Take(10);

        StringBuilder sb = new StringBuilder();
        sb.AppendLine("Task 2 -- The most common words are:");
        foreach (var v in commonWords)
        {
            sb.AppendLine(" " + v);
        }
        Console.WriteLine(sb.ToString());
    }

    private static string GetLongestWord(string[] words)
    {
        var longestWord = (from w in words
                           orderby w.Length descending
                           select w).First();

        Console.WriteLine($"Task 1 -- The longest word is {longestWord}.");
        return longestWord;
    }

    // An http request performed synchronously for simplicity.
    static string[] CreateWordArray(string uri)
    {
        Console.WriteLine($"Retrieving from {uri}");

        // Download a web page the easy way.
        string s = new WebClient().DownloadString(uri);

        // Separate string into an array of words, removing some common punctuation.
        return s.Split(
            new char[] { ' ', '\u000A', ',', '.', ';', ':', '-', '_', '/' },
            StringSplitOptions.RemoveEmptyEntries);
    }
    #endregion
}

// The example displays output like the following:
//     Retrieving from http://www.gutenberg.org/files/54700/54700-0.txt
//     Begin first task...
//     Begin second task...
//     Begin third task...
//     Task 2 -- The most common words are:
//     Oblomov
//     himself
//     Schtoltz
//     Gutenberg
//     Project

```

```

//           another
//           thought
//           Oblomov's
//           nothing
//           replied
//
//           Task 1 -- The longest word is incomprehensible.
//           Task 3 -- The word "sleep" occurs 57 times.
//           Returned from Parallel.Invoke
//           Press any key to exit

```

```

Imports System.Net
Imports System.Threading.Tasks

Module ParallelTasks
    Sub Main()
        ' Retrieve Goncharov's "Oblomov" from Gutenberg.org.
        Dim words As String() = CreateWordArray("http://www.gutenberg.org/files/54700/54700-0.txt")

        '#Region "ParallelTasks"
        ' Perform three tasks in parallel on the source array
        Parallel.Invoke(Sub()
                            Console.WriteLine("Begin first task...")
                            GetLongestWord(words)
                            ' close first Action
                        End Sub,
                        Sub()
                            Console.WriteLine("Begin second task...")
                            GetMostCommonWords(words)
                            'close second Action
                        End Sub,
                        Sub()
                            Console.WriteLine("Begin third task...")
                            GetCountForWord(words, "sleep")
                            'close third Action
                        End Sub)
        'close parallel.invoke
        Console.WriteLine("Returned from Parallel.Invoke")
        '#End Region

        Console.WriteLine("Press any key to exit")
        Console.ReadKey()
    End Sub

    #Region "HelperMethods"
    Sub GetCountForWord(ByVal words As String(), ByVal term As String)
        Dim findWord = From word In words
                      Where word.ToUpper().Contains(term.ToUpper())
                      Select word

        Console.WriteLine($"Task 3 -- The word """{term}"" occurs {findWord.Count()} times.")
    End Sub

    Sub GetMostCommonWords(ByVal words As String())
        Dim frequencyOrder = From word In words
                              Where word.Length > 6
                              Group By word
                              Into wordGroup = Group, Count()
                              Order By wordGroup.Count() Descending
                              Select wordGroup

        Dim commonWords = From grp In frequencyOrder
                         Select grp
                         Take (10)

        Dim s As String
        s = "Task 2 -- The most common words are:" & vbCrLf
    End Sub

```

```

    For Each v In commonWords
        s = s & v(0) & vbCrLf
    Next
    Console.WriteLine(s)
End Sub

Function GetLongestWord(ByVal words As String()) As String
    Dim longestWord = (From w In words
                           Order By w.Length Descending
                           Select w).First()

    Console.WriteLine($"Task 1 -- The longest word is {longestWord}.")
    Return longestWord
End Function

' An http request performed synchronously for simplicity.
Function CreateWordArray(ByVal uri As String) As String()
    Console.WriteLine($"Retrieving from {uri}")

    ' Download a web page the easy way.
    Dim s As String = New WebClient().DownloadString(uri)

    ' Separate string into an array of words, removing some common punctuation.
    Return s.Split(New Char() {"c, ControlChars.Lf, "c, ".c, ";"c, ":"c,
    "-"c, "_"c, "/"c}, StringSplitOptions.RemoveEmptyEntries)
End Function
#End Region
End Module

' The example displays output like the following:
'     Retrieving from http://www.gutenberg.org/files/54700/54700-0.txt
'     Begin first task...
'     Begin second task...
'     Begin third task...
'     Task 2 -- The most common words are:
'     Oblomov
'     himself
'     Schtoltz
'     Gutenberg
'     Project
'     another
'     thought
'     Oblomov's
'     nothing
'     replied
'
'     Task 1 -- The longest word is incomprehensible.
'     Task 3 -- The word "sleep" occurs 57 times.
'     Returned from Parallel.Invoke
'     Press any key to exit

```

With [Invoke](#), you simply express which actions you want to run concurrently, and the runtime handles all thread scheduling details, including scaling automatically to the number of cores on the host computer.

This example parallelizes the operations, not the data. As an alternate approach, you can parallelize the LINQ queries by using PLINQ and run the queries sequentially. Alternatively, you could parallelize the data by using PLINQ. Another option is to parallelize both the queries and the tasks. Although the resulting overhead might degrade performance on host computers with relatively few processors, it scales better on computers with many processors.

Compile the Code

Copy and paste the entire example into a Microsoft Visual Studio project and press F5.

See also

- [Parallel Programming](#)
- [How to: Cancel a Task and Its Children](#)
- [Parallel LINQ \(PLINQ\)](#)

How to: Return a Value from a Task

9/20/2022 • 2 minutes to read • [Edit Online](#)

This example shows how to use the `System.Threading.Tasks.Task<TResult>` class to return a value from the `Result` property. To use this example, you must ensure that the `C:\Users\Public\Pictures\Sample Pictures` directory exists and that it contains files.

Example

```
using System;
using System.Linq;
using System.Threading.Tasks;

class Program
{
    static void Main()
    {
        // Return a value type with a lambda expression
        Task<int> task1 = Task<int>.Factory.StartNew(() => 1);
        int i = task1.Result;

        // Return a named reference type with a multi-line statement lambda.
        Task<Test> task2 = Task<Test>.Factory.StartNew(() =>
        {
            string s = ".NET";
            double d = 4.0;
            return new Test { Name = s, Number = d };
        });
        Test test = task2.Result;

        // Return an array produced by a PLINQ query
        Task<string[]> task3 = Task<string[]>.Factory.StartNew(() =>
        {
            string path = @"C:\Users\Public\Pictures\Sample Pictures\";
            string[] files = System.IO.Directory.GetFiles(path);

            var result = (from file in files.AsParallel()
                         let info = new System.IO.FileInfo(file)
                         where info.Extension == ".jpg"
                         select file).ToArray();

            return result;
        });

        foreach (var name in task3.Result)
            Console.WriteLine(name);
    }
    class Test
    {
        public string Name { get; set; }
        public double Number { get; set; }
    }
}
```

```

Imports System.Threading.Tasks

Module Module1

    Sub Main()
        ReturnAValue()

        Console.WriteLine("Press any key to exit.")
        Console.ReadKey()

    End Sub

    Sub ReturnAValue()

        ' Return a value type with a lambda expression
        Dim task1 = Task(Of Integer).Factory.StartNew(Function() 1)
        Dim i As Integer = task1.Result

        ' Return a named reference type with a multi-line statement lambda.
        Dim task2 As Task(Of Test) = Task.Factory.StartNew(Function()
            Dim s As String = ".NET"
            Dim d As Integer = 4
            Return New Test With {.Name = s, .Number = d}
        End Function)

        Dim myTest As Test = task2.Result
        Console.WriteLine(myTest.Name & ":" & myTest.Number)

        ' Return an array produced by a PLINQ query.
        Dim task3 As Task(Of String()) = Task(Of String()).Factory.StartNew(Function()

            Dim path =
"C:\Users\Public\Pictures\Sample Pictures\"

            Dim files =
System.IO.Directory.GetFiles(path)

            Dim result = (From file In
files.AsParallel()
            Let info = New
System.IO.FileInfo(file)
            info.Extension = ".jpg"
            Select
file).ToArray()

            Return result
        End Function)

        For Each name As String In task3.Result
            Console.WriteLine(name)
        Next
    End Sub

    Class Test
        Public Name As String
        Public Number As Double
    End Class
End Module

```

The [Result](#) property blocks the calling thread until the task finishes.

To see how to pass the result of a [System.Threading.Tasks.Task<TResult>](#) class to a continuation task, see [Chaining Tasks by Using Continuation Tasks](#).

See also

- Task-based Asynchronous Programming
- Lambda Expressions in PLINQ and TPL

How to: Cancel a Task and Its Children

9/20/2022 • 7 minutes to read • [Edit Online](#)

These examples show how to perform the following tasks:

1. Create and start a cancelable task.
2. Pass a cancellation token to your user delegate and optionally to the task instance.
3. Notice and respond to the cancellation request in your user delegate.
4. Optionally notice on the calling thread that the task was canceled.

The calling thread does not forcibly end the task; it only signals that cancellation is requested. If the task is already running, it is up to the user delegate to notice the request and respond appropriately. If cancellation is requested before the task runs, then the user delegate is never executed and the task object transitions into the Canceled state.

Example

This example shows how to terminate a [Task](#) and its children in response to a cancellation request. It also shows that when a user delegate terminates by throwing a [TaskCanceledException](#), the calling thread can optionally use the [Wait](#) method or [WaitAll](#) method to wait for the tasks to finish. In this case, you must use a `try/catch` block to handle the exceptions on the calling thread.

```
using System;
using System.Collections.Concurrent;
using System.Threading;
using System.Threading.Tasks;

public class Example
{
    public static async Task Main()
    {
        var tokenSource = new CancellationTokenSource();
        var token = tokenSource.Token;

        // Store references to the tasks so that we can wait on them and
        // observe their status after cancellation.
        Task t;
        var tasks = new ConcurrentBag<Task>();

        Console.WriteLine("Press any key to begin tasks...");
        Console.ReadKey(true);
        Console.WriteLine("To terminate the example, press 'c' to cancel and exit...");
        Console.WriteLine();

        // Request cancellation of a single task when the token source is canceled.
        // Pass the token to the user delegate, and also to the task so it can
        // handle the exception correctly.
        t = Task.Run(() => DoSomeWork(1, token), token);
        Console.WriteLine("Task {0} executing", t.Id);
        tasks.Add(t);

        // Request cancellation of a task and its children. Note the token is passed
        // to (1) the user delegate and (2) as the second argument to Task.Run, so
        // that the task instance can correctly handle the OperationCanceledException.
        t = Task.Run(() =>
        {
            // User code here
        }, token);
        tasks.Add(t);
    }
}

async void DoSomeWork(int id, CancellationToken token)
{
    await Task.Delay(1000 * id);
    if (token.IsCancellationRequested)
    {
        throw new TaskCanceledException();
    }
}
```

```

// Create some cancelable child tasks.
Task tc;
for (int i = 3; i <= 10; i++)
{
    // For each child task, pass the same token
    // to each user delegate and to Task.Run.
    tc = Task.Run(() => DoSomeWork(i, token), token);
    Console.WriteLine("Task {0} executing", tc.Id);
    tasks.Add(tc);
    // Pass the same token again to do work on the parent task.
    // All will be signaled by the call to tokenSource.Cancel below.
    DoSomeWork(2, token);
}
}, token);

Console.WriteLine("Task {0} executing", t.Id);
tasks.Add(t);

// Request cancellation from the UI thread.
char ch = Console.ReadKey().KeyChar;
if (ch == 'c' || ch == 'C')
{
    tokenSource.Cancel();
    Console.WriteLine("\nTask cancellation requested.");

    // Optional: Observe the change in the Status property on the task.
    // It is not necessary to wait on tasks that have canceled. However,
    // if you do wait, you must enclose the call in a try-catch block to
    // catch the TaskCanceledExceptions that are thrown. If you do
    // not wait, no exception is thrown if the token that was passed to the
    // Task.Run method is the same token that requested the cancellation.
}

try
{
    await Task.WhenAll(tasks.ToArray());
}
catch (OperationCanceledException)
{
    Console.WriteLine($"\\n{nameof(OperationCanceledException)} thrown\\n");
}
finally
{
    tokenSource.Dispose();
}

// Display status of all tasks.
foreach (var task in tasks)
    Console.WriteLine("Task {0} status is now {1}", task.Id, task.Status);
}

static void DoSomeWork(int taskNum, CancellationToken ct)
{
    // Was cancellation already requested?
    if (ct.IsCancellationRequested)
    {
        Console.WriteLine("Task {0} was cancelled before it got started.",
                         taskNum);
        ct.ThrowIfCancellationRequested();
    }

    int maxIterations = 100;

    // NOTE!!! A "TaskCanceledException was unhandled
    // by user code" error will be raised here if "Just My Code"
    // is enabled on your computer. On Express editions JMC is
    // enabled and cannot be disabled. The exception is benign.
    // Just press F5 to continue executing your code.
    for (int i = 0; i <= maxIterations: i++)

```

```

    ...
    ...
    ...
}

// Do a bit of work. Not too much.
var sw = new SpinWait();
for (int j = 0; j <= 100; j++)
    sw.SpinOnce();

if (ct.IsCancellationRequested)
{
    Console.WriteLine("Task {0} cancelled", taskNum);
    ct.ThrowIfCancellationRequested();
}
}

}

}

}

// The example displays output like the following:
//      Press any key to begin tasks...
//      To terminate the example, press 'c' to cancel and exit...
//
//      Task 1 executing
//      Task 2 executing
//      Task 3 executing
//      Task 4 executing
//      Task 5 executing
//      Task 6 executing
//      Task 7 executing
//      Task 8 executing
//      c
//      Task cancellation requested.
//      Task 2 cancelled
//      Task 7 cancelled
//
//      OperationCanceledException thrown
//
//      Task 2 status is now Canceled
//      Task 1 status is now RanToCompletion
//      Task 8 status is now Canceled
//      Task 7 status is now Canceled
//      Task 6 status is now RanToCompletion
//      Task 5 status is now RanToCompletion
//      Task 4 status is now RanToCompletion
//      Task 3 status is now RanToCompletion

```

```

Imports System.Collections.Concurrent
Imports System.Threading
Imports System.Threading.Tasks

Module Example
Sub Main()
    Dim tokenSource As New CancellationTokenSource()
    Dim token As CancellationToken = tokenSource.Token

    ' Store references to the tasks so that we can wait on them and
    ' observe their status after cancellation.
    Dim t As Task
    Dim tasks As New ConcurrentBag(Of Task)()

    Console.WriteLine("Press any key to begin tasks...")
    Console.ReadKey(True)
    Console.WriteLine("To terminate the example, press 'c' to cancel and exit...")
    Console.WriteLine()

    ' Request cancellation of a single task when the token source is canceled.
    ' Pass the token to the user delegate, and also to the task so it can
    ' handle the exception correctly.
    t = Task.Factory.StartNew(Sub() DoSomeWork(1, token), token)
    Console.WriteLine("Task {0} executing", t.Id)
    tasks.Add(t)

```

```

' Request cancellation of a task and its children. Note the token is passed
' to (1) the user delegate and (2) as the second argument to StartNew, so
' that the task instance can correctly handle the OperationCanceledException.
t = Task.Factory.StartNew(Sub()
    ' Create some cancelable child tasks.
    Dim tc As Task
    For i As Integer = 3 To 10
        ' For each child task, pass the same token
        ' to each user delegate and to StartNew.
        tc = Task.Factory.StartNew(Sub(iteration) DoSomeWork(iteration,
token), i, token)
        Console.WriteLine("Task {0} executing", tc.Id)
        tasks.Add(tc)
        ' Pass the same token again to do work on the parent task.
        ' All will be signaled by the call to tokenSource.Cancel below.
        DoSomeWork(2, token)
    Next
End Sub,
token)

Console.WriteLine("Task {0} executing", t.Id)
tasks.Add(t)

' Request cancellation from the UI thread.
Dim ch As Char = Console.ReadKey().KeyChar
If ch = "c"c Or ch = "C"c Then
    tokenSource.Cancel()
    Console.WriteLine(vbCrLf + "Task cancellation requested.")

    ' Optional: Observe the change in the Status property on the task.
    ' It is not necessary to wait on tasks that have canceled. However,
    ' if you do wait, you must enclose the call in a try-catch block to
    ' catch the TaskCanceledExceptions that are thrown. If you do
    ' not wait, no exception is thrown if the token that was passed to the
    ' StartNew method is the same token that requested the cancellation.
End If

Try
    Task.WaitAll(tasks.ToArray())
Catch e As AggregateException
    Console.WriteLine()
    Console.WriteLine("AggregateException thrown with the following inner exceptions:")
    ' Display information about each exception.
    For Each v In e.InnerExceptions
        If TypeOf v Is TaskCanceledException
            Console.WriteLine("  TaskCanceledException: Task {0}",
                DirectCast(v, TaskCanceledException).Task.Id)
        Else
            Console.WriteLine("  Exception: {0}", v.GetType().Name)
        End If
    Next
    Console.WriteLine()
Finally
    tokenSource.Dispose()
End Try

' Display status of all tasks.
For Each t In tasks
    Console.WriteLine("Task {0} status is now {1}", t.Id, t.Status)
Next
End Sub

Sub DoSomeWork(ByVal taskNum As Integer, ByVal ct As CancellationToken)
    ' Was cancellation already requested?
    If ct.IsCancellationRequested = True Then
        Console.WriteLine("Task {0} was cancelled before it got started.",
            taskNum)
        ct.ThrowIfCancellationRequested()
    End If
End Sub

```

```

    End If

    Dim maxIterations As Integer = 100

    ' NOTE!!! A "TaskCanceledException was unhandled
    ' by user code" error will be raised here if "Just My Code"
    ' is enabled on your computer. On Express editions JMC is
    ' enabled and cannot be disabled. The exception is benign.
    ' Just press F5 to continue executing your code.

    For i As Integer = 0 To maxIterations
        ' Do a bit of work. Not too much.
        Dim sw As New SpinWait()
        For j As Integer = 0 To 100
            sw.SpinOnce()
        Next
        If ct.IsCancellationRequested Then
            Console.WriteLine("Task {0} cancelled", taskNum)
            ct.ThrowIfCancellationRequested()
        End If
        Next
    End Sub
End Module

```

' The example displays output like the following:

' Press any key to begin tasks...

' To terminate the example, press 'c' to cancel and exit...

'

' Task 1 executing

' Task 2 executing

' Task 3 executing

' Task 4 executing

' Task 5 executing

' Task 6 executing

' Task 7 executing

' Task 8 executing

' c

' Task cancellation requested.

' Task 2 cancelled

' Task 7 cancelled

'

' AggregateException thrown with the following inner exceptions:

' TaskCanceledException: Task 2

' TaskCanceledException: Task 8

' TaskCanceledException: Task 7

'

' Task 2 status is now Canceled

' Task 1 status is now RanToCompletion

' Task 8 status is now Canceled

' Task 7 status is now Canceled

' Task 6 status is now RanToCompletion

' Task 5 status is now RanToCompletion

' Task 4 status is now RanToCompletion

' Task 3 status is now RanToCompletion

The [System.Threading.Tasks.Task](#) class is fully integrated with the cancellation model that is based on the [System.Threading.CancellationTokenSource](#) and [System.Threading.CancellationToken](#) types. For more information, see [Cancellation in Managed Threads](#) and [Task Cancellation](#).

See also

- [System.Threading.CancellationTokenSource](#)
- [System.Threading.CancellationToken](#)
- [System.Threading.Tasks.Task](#)
- [System.Threading.Tasks.Task<TResult>](#)
- [Task-based Asynchronous Programming](#)

- Attached and Detached Child Tasks
- Lambda Expressions in PLINQ and TPL

Create pre-computed tasks

9/20/2022 • 3 minutes to read • [Edit Online](#)

In this article, you'll learn how to use the `Task.FromResult` method to retrieve the results of asynchronous download operations that are held in a cache. The `FromResult` method returns a finished `Task<TResult>` object that holds the provided value as its `Result` property. This method is useful when you perform an asynchronous operation that returns a `Task<TResult>` object, and the result of that `Task<TResult>` object is already computed.

Example

The following example downloads strings from the web. It defines the `DownloadStringAsync` method. This method downloads strings from the web asynchronously. This example also uses a `ConcurrentDictionary< TKey, TValue >` object to cache the results of previous operations. If the input address is held in this cache, `DownloadStringAsync` uses the `FromResult` method to produce a `Task< TResult >` object that holds the content at that address. Otherwise, `DownloadStringAsync` downloads the file from the web and adds the result to the cache.

```
using System;
using System.Collections.Concurrent;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Net.Http;
using System.Threading.Tasks;

public static class DownloadCache
{
    private static readonly ConcurrentDictionary<string, string> s_cachedDownloads = new();
    private static readonly HttpClient s_httpClient = new();

    public static Task<string> DownloadStringAsync(string address)
    {
        if (s_cachedDownloads.TryGetValue(address, out string? content))
        {
            return Task.FromResult(content);
        }

        return Task.Run(async () =>
        {
            content = await s_httpClient.GetStringAsync(address);
            s_cachedDownloads.TryAdd(address, content);

            return content;
        });
    }

    public static async Task Main()
    {
        string[] urls = new[]
        {
            "https://docs.microsoft.com/aspnet/core",
            "https://docs.microsoft.com/dotnet",
            "https://docs.microsoft.com/dotnet/architecture/dapr-for-net-developers",
            "https://docs.microsoft.com/dotnet/azure",
            "https://docs.microsoft.com/dotnet/desktop/wpf",
            "https://docs.microsoft.com/dotnet/devops/create-dotnet-github-action",
            "https://docs.microsoft.com/dotnet/machine-learning",
            "https://docs.microsoft.com/xamarin",
            "https://dotnet.microsoft.com/"
        };
    }
}
```

```

    "https://dotnet.microsoft.com/",
    "https://www.microsoft.com"
};

Stopwatch stopwatch = Stopwatch.StartNew();
IEnumerable<Task<string>> downloads = urls.Select(DownloadStringAsync);

static void StopAndLogElapsedTime(
    int attemptNumber, Stopwatch stopwatch, Task<string[]> downloadTasks)
{
    stopwatch.Stop();

    int charCount = downloadTasks.Result.Sum(result => result.Length);
    long elapsedMs = stopwatch.ElapsedMilliseconds;

    Console.WriteLine(
        $"Attempt number: {attemptNumber}\n" +
        $"Retrieved characters: {charCount:#,0}\n" +
        $"Elapsed retrieval time: {elapsedMs:#,0} milliseconds.\n");
}

await Task.WhenAll(downloads).ContinueWith(
    downloadTasks => StopAndLogElapsedTime(1, stopwatch, downloadTasks));

// Perform the same operation a second time. The time required
// should be shorter because the results are held in the cache.
stopwatch.Restart();

downloads = urls.Select(DownloadStringAsync);

await Task.WhenAll(downloads).ContinueWith(
    downloadTasks => StopAndLogElapsedTime(2, stopwatch, downloadTasks));
}

// Sample output:
//   Attempt number: 1
//   Retrieved characters: 754,585
//   Elapsed retrieval time: 2,857 milliseconds.

//   Attempt number: 2
//   Retrieved characters: 754,585
//   Elapsed retrieval time: 1 milliseconds.
}

```

```

Imports System.Collections.Concurrent
Imports System.Net.Http

Module Snippets
    Class DownloadCache
        Private Shared ReadOnly s_cachedDownloads As ConcurrentDictionary(Of String, String) =
            New ConcurrentDictionary(Of String, String)()
        Private Shared ReadOnly s_httpClient As HttpClient = New HttpClient()

        Public Function DownloadStringAsync(address As String) As Task(Of String)
            Dim content As String = Nothing

            If s_cachedDownloads.TryGetValue(address, content) Then
                Return Task.FromResult(Of String)(content)
            End If

            Return Task.Run(
                Async Function()
                    content = Await s_httpClient.GetStringAsync(address)
                    s_cachedDownloads.TryAdd(address, content)
                    Return content
                End Function)
        End Function
    End Class

```

```

Public Sub StopAndLogElapsedTime(
    attemptNumber As Integer,
    stopwatch As Stopwatch,
    downloadTasks As Task(Of String()))

    stopwatch.Stop()

    Dim charCount As Integer = downloadTasks.Result.Sum(Function(result) result.Length)
    Dim elapsedMs As Long = stopwatch.ElapsedMilliseconds

    Console.WriteLine(
        $"Attempt number: {attemptNumber}{vbCrLf}" &
        $"Retrieved characters: {charCount:#,0}{vbCrLf}" &
        $"Elapsed retrieval time: {elapsedMs:#,0} milliseconds.{vbCrLf}")
End Sub

Sub Main()
    Dim cache As DownloadCache = New DownloadCache()
    Dim urls As String() = {
        "https://docs.microsoft.com/aspnet/core",
        "https://docs.microsoft.com/dotnet",
        "https://docs.microsoft.com/dotnet/architecture/dapr-for-net-developers",
        "https://docs.microsoft.com/dotnet/azure",
        "https://docs.microsoft.com/dotnet/desktop/wpf",
        "https://docs.microsoft.com/dotnet/devops/create-dotnet-github-action",
        "https://docs.microsoft.com/dotnet/machine-learning",
        "https://docs.microsoft.com/xamarin",
        "https://dotnet.microsoft.com/",
        "https://www.microsoft.com"
    }
    Dim stopwatch As Stopwatch = Stopwatch.StartNew()
    Dim downloads As IEnumerable(Of Task(Of String)) =
        urls.Select(AddressOf cache.DownloadStringAsync)
    Task.WhenAll(downloads).ContinueWith(
        Sub(downloadTasks)
            StopAndLogElapsedTime(1, stopwatch, downloadTasks)
        End Sub).Wait()

    stopwatch.Restart()
    downloads = urls.Select(AddressOf cache.DownloadStringAsync)
    Task.WhenAll(downloads).ContinueWith(
        Sub(downloadTasks)
            StopAndLogElapsedTime(2, stopwatch, downloadTasks)
        End Sub).Wait()
End Sub

' Sample output:
'   Attempt number 1
'   Retrieved characters: 754,585
'   Elapsed retrieval time: 2,857 milliseconds.

'
'   Attempt number 2
'   Retrieved characters: 754,585
'   Elapsed retrieval time: 1 milliseconds.
End Module

```

In the preceding example, the first time each url is downloaded, its value is stored in the cache. The [FromResult](#) method enables the `DownloadStringAsync` method to create `Task<TResult>` objects that hold these pre-computed results. Subsequent calls to download the string return the cached values, and is much faster.

See also

- [Task-based asynchronous programming](#)

How to: Traverse a Binary Tree with Parallel Tasks

9/20/2022 • 2 minutes to read • [Edit Online](#)

The following example shows two ways in which parallel tasks can be used to traverse a tree data structure. The creation of the tree itself is left as an exercise.

Example

```

public class TreeWalk
{
    static void Main()
    {
        Tree<MyClass> tree = new Tree<MyClass>();

        // ...populate tree (left as an exercise)

        // Define the Action to perform on each node.
        Action<MyClass> myAction = x => Console.WriteLine("{0} : {1}", x.Name, x.Number);

        // Traverse the tree with parallel tasks.
        DoTree(tree, myAction);
    }

    public class MyClass
    {
        public string Name { get; set; }
        public int Number { get; set; }
    }
    public class Tree<T>
    {
        public Tree<T> Left;
        public Tree<T> Right;
        public T Data;
    }

    // By using tasks explicitly.
    public static void DoTree<T>(Tree<T> tree, Action<T> action)
    {
        if (tree == null) return;
        var left = Task.Factory.StartNew(() => DoTree(tree.Left, action));
        var right = Task.Factory.StartNew(() => DoTree(tree.Right, action));
        action(tree.Data);

        try
        {
            Task.WaitAll(left, right);
        }
        catch (AggregateException )
        {
            //handle exceptions here
        }
    }

    // By using Parallel.Invoke
    public static void DoTree2<T>(Tree<T> tree, Action<T> action)
    {
        if (tree == null) return;
        Parallel.Invoke(
            () => DoTree2(tree.Left, action),
            () => DoTree2(tree.Right, action),
            () => action(tree.Data)
        );
    }
}

```

```

Imports System.Threading.Tasks

Public Class TreeWalk

    Shared Sub Main()

        Dim tree As Tree(Of Person) = New Tree(Of Person)()

        ' ...populate tree (left as an exercise)

        ' Define the Action to perform on each node.
        Dim myAction As Action(Of Person) = New Action(Of Person)(Sub(x)
                                                                Console.WriteLine("{0} : {1} ",
                                                                x.Name, x.Number))
                                                                End Sub)

        ' Traverse the tree with parallel tasks.
        DoTree(tree, myAction)
    End Sub

    Public Class Person
        Public Name As String
        Public Number As Integer
    End Class

    Public Class Tree(Of T)
        Public Left As Tree(Of T)
        Public Right As Tree(Of T)
        Public Data As T
    End Class

    ' By using tasks explicitly.
    Public Shared Sub DoTree(Of T)(ByVal myTree As Tree(Of T), ByVal a As Action(Of T))
        If myTree Is Nothing Then
            Return
        End If
        Dim left = Task.Factory.StartNew(Sub() DoTree(myTree.Left, a))
        Dim right = Task.Factory.StartNew(Sub() DoTree(myTree.Right, a))
        a(myTree.Data)

        Try
            Task.WaitAll(left, right)
        Catch ae As AggregateException
            'handle exceptions here
        End Try
    End Sub

    ' By using Parallel.Invoke
    Public Shared Sub DoTree2(Of T)(ByVal myTree As Tree(Of T), ByVal myAct As Action(Of T))
        If myTree Is Nothing Then
            Return
        End If
        Parallel.Invoke(
            Sub() DoTree2(myTree.Left, myAct),
            Sub() DoTree2(myTree.Right, myAct),
            Sub() myAct(myTree.Data)
        )
    End Sub
End Class

```

The two methods shown are functionally equivalent. By using the [StartNew](#) method to create and run the tasks, you get a handle back from the tasks which can be used to wait on the tasks and handle exceptions.

See also

- Task Parallel Library (TPL)

How to: Unwrap a Nested Task

9/20/2022 • 4 minutes to read • [Edit Online](#)

You can return a task from a method, and then wait on or continue from that task, as shown in the following example:

```
static Task<string> DoWorkAsync()
{
    return Task<String>.Factory.StartNew(() =>
    {
        //...
        return "Work completed.";
    });
}

static void StartTask()
{
    Task<String> t = DoWorkAsync();
    t.Wait();
    Console.WriteLine(t.Result);
}
```

```
Shared Function DoWorkAsync() As Task(Of String)

    Return Task(Of String).Run(Function()
        ...
        Return "Work completed."
    End Function)
End Function

Shared Sub StartTask()

    Dim t As Task(Of String) = DoWorkAsync()
    t.Wait()
    Console.WriteLine(t.Result)
End Sub
```

In the previous example, the `Result` property is of type `string` (`String` in Visual Basic).

However, in some scenarios, you might want to create a task within another task, and then return the nested task. In this case, the `TResult` of the enclosing task is itself a task. In the following example, the `Result` property is a `Task<Task<string>>` in C# or `Task(Of Task(Of String))` in Visual Basic.

```
// Note the type of t and t2.
Task<Task<string>> t = Task.Factory.StartNew(() => DoWorkAsync());
Task<Task<string>> t2 = DoWorkAsync().ContinueWith((s) => DoMoreWorkAsync());

// Outputs: System.Threading.Tasks.Task`1[System.String]
Console.WriteLine(t.Result);
```

```

' Note the type of t and t2.
Dim t As Task(Of Task(Of String)) = Task.Run(Function() DoWorkAsync())
Dim t2 As Task(Of Task(Of String)) = DoWorkAsync().ContinueWith(Function(s) DoMoreWorkAsync())

' Outputs: System.Threading.Tasks.Task`1[System.String]
Console.WriteLine(t.Result)

```

Although it is possible to write code to unwrap the outer task and retrieve the original task and its [Result](#) property, such code is not easy to write because you must handle exceptions and also cancellation requests. In this situation, we recommend that you use one of the [Unwrap](#) extension methods, as shown in the following example.

```

// Unwrap the inner task.
Task<string> t3 = DoWorkAsync().ContinueWith((s) => DoMoreWorkAsync()).Unwrap();

// Outputs "More work completed."
Console.WriteLine(t.Result);

```

```

' Unwrap the inner task.
Dim t3 As Task(Of String) = DoWorkAsync().ContinueWith(Function(s) DoMoreWorkAsync()).Unwrap()

' Outputs "More work completed."
Console.WriteLine(t.Result)

```

The [Unwrap](#) methods can be used to transform any `Task<Task>` or `Task<Task<TResult>>` (`Task(Of Task)` or `Task(Of Task(Of TResult))` in Visual Basic) to a `Task` or `Task<TResult>` (`Task(Of TResult)` in Visual Basic). The new task fully represents the inner nested task, and includes cancellation state and all exceptions.

Example

The following example demonstrates how to use the [Unwrap](#) extension methods.

```

namespace Unwrap
{
    using System;
    using System.Collections.Generic;
    using System.Linq;
    using System.Text;
    using System.Threading;
    using System.Threading.Tasks;
    // A program whose only use is to demonstrate Unwrap.
    class Program
    {
        static void Main()
        {
            // An arbitrary threshold value.
            byte threshold = 0x40;

            // data is a Task<byte[]>
            var data = Task<byte[]>.Factory.StartNew(() =>
            {
                return GetData();
            });

            // We want to return a task so that we can
            // continue from it later in the program.
            // Without Unwrap: stepTwo is a Task<Task<byte[]>>
            // With Unwrap: stepTwo is a Task<byte[]>
            var stepTwo = data.ContinueWith((antecedent) =>

```

```

    {
        return Task<byte>.Factory.StartNew( () => Compute(antecedent.Result));
    })
    .Unwrap();

    // Without Unwrap: antecedent.Result = Task<byte>
    // and the following method will not compile.
    // With Unwrap: antecedent.Result = byte and
    // we can work directly with the result of the Compute method.
    var lastStep = stepTwo.ContinueWith( (antecedent) =>
    {
        if (antecedent.Result >= threshold)
        {
            return Task.Factory.StartNew( () => Console.WriteLine("Program complete. Final =
0x{0:x} threshold = 0x{1:x}", stepTwo.Result, threshold));
        }
        else
        {
            return DoSomeOtherAsynchronousWork(stepTwo.Result, threshold);
        }
    });
}

lastStep.Wait();
Console.WriteLine("Press any key");
Console.ReadKey();
}

#region Dummy_Methods
private static byte[] GetData()
{
    Random rand = new Random();
    byte[] bytes = new byte[64];
    rand.NextBytes(bytes);
    return bytes;
}

static Task DoSomeOtherAsynchronousWork(int i, byte b2)
{
    return Task.Factory.StartNew(() =>
    {
        Thread.Sleep(500000);
        Console.WriteLine("Doing more work. Value was <= threshold");
    });
}
static byte Compute(byte[] data)
{
    byte final = 0;
    foreach (byte item in data)
    {
        final ^= item;
        Console.WriteLine("{0:x}", final);
    }
    Console.WriteLine("Done computing");
    return final;
}
#endregion
}
}

```

```

'How to: Unwrap a Task
Imports System.Threading
Imports System.Threading.Tasks

Module UnwrapATask2

Sub Main()

```

```

' An arbitrary threshold value.
Dim threshold As Byte = &H40

' myData is a Task(Of Byte())

Dim myData As Task(Of Byte()) = Task.Factory.StartNew(Function()
                                                       Return GetData()
                                                       End Function)

' We want to return a task so that we can
' continue from it later in the program.
' Without Unwrap: stepTwo is a Task(Of Task(Of Byte()))
' With Unwrap: stepTwo is a Task(Of Byte)

Dim stepTwo = myData.ContinueWith(Function(antecedent)
                                    Return Task.Factory.StartNew(Function()
                                                               Return
Compute(antecedent.Result)
                                                               End Function)
                                    End Function).Unwrap()

Dim lastStep = stepTwo.ContinueWith(Function(antecedent)
                                      Console.WriteLine("Result = {0}", antecedent.Result)
                                      If antecedent.Result >= threshold Then
                                        Return Task.Factory.StartNew(Sub()
                                                              
Console.WriteLine("Program complete. Final = &H{1:x} threshold = &H{1:x}",
stepTwo.Result, threshold)
                                                               End Sub)
                                      Else
                                        Return DoSomeOtherAsynchronousWork(stepTwo.Result,
threshold)
                                      End If
                                      End Function)

Try
  lastStep.Wait()
Catch ae As AggregateException
  For Each ex As Exception In ae.InnerExceptions
    Console.WriteLine(ex.Message & ex.StackTrace & ex.GetBaseException.ToString())
  Next
End Try

Console.WriteLine("Press any key")
Console.ReadKey()
End Sub

#Region "Dummy_Methods"
Function GetData() As Byte()
  Dim rand As Random = New Random()
  Dim bytes(64) As Byte
  rand.NextBytes(bytes)
  Return bytes
End Function

Function DoSomeOtherAsynchronousWork(ByVal i As Integer, ByVal b2 As Byte) As Task
  Return Task.Factory.StartNew(Sub()
                               Thread.Sleep(500000)
                               Console.WriteLine("Doing more work. Value was <= threshold.")
                               End Sub)
End Function

Function Compute(ByVal d As Byte()) As Byte
  Dim final As Byte = 0
  For Each item As Byte In d
    final = final Xor item
    Console.WriteLine("{0:x}", final)
  Next
  Console.WriteLine("Done computing")
  Return final
End Function

```

```
    End Function  
#End Region  
End Module
```

See also

- [System.Threading.Tasks.TaskExtensions](#)
- [Task-based Asynchronous Programming](#)

How to: Prevent a Child Task from Attaching to its Parent

9/20/2022 • 5 minutes to read • [Edit Online](#)

This document demonstrates how to prevent a child task from attaching to the parent task. Preventing a child task from attaching to its parent is useful when you call a component that is written by a third party and that also uses tasks. For example, a third-party component that uses the [TaskCreationOptions.AttachedToParent](#) option to create a [Task](#) or [Task<TResult>](#) object can cause problems in your code if it is long-running or throws an unhandled exception.

Example

The following example compares the effects of using the default options to the effects of preventing a child task from attaching to the parent. The example creates a [Task](#) object that calls into a third-party library that also uses a [Task](#) object. The third-party library uses the [AttachedToParent](#) option to create the [Task](#) object. The application uses the [TaskCreationOptions.DenyChildAttach](#) option to create the parent task. This option instructs the runtime to remove the [AttachedToParent](#) specification in child tasks.

```
using System;
using System.Diagnostics;
using System.Threading;
using System.Threading.Tasks;

// Defines functionality that is provided by a third-party.
// In a real-world scenario, this would likely be provided
// in a separate code file or assembly.
namespace Contoso
{
    public class Widget
    {
        public Task Run()
        {
            // Create a long-running task that is attached to the
            // parent in the task hierarchy.
            return Task.Factory.StartNew(() =>
            {
                // Simulate a lengthy operation.
                Thread.Sleep(5000);
            }, TaskCreationOptions.AttachedToParent);
        }
    }
}

// Demonstrates how to prevent a child task from attaching to the parent.
class DenyChildAttach
{
    static void RunWidget(Contoso.Widget widget,
        TaskCreationOptions parentTaskOptions)
    {
        // Record the time required to run the parent
        // and child tasks.
        Stopwatch stopwatch = new Stopwatch();
        stopwatch.Start();

        Console.WriteLine("Starting widget as a background task...");

        // Run the widget task in the background.
    }
}
```

```

Task<Task> runWidget = Task.Factory.StartNew(() =>
{
    Task widgetTask = widget.Run();

    // Perform other work while the task runs...
    Thread.Sleep(1000);

    return widgetTask;
}, parentTaskOptions);

// Wait for the parent task to finish.
Console.WriteLine("Waiting for parent task to finish...");
runWidget.Wait();
Console.WriteLine("Parent task has finished. Elapsed time is {0} ms.",
    stopwatch.ElapsedMilliseconds);

// Perform more work...
Console.WriteLine("Performing more work on the main thread...");
Thread.Sleep(2000);
Console.WriteLine("Elapsed time is {0} ms.", stopwatch.ElapsedMilliseconds);

// Wait for the child task to finish.
Console.WriteLine("Waiting for child task to finish...");
runWidget.Result.Wait();
Console.WriteLine("Child task has finished. Elapsed time is {0} ms.",
    stopwatch.ElapsedMilliseconds);
}

static void Main(string[] args)
{
    Contoso.Widget w = new Contoso.Widget();

    // Perform the same operation two times. The first time, the operation
    // is performed by using the default task creation options. The second
    // time, the operation is performed by using the DenyChildAttach option
    // in the parent task.

    Console.WriteLine("Demonstrating parent/child tasks with default options...");
    RunWidget(w, TaskCreationOptions.None);

    Console.WriteLine();

    Console.WriteLine("Demonstrating parent/child tasks with the DenyChildAttach option...");
    RunWidget(w, TaskCreationOptions.DenyChildAttach);
}
}

/* Sample output:
Demonstrating parent/child tasks with default options...
Starting widget as a background task...
Waiting for parent task to finish...
Parent task has finished. Elapsed time is 5014 ms.
Performing more work on the main thread...
Elapsed time is 7019 ms.
Waiting for child task to finish...
Child task has finished. Elapsed time is 7019 ms.

Demonstrating parent/child tasks with the DenyChildAttach option...
Starting widget as a background task...
Waiting for parent task to finish...
Parent task has finished. Elapsed time is 1007 ms.
Performing more work on the main thread...
Elapsed time is 3015 ms.
Waiting for child task to finish...
Child task has finished. Elapsed time is 5015 ms.
*/

```

```

Imports System.Diagnostics
Imports System.Threading
Imports System.Threading.Tasks

' Defines functionality that is provided by a third-party.
' In a real-world scenario, this would likely be provided
' in a separate code file or assembly.
Namespace Contoso
    Public Class Widget
        Public Function Run() As Task
            ' Create a long-running task that is attached to the
            ' parent in the task hierarchy.
            Return Task.Factory.StartNew(Sub() Thread.Sleep(5000), TaskCreationOptions.AttachedToParent)
            ' Simulate a lengthy operation.
        End Function
    End Class
End Namespace

' Demonstrates how to prevent a child task from attaching to the parent.
Friend Class DenyChildAttach
    Private Shared Sub RunWidget(ByVal widget As Contoso.Widget, ByVal parentTaskOptions As
TaskCreationOptions)
        ' Record the time required to run the parent
        ' and child tasks.
        Dim stopwatch As New Stopwatch()
        stopwatch.Start()

        Console.WriteLine("Starting widget as a background task...")

        ' Run the widget task in the background.
        Dim runWidget As Task(Of Task) = Task.Factory.StartNew(Function()
            ' Perform other work while the task
        runs...
            Dim widgetTask As Task = widget.Run()
            Thread.Sleep(1000)
            Return widgetTask
        End Function, parentTaskOptions)

        ' Wait for the parent task to finish.
        Console.WriteLine("Waiting for parent task to finish...")
        runWidget.Wait()
        Console.WriteLine("Parent task has finished. Elapsed time is {0} ms.",
stopwatch.ElapsedMilliseconds)

        ' Perform more work...
        Console.WriteLine("Performing more work on the main thread...")
        Thread.Sleep(2000)
        Console.WriteLine("Elapsed time is {0} ms.", stopwatch.ElapsedMilliseconds)

        ' Wait for the child task to finish.
        Console.WriteLine("Waiting for child task to finish...")
        runWidget.Result.Wait()
        Console.WriteLine("Child task has finished. Elapsed time is {0} ms.", stopwatch.ElapsedMilliseconds)
    End Sub

    Shared Sub Main(ByVal args() As String)
        Dim w As New Contoso.Widget()

        ' Perform the same operation two times. The first time, the operation
        ' is performed by using the default task creation options. The second
        ' time, the operation is performed by using the DenyChildAttach option
        ' in the parent task.

        Console.WriteLine("Demonstrating parent/child tasks with default options...")
        RunWidget(w, TaskCreationOptions.None)

        Console.WriteLine()

        Console.WriteLine("Demonstrating parent/child tasks with the DenyChildAttach option...")
    End Sub
End Class

```

```
RunWidget(w, TaskCreationOptions.DenyChildAttach)
End Sub
End Class

' Sample output:
'Demonstrating parent/child tasks with default options...
'Starting widget as a background task...
'Waiting for parent task to finish...
'Parent task has finished. Elapsed time is 5014 ms.
'Performing more work on the main thread...
'Elapsed time is 7019 ms.
'Waiting for child task to finish...
'Child task has finished. Elapsed time is 7019 ms.
'

'Demonstrating parent/child tasks with the DenyChildAttach option...
'Starting widget as a background task...
'Waiting for parent task to finish...
'Parent task has finished. Elapsed time is 1007 ms.
'Performing more work on the main thread...
'Elapsed time is 3015 ms.
'Waiting for child task to finish...
'Child task has finished. Elapsed time is 5015 ms.
'
```

Because a parent task does not finish until all child tasks finish, a long-running child task can cause the overall application to perform poorly. In this example, when the application uses the default options to create the parent task, the child task must finish before the parent task finishes. When the application uses the [TaskCreationOptions.DenyChildAttach](#) option, the child is not attached to the parent. Therefore, the application can perform additional work after the parent task finishes and before it must wait for the child task to finish.

See also

- [Task-based Asynchronous Programming](#)

Dataflow (Task Parallel Library)

9/20/2022 • 37 minutes to read • [Edit Online](#)

The Task Parallel Library (TPL) provides dataflow components to help increase the robustness of concurrency-enabled applications. These dataflow components are collectively referred to as the *TPL Dataflow Library*. This dataflow model promotes actor-based programming by providing in-process message passing for coarse-grained dataflow and pipelining tasks. The dataflow components build on the types and scheduling infrastructure of the TPL and integrate with the C#, Visual Basic, and F# language support for asynchronous programming. These dataflow components are useful when you have multiple operations that must communicate with one another asynchronously or when you want to process data as it becomes available. For example, consider an application that processes image data from a web camera. By using the dataflow model, the application can process image frames as they become available. If the application enhances image frames, for example, by performing light correction or red-eye reduction, you can create a *pipeline* of dataflow components. Each stage of the pipeline might use more coarse-grained parallelism functionality, such as the functionality that is provided by the TPL, to transform the image.

This document provides an overview of the TPL Dataflow Library. It describes the programming model, the predefined dataflow block types, and how to configure dataflow blocks to meet the specific requirements of your applications.

NOTE

The TPL Dataflow Library (the `System.Threading.Tasks.Dataflow` namespace) is not distributed with .NET. To install the `System.Threading.Tasks.Dataflow` namespace in Visual Studio, open your project, choose **Manage NuGet Packages** from the **Project** menu, and search online for the `System.Threading.Tasks.Dataflow` package. Alternatively, to install it using the [.NET Core CLI](#), run `dotnet add package System.Threading.Tasks.Dataflow`.

Programming Model

The TPL Dataflow Library provides a foundation for message passing and parallelizing CPU-intensive and I/O-intensive applications that have high throughput and low latency. It also gives you explicit control over how data is buffered and moves around the system. To better understand the dataflow programming model, consider an application that asynchronously loads images from disk and creates a composite of those images. Traditional programming models typically require that you use callbacks and synchronization objects, such as locks, to coordinate tasks and access to shared data. By using the dataflow programming model, you can create dataflow objects that process images as they are read from disk. Under the dataflow model, you declare how data is handled when it becomes available, and also any dependencies between data. Because the runtime manages dependencies between data, you can often avoid the requirement to synchronize access to shared data. In addition, because the runtime schedules work based on the asynchronous arrival of data, dataflow can improve responsiveness and throughput by efficiently managing the underlying threads. For an example that uses the dataflow programming model to implement image processing in a Windows Forms application, see [Walkthrough: Using Dataflow in a Windows Forms Application](#).

Sources and Targets

The TPL Dataflow Library consists of *dataflow blocks*, which are data structures that buffer and process data. The TPL defines three kinds of dataflow blocks: *source blocks*, *target blocks*, and *propagator blocks*. A source block acts as a source of data and can be read from. A target block acts as a receiver of data and can be written to. A propagator block acts as both a source block and a target block, and can be read from and written to. The TPL defines the `System.Threading.Tasks.Dataflow.ISourceBlock<TOutput>` interface to represent sources,

`System.Threading.Tasks.Dataflow.ITargetBlock<TInput>` to represent targets, and
`System.Threading.Tasks.Dataflow.IPropagatorBlock<TInput,TOutput>` to represent propagators.
`IPropagatorBlock<TInput,TOutput>` inherits from both `ISourceBlock<TOutput>`, and `ITargetBlock<TInput>`.

The TPL Dataflow Library provides several predefined dataflow block types that implement the `ISourceBlock<TOutput>`, `ITargetBlock<TInput>`, and `IPropagatorBlock<TInput,TOutput>` interfaces. These dataflow block types are described in this document in the section [Predefined Dataflow Block Types](#).

Connecting Blocks

You can connect dataflow blocks to form *pipelines*, which are linear sequences of dataflow blocks, or *networks*, which are graphs of dataflow blocks. A pipeline is one form of network. In a pipeline or network, sources asynchronously propagate data to targets as that data becomes available. The `ISourceBlock<TOutput>.LinkTo` method links a source dataflow block to a target block. A source can be linked to zero or more targets; targets can be linked from zero or more sources. You can add or remove dataflow blocks to or from a pipeline or network concurrently. The predefined dataflow block types handle all thread-safety aspects of linking and unlinking.

For an example that connects dataflow blocks to form a basic pipeline, see [Walkthrough: Creating a Dataflow Pipeline](#). For an example that connects dataflow blocks to form a more complex network, see [Walkthrough: Using Dataflow in a Windows Forms Application](#). For an example that unlinks a target from a source after the source offers the target a message, see [How to: Unlink Dataflow Blocks](#).

Filtering

When you call the `ISourceBlock<TOutput>.LinkTo` method to link a source to a target, you can supply a delegate that determines whether the target block accepts or rejects a message based on the value of that message. This filtering mechanism is a useful way to guarantee that a dataflow block receives only certain values. For most of the predefined dataflow block types, if a source block is connected to multiple target blocks, when a target block rejects a message, the source offers that message to the next target. The order in which a source offers messages to targets is defined by the source and can vary according to the type of the source. Most source block types stop offering a message after one target accepts that message. One exception to this rule is the `BroadcastBlock<T>` class, which offers each message to all targets, even if some targets reject the message. For an example that uses filtering to process only certain messages, see [Walkthrough: Using Dataflow in a Windows Forms Application](#).

IMPORTANT

Because each predefined source dataflow block type guarantees that messages are propagated out in the order in which they are received, every message must be read from the source block before the source block can process the next message. Therefore, when you use filtering to connect multiple targets to a source, make sure that at least one target block receives each message. Otherwise, your application might deadlock.

Message Passing

The dataflow programming model is related to the concept of *message passing*, where independent components of a program communicate with one another by sending messages. One way to propagate messages among application components is to call the `Post` and `DataflowBlock.SendAsync` methods to send messages to target dataflow blocks post (`Post` acts synchronously; `SendAsync` acts asynchronously) and the `Receive`, `ReceiveAsync`, and `TryReceive` methods to receive messages from source blocks. You can combine these methods with dataflow pipelines or networks by sending input data to the head node (a target block), and receiving output data from the terminal node of the pipeline or the terminal nodes of the network (one or more source blocks). You can also use the `Choose` method to read from the first of the provided sources that has data available and perform action on that data.

Source blocks offer data to target blocks by calling the `ITargetBlock<TInput>.OfferMessage` method. The target block responds to an offered message in one of three ways: it can accept the message, decline the message, or

postpone the message. When the target accepts the message, the [OfferMessage](#) method returns [Accepted](#). When the target declines the message, the [OfferMessage](#) method returns [Declined](#). When the target requires that it no longer receives any messages from the source, [OfferMessage](#) returns [DecliningPermanently](#). The predefined source block types do not offer messages to linked targets after such a return value is received, and they automatically unlink from such targets.

When a target block postpones the message for later use, the [OfferMessage](#) method returns [Postponed](#). A target block that postpones a message can later call the [ISourceBlock<TOutput>.ReserveMessage](#) method to try to reserve the offered message. At this point, the message is either still available and can be used by the target block, or the message has been taken by another target. When the target block later requires the message or no longer needs the message, it calls the [ISourceBlock<TOutput>.ConsumeMessage](#) or [ReleaseReservation](#) method, respectively. Message reservation is typically used by the dataflow block types that operate in non-greedy mode. Non-greedy mode is explained later in this document. Instead of reserving a postponed message, a target block can also use the [ISourceBlock<TOutput>.ConsumeMessage](#) method to attempt to directly consume the postponed message.

Dataflow Block Completion

Dataflow blocks also support the concept of *completion*. A dataflow block that is in the completed state does not perform any further work. Each dataflow block has an associated [System.Threading.Tasks.Task](#) object, known as a *completion task*, that represents the completion status of the block. Because you can wait for a [Task](#) object to finish, by using completion tasks, you can wait for one or more terminal nodes of a dataflow network to finish. The [IDataflowBlock](#) interface defines the [Complete](#) method, which informs the dataflow block of a request for it to complete, and the [Completion](#) property, which returns the completion task for the dataflow block. Both [ISourceBlock<TOutput>](#) and [ITargetBlock<TInput>](#) inherit the [IDataflowBlock](#) interface.

There are two ways to determine whether a dataflow block completed without error, encountered one or more errors, or was canceled. The first way is to call the [Task.Wait](#) method on the completion task in a `try - catch` block (`Try - Catch` in Visual Basic). The following example creates an [ActionBlock<TInput>](#) object that throws [ArgumentOutOfRangeException](#) if its input value is less than zero. [AggregateException](#) is thrown when this example calls [Wait](#) on the completion task. The [ArgumentOutOfRangeException](#) is accessed through the [InnerExceptions](#) property of the [AggregateException](#) object.

```
// Create an ActionBlock<int> object that prints its input
// and throws ArgumentException if the input
// is less than zero.
var throwIfNegative = new ActionBlock<int>(n =>
{
    Console.WriteLine("n = {0}", n);
    if (n < 0)
    {
        throw new ArgumentException();
    }
});

// Post values to the block.
throwIfNegative.Post(0);
throwIfNegative.Post(-1);
throwIfNegative.Post(1);
throwIfNegative.Post(-2);
throwIfNegative.Complete();

// Wait for completion in a try/catch block.
try
{
    throwIfNegative.Completion.Wait();
}
catch (AggregateException ae)
{
    // If an unhandled exception occurs during dataflow processing, all
    // exceptions are propagated through an AggregateException object.
    ae.Handle(e =>
    {
        Console.WriteLine("Encountered {0}: {1}",
            e.GetType().Name, e.Message);
        return true;
    });
}

/* Output:
n = 0
n = -1
Encountered ArgumentException: Specified argument was out of the range
of valid values.
*/
```

```

' Create an ActionBlock<int> object that prints its input
' and throws ArgumentOutOfRangeException if the input
' is less than zero.
Dim throwIfNegative = New ActionBlock(Of Integer)(Sub(n)
    Console.WriteLine("n = {0}", n)
    If n < 0 Then
        Throw New ArgumentOutOfRangeException()
    End If
End Sub)

' Post values to the block.
throwIfNegative.Post(0)
throwIfNegative.Post(-1)
throwIfNegative.Post(1)
throwIfNegative.Post(-2)
throwIfNegative.Complete()

' Wait for completion in a try/catch block.
Try
    throwIfNegative.Completion.Wait()
Catch ae As AggregateException
    ' If an unhandled exception occurs during dataflow processing, all
    ' exceptions are propagated through an AggregateException object.
    ae.Handle(Function(e)
        Console.WriteLine("Encountered {0}: {1}", e.GetType().Name, e.Message)
        Return True
    End Function)
End Try

' Output:
' n = 0
' n = -1
' Encountered ArgumentOutOfRangeException: Specified argument was out of the range
' of valid values.
'
```

This example demonstrates the case in which an exception goes unhandled in the delegate of an execution dataflow block. We recommend that you handle exceptions in the bodies of such blocks. However, if you are unable to do so, the block behaves as though it was canceled and does not process incoming messages.

When a dataflow block is canceled explicitly, the [AggregateException](#) object contains [OperationCanceledException](#) in the [InnerExceptions](#) property. For more information about dataflow cancellation, see [Enabling Cancellation](#) section.

The second way to determine the completion status of a dataflow block is to use a continuation of the completion task, or to use the asynchronous language features of C# and Visual Basic to asynchronously wait for the completion task. The delegate that you provide to the [Task.ContinueWith](#) method takes a [Task](#) object that represents the antecedent task. In the case of the [Completion](#) property, the delegate for the continuation takes the completion task itself. The following example resembles the previous one, except that it also uses the [ContinueWith](#) method to create a continuation task that prints the status of the overall dataflow operation.

```

// Create an ActionBlock<int> object that prints its input
// and throws ArgumentException if the input
// is less than zero.
var throwIfNegative = new ActionBlock<int>(n =>
{
    Console.WriteLine("n = {0}", n);
    if (n < 0)
    {
        throw new ArgumentException();
    }
});

// Create a continuation task that prints the overall
// task status to the console when the block finishes.
throwIfNegative.Completion.ContinueWith(task =>
{
    Console.WriteLine("The status of the completion task is '{0}'.",
        task.Status);
});

// Post values to the block.
throwIfNegative.Post(0);
throwIfNegative.Post(-1);
throwIfNegative.Post(1);
throwIfNegative.Post(-2);
throwIfNegative.Complete();

// Wait for completion in a try/catch block.
try
{
    throwIfNegative.Completion.Wait();
}
catch (AggregateException ae)
{
    // If an unhandled exception occurs during dataflow processing, all
    // exceptions are propagated through an AggregateException object.
    ae.Handle(e =>
    {
        Console.WriteLine("Encountered {0}: {1}",
            e.GetType().Name, e.Message);
        return true;
    });
}

/* Output:
n = 0
n = -1
The status of the completion task is 'Faulted'.
Encountered ArgumentException: Specified argument was out of the range
of valid values.
*/

```

```

' Create an ActionBlock<int> object that prints its input
' and throws ArgumentOutOfRangeException if the input
' is less than zero.
Dim throwIfNegative = New ActionBlock(Of Integer)(Sub(n)
    Console.WriteLine("n = {0}", n)
    If n < 0 Then
        Throw New ArgumentOutOfRangeException()
    End If
End Sub)

' Create a continuation task that prints the overall
' task status to the console when the block finishes.
throwIfNegative.Completion.ContinueWith(Sub(task) Console.WriteLine("The status of the completion task is
'{0}'.", task.Status))

' Post values to the block.
throwIfNegative.Post(0)
throwIfNegative.Post(-1)
throwIfNegative.Post(1)
throwIfNegative.Post(-2)
throwIfNegative.Complete()

' Wait for completion in a try/catch block.
Try
    throwIfNegative.Completion.Wait()
Catch ae As AggregateException
    ' If an unhandled exception occurs during dataflow processing, all
    ' exceptions are propagated through an AggregateException object.
    ae.Handle(Function(e)
        Console.WriteLine("Encountered {0}: {1}", e.GetType().Name, e.Message)
        Return True
    End Function)
End Try

'      Output:
'      n = 0
'      n = -1
'      The status of the completion task is 'Faulted'.
'      Encountered ArgumentOutOfRangeException: Specified argument was out of the range
'      of valid values.
'

```

You can also use properties such as [IsCanceled](#) in the body of the continuation task to determine additional information about the completion status of a dataflow block. For more information about continuation tasks and how they relate to cancellation and error handling, see [Chaining Tasks by Using Continuation Tasks](#), [Task Cancellation](#), and [Exception Handling](#).

Predefined Dataflow Block Types

The TPL Dataflow Library provides several predefined dataflow block types. These types are divided into three categories: *buffering blocks*, *execution blocks*, and *grouping blocks*. The following sections describe the block types that make up these categories.

Buffering Blocks

Buffering blocks hold data for use by data consumers. The TPL Dataflow Library provides three buffering block types: [System.Threading.Tasks.Dataflow.BufferBlock<T>](#), [System.Threading.Tasks.Dataflow.BroadcastBlock<T>](#), and [System.Threading.Tasks.Dataflow.WriteOnceBlock<T>](#).

[BufferBlock\(T\)](#)

The [BufferBlock<T>](#) class represents a general-purpose asynchronous messaging structure. This class stores a first in, first out (FIFO) queue of messages that can be written to by multiple sources or read from by multiple targets. When a target receives a message from a [BufferBlock<T>](#) object, that message is removed from the

message queue. Therefore, although a [BufferBlock<T>](#) object can have multiple targets, only one target will receive each message. The [BufferBlock<T>](#) class is useful when you want to pass multiple messages to another component, and that component must receive each message.

The following basic example posts several [Int32](#) values to a [BufferBlock<T>](#) object and then reads those values back from that object.

```
// Create a BufferBlock<int> object.  
var bufferBlock = new BufferBlock<int>();  
  
// Post several messages to the block.  
for (int i = 0; i < 3; i++)  
{  
    bufferBlock.Post(i);  
}  
  
// Receive the messages back from the block.  
for (int i = 0; i < 3; i++)  
{  
    Console.WriteLine(bufferBlock.Receive());  
}  
  
/* Output:  
 0  
 1  
 2  
 */
```

```
' Create a BufferBlock<int> object.  
Dim bufferBlock = New BufferBlock(Of Integer)()  
  
' Post several messages to the block.  
For i As Integer = 0 To 2  
    bufferBlock.Post(i)  
Next i  
  
' Receive the messages back from the block.  
For i As Integer = 0 To 2  
    Console.WriteLine(bufferBlock.Receive())  
Next i  
  
' Output:  
'     0  
'     1  
'     2  
'
```

For a complete example that demonstrates how to write messages to and read messages from a [BufferBlock<T>](#) object, see [How to: Write Messages to and Read Messages from a Dataflow Block](#).

BroadcastBlock(T)

The [BroadcastBlock<T>](#) class is useful when you must pass multiple messages to another component, but that component needs only the most recent value. This class is also useful when you want to broadcast a message to multiple components.

The following basic example posts a [Double](#) value to a [BroadcastBlock<T>](#) object and then reads that value back from that object several times. Because values are not removed from [BroadcastBlock<T>](#) objects after they are read, the same value is available every time.

```

// Create a BroadcastBlock<double> object.
var broadcastBlock = new BroadcastBlock<double>(null);

// Post a message to the block.
broadcastBlock.Post(Math.PI);

// Receive the messages back from the block several times.
for (int i = 0; i < 3; i++)
{
    Console.WriteLine(broadcastBlock.Receive());
}

/* Output:
   3.14159265358979
   3.14159265358979
   3.14159265358979
*/

```

```

' Create a BroadcastBlock<double> object.
Dim broadcastBlock = New BroadcastBlock(Of Double)(Nothing)

' Post a message to the block.
broadcastBlock.Post(Math.PI)

' Receive the messages back from the block several times.
For i As Integer = 0 To 2
    Console.WriteLine(broadcastBlock.Receive())
Next i

'       Output:
'       3.14159265358979
'       3.14159265358979
'       3.14159265358979
'

```

For a complete example that demonstrates how to use [BroadcastBlock<T>](#) to broadcast a message to multiple target blocks, see [How to: Specify a Task Scheduler in a Dataflow Block](#).

WriteOnceBlock<T>

The [WriteOnceBlock<T>](#) class resembles the [BroadcastBlock<T>](#) class, except that a [WriteOnceBlock<T>](#) object can be written to one time only. You can think of [WriteOnceBlock<T>](#) as being similar to the C# `readonly` (`ReadOnly` in Visual Basic) keyword, except that a [WriteOnceBlock<T>](#) object becomes immutable after it receives a value instead of at construction. Like the [BroadcastBlock<T>](#) class, when a target receives a message from a [WriteOnceBlock<T>](#) object, that message is not removed from that object. Therefore, multiple targets receive a copy of the message. The [WriteOnceBlock<T>](#) class is useful when you want to propagate only the first of multiple messages.

The following basic example posts multiple [String](#) values to a [WriteOnceBlock<T>](#) object and then reads the value back from that object. Because a [WriteOnceBlock<T>](#) object can be written to one time only, after a [WriteOnceBlock<T>](#) object receives a message, it discards subsequent messages.

```

// Create a WriteOnceBlock<string> object.
var writeOnceBlock = new WriteOnceBlock<string>(null);

// Post several messages to the block in parallel. The first
// message to be received is written to the block.
// Subsequent messages are discarded.
Parallel.Invoke(
    () => writeOnceBlock.Post("Message 1"),
    () => writeOnceBlock.Post("Message 2"),
    () => writeOnceBlock.Post("Message 3"));

// Receive the message from the block.
Console.WriteLine(writeOnceBlock.Receive());

/* Sample output:
   Message 2
*/

```

```

' Create a WriteOnceBlock<string> object.
Dim writeOnceBlock = New WriteOnceBlock(Of String)(Nothing)

' Post several messages to the block in parallel. The first
' message to be received is written to the block.
' Subsequent messages are discarded.
Parallel.Invoke(Function() writeOnceBlock.Post("Message 1"), Function() writeOnceBlock.Post("Message 2"),
Function() writeOnceBlock.Post("Message 3"))

' Receive the message from the block.
Console.WriteLine(writeOnceBlock.Receive())

'       Sample output:
'       Message 2
'

```

For a complete example that demonstrates how to use [WriteOnceBlock<T>](#) to receive the value of the first operation that finishes, see [How to: Unlink Dataflow Blocks](#).

Execution Blocks

Execution blocks call a user-provided delegate for each piece of received data. The TPL Dataflow Library provides three execution block types: [ActionBlock<TInput>](#), [System.Threading.Tasks.Dataflow.TransformBlock<TInput,TOutput>](#), and [System.Threading.Tasks.Dataflow.TransformManyBlock<TInput,TOutput>](#).

ActionBlock(T)

The [ActionBlock<TInput>](#) class is a target block that calls a delegate when it receives data. Think of a [ActionBlock<TInput>](#) object as a delegate that runs asynchronously when data becomes available. The delegate that you provide to an [ActionBlock<TInput>](#) object can be of type [Action<T>](#) or type [System.Func<TInput, Task>](#). When you use an [ActionBlock<TInput>](#) object with [Action<T>](#), processing of each input element is considered completed when the delegate returns. When you use an [ActionBlock<TInput>](#) object with [System.Func<TInput, Task>](#), processing of each input element is considered completed only when the returned [Task](#) object is completed. By using these two mechanisms, you can use [ActionBlock<TInput>](#) for both synchronous and asynchronous processing of each input element.

The following basic example posts multiple [Int32](#) values to an [ActionBlock<TInput>](#) object. The [ActionBlock<TInput>](#) object prints those values to the console. This example then sets the block to the completed state and waits for all dataflow tasks to finish.

```

// Create an ActionBlock<int> object that prints values
// to the console.
var actionBlock = new ActionBlock<int>(n => Console.WriteLine(n));

// Post several messages to the block.
for (int i = 0; i < 3; i++)
{
    actionBlock.Post(i * 10);
}

// Set the block to the completed state and wait for all
// tasks to finish.
actionBlock.Complete();
actionBlock.Completion.Wait();

/* Output:
0
10
20
*/

```

```

' Create an ActionBlock<int> object that prints values
' to the console.
Dim actionBlock = New ActionBlock(Of Integer)(Function(n) WriteLine(n))

' Post several messages to the block.
For i As Integer = 0 To 2
    actionBlock.Post(i * 10)
Next i

' Set the block to the completed state and wait for all
' tasks to finish.
actionBlock.Complete()
actionBlock.Completion.Wait()

'       Output:
'       0
'       10
'       20
'

```

For complete examples that demonstrate how to use delegates with the [ActionBlock<TInput>](#) class, see [How to: Perform Action When a Dataflow Block Receives Data](#).

TransformBlock<TInput, TOutput>

The [TransformBlock<TInput,TOutput>](#) class resembles the [ActionBlock<TInput>](#) class, except that it acts as both a source and as a target. The delegate that you pass to a [TransformBlock<TInput,TOutput>](#) object returns a value of type `TOutput`. The delegate that you provide to a [TransformBlock<TInput,TOutput>](#) object can be of type `System.Func<TInput, TOutput>` or type `System.Func<TInput, Task<TOutput>>`. When you use a [TransformBlock<TInput,TOutput>](#) object with `System.Func<TInput, TOutput>`, processing of each input element is considered completed when the delegate returns. When you use a [TransformBlock<TInput,TOutput>](#) object used with `System.Func<TInput, Task<TOutput>>`, processing of each input element is considered completed only when the returned `Task<TResult>` object is completed. As with [ActionBlock<TInput>](#), by using these two mechanisms, you can use [TransformBlock<TInput,TOutput>](#) for both synchronous and asynchronous processing of each input element.

The following basic example creates a [TransformBlock<TInput,TOutput>](#) object that computes the square root of its input. The [TransformBlock<TInput,TOutput>](#) object takes `Int32` values as input and produces `Double` values as output.

```

// Create a TransformBlock<int, double> object that
// computes the square root of its input.
var transformBlock = new TransformBlock<int, double>(n => Math.Sqrt(n));

// Post several messages to the block.
transformBlock.Post(10);
transformBlock.Post(20);
transformBlock.Post(30);

// Read the output messages from the block.
for (int i = 0; i < 3; i++)
{
    Console.WriteLine(transformBlock.Receive());
}

/* Output:
   3.16227766016838
   4.47213595499958
   5.47722557505166
*/

```

```

' Create a TransformBlock<int, double> object that
' computes the square root of its input.
Dim transformBlock = New TransformBlock(Of Integer, Double)(Function(n) Math.Sqrt(n))

' Post several messages to the block.
transformBlock.Post(10)
transformBlock.Post(20)
transformBlock.Post(30)

' Read the output messages from the block.
For i As Integer = 0 To 2
    Console.WriteLine(transformBlock.Receive())
Next i

'       Output:
'       3.16227766016838
'       4.47213595499958
'       5.47722557505166
'

```

For complete examples that uses [TransformBlock<TInput,TOutput>](#) in a network of dataflow blocks that performs image processing in a Windows Forms application, see [Walkthrough: Using Dataflow in a Windows Forms Application](#).

TransformManyBlock<TInput, TOutput>

The [TransformManyBlock<TInput,TOutput>](#) class resembles the [TransformBlock<TInput,TOutput>](#) class, except that [TransformManyBlock<TInput,TOutput>](#) produces zero or more output values for each input value, instead of only one output value for each input value. The delegate that you provide to a [TransformManyBlock<TInput,TOutput>](#) object can be of type [System.Func<TInput, IEnumerable<TOutput>>](#) or type [System.Func<TInput, Task<IEnumerable<TOutput>>>](#). When you use a [TransformManyBlock<TInput,TOutput>](#) object with [System.Func<TInput, IEnumerable<TOutput>>](#), processing of each input element is considered completed when the delegate returns. When you use a [TransformManyBlock<TInput,TOutput>](#) object with [System.Func<TInput, Task<IEnumerable<TOutput>>>](#), processing of each input element is considered complete only when the returned [System.Threading.Tasks.Task<IEnumerable<TOutput>>](#) object is completed.

The following basic example creates a [TransformManyBlock<TInput,TOutput>](#) object that splits strings into their individual character sequences. The [TransformManyBlock<TInput,TOutput>](#) object takes [String](#) values as input and produces [Char](#) values as output.

```

// Create a TransformManyBlock<string, char> object that splits
// a string into its individual characters.
var transformManyBlock = new TransformManyBlock<string, char>(
    s => s.ToCharArray());

// Post two messages to the first block.
transformManyBlock.Post("Hello");
transformManyBlock.Post("World");

// Receive all output values from the block.
for (int i = 0; i < ("Hello" + "World").Length; i++)
{
    Console.WriteLine(transformManyBlock.Receive());
}

/* Output:
H
e
l
l
o
W
o
r
l
d
*/

```

```

' Create a TransformManyBlock<string, Char> object that splits
' a string into its individual characters.
Dim transformManyBlock = New TransformManyBlock(Of String, Char)(Function(s) s.ToCharArray())

' Post two messages to the first block.
transformManyBlock.Post("Hello")
transformManyBlock.Post("World")

' Receive all output values from the block.
For i As Integer = 0 To ("Hello" & "World").Length - 1
    Console.WriteLine(transformManyBlock.Receive())
Next i

'       Output:
'       H
'       e
'       l
'       l
'       o
'       W
'       o
'       r
'       l
'       d
'

```

For complete examples that use [TransformManyBlock<TInput,TOutput>](#) to produce multiple independent outputs for each input in a dataflow pipeline, see [Walkthrough: Creating a Dataflow Pipeline](#).

Degree of Parallelism

Every [ActionBlock<TInput>](#), [TransformBlock<TInput,TOutput>](#), and [TransformManyBlock<TInput,TOutput>](#) object buffers input messages until the block is ready to process them. By default, these classes process messages in the order in which they are received, one message at a time. You can also specify the degree of parallelism to enable [ActionBlock<TInput>](#), [TransformBlock<TInput,TOutput>](#) and [TransformManyBlock<TInput,TOutput>](#) objects to process multiple messages concurrently. For more

information about concurrent execution, see the section Specifying the Degree of Parallelism later in this document. For an example that sets the degree of parallelism to enable an execution dataflow block to process more than one message at a time, see [How to: Specify the Degree of Parallelism in a Dataflow Block](#).

Summary of Delegate Types

The following table summarizes the delegate types that you can provide to `ActionBlock<TInput>`, `TransformBlock<TInput,TOutput>`, and `TransformManyBlock<TInput,TOutput>` objects. This table also specifies whether the delegate type operates synchronously or asynchronously.

TYPE	SYNCHRONOUS DELEGATE TYPE	ASYNCRONOUS DELEGATE TYPE
<code>ActionBlock<TInput></code>	<code>System.Action</code>	<code>System.Func<TInput, Task></code>
<code>TransformBlock<TInput,TOutput></code>	<code>System.Func<TInput, TOutput></code>	<code>System.Func<TInput, Task<TOutput>></code>
<code>TransformManyBlock<TInput,TOutput></code>	<code>System.Func<TInput, IEnumerable<TOutput>></code>	<code>System.Func<TInput, Task<IEnumerable<TOutput>>></code>

You can also use lambda expressions when you work with execution block types. For an example that shows how to use a lambda expression with an execution block, see [How to: Perform Action When a Dataflow Block Receives Data](#).

Grouping Blocks

Grouping blocks combine data from one or more sources and under various constraints. The TPL Dataflow Library provides three join block types: `BatchBlock<T>`, `JoinBlock<T1,T2>`, and `BatchedJoinBlock<T1,T2>`.

`BatchBlock<T>`

The `BatchBlock<T>` class combines sets of input data, which are known as batches, into arrays of output data. You specify the size of each batch when you create a `BatchBlock<T>` object. When the `BatchBlock<T>` object receives the specified count of input elements, it asynchronously propagates out an array that contains those elements. If a `BatchBlock<T>` object is set to the completed state but does not contain enough elements to form a batch, it propagates out a final array that contains the remaining input elements.

The `BatchBlock<T>` class operates in either *greedy* or *non-greedy* mode. In greedy mode, which is the default, a `BatchBlock<T>` object accepts every message that it is offered and propagates out an array after it receives the specified count of elements. In non-greedy mode, a `BatchBlock<T>` object postpones all incoming messages until enough sources have offered messages to the block to form a batch. Greedy mode typically performs better than non-greedy mode because it requires less processing overhead. However, you can use non-greedy mode when you must coordinate consumption from multiple sources in an atomic fashion. Specify non-greedy mode by setting `Greedy` to `False` in the `dataflowBlockOptions` parameter in the `BatchBlock<T>` constructor.

The following basic example posts several `Int32` values to a `BatchBlock<T>` object that holds ten elements in a batch. To guarantee that all values propagate out of the `BatchBlock<T>`, this example calls the `Complete` method. The `Complete` method sets the `BatchBlock<T>` object to the completed state, and therefore, the `BatchBlock<T>` object propagates out any remaining elements as a final batch.

```

// Create a BatchBlock<int> object that holds ten
// elements per batch.
var batchBlock = new BatchBlock<int>(10);

// Post several values to the block.
for (int i = 0; i < 13; i++)
{
    batchBlock.Post(i);
}
// Set the block to the completed state. This causes
// the block to propagate out any remaining
// values as a final batch.
batchBlock.Complete();

// Print the sum of both batches.

Console.WriteLine("The sum of the elements in batch 1 is {0}.",
    batchBlock.Receive().Sum());

Console.WriteLine("The sum of the elements in batch 2 is {0}.",
    batchBlock.Receive().Sum());

/* Output:
   The sum of the elements in batch 1 is 45.
   The sum of the elements in batch 2 is 33.
*/

```

```

' Create a BatchBlock<int> object that holds ten
' elements per batch.
Dim batchBlock = New BatchBlock(Of Integer)(10)

' Post several values to the block.
For i As Integer = 0 To 12
    batchBlock.Post(i)
Next i
' Set the block to the completed state. This causes
' the block to propagate out any remaining
' values as a final batch.
batchBlock.Complete()

' Print the sum of both batches.

Console.WriteLine("The sum of the elements in batch 1 is {0}.", batchBlock.Receive().Sum())

Console.WriteLine("The sum of the elements in batch 2 is {0}.", batchBlock.Receive().Sum())

'       Output:
'           The sum of the elements in batch 1 is 45.
'           The sum of the elements in batch 2 is 33.
'

```

For a complete example that uses [BatchBlock<T>](#) to improve the efficiency of database insert operations, see [Walkthrough: Using BatchBlock and BatchedJoinBlock to Improve Efficiency](#).

JoinBlock<T1, T2, ...>

The [JoinBlock<T1,T2>](#) and [JoinBlock<T1,T2,T3>](#) classes collect input elements and propagate out [System.Tuple<T1,T2>](#) or [System.Tuple<T1,T2,T3>](#) objects that contain those elements. The [JoinBlock<T1,T2>](#) and [JoinBlock<T1,T2,T3>](#) classes do not inherit from [ITargetBlock<TInput>](#). Instead, they provide properties, [Target1](#), [Target2](#), and [Target3](#), that implement [ITargetBlock<TInput>](#).

Like [BatchBlock<T>](#), [JoinBlock<T1,T2>](#) and [JoinBlock<T1,T2,T3>](#) operate in either greedy or non-greedy mode. In greedy mode, which is the default, a [JoinBlock<T1,T2>](#) or [JoinBlock<T1,T2,T3>](#) object accepts every message that it is offered and propagates out a tuple after each of its targets receives at least one message. In non-greedy

mode, a [JoinBlock<T1,T2>](#) or [JoinBlock<T1,T2,T3>](#) object postpones all incoming messages until all targets have been offered the data that is required to create a tuple. At this point, the block engages in a two-phase commit protocol to atomically retrieve all required items from the sources. This postponement makes it possible for another entity to consume the data in the meantime, to allow the overall system to make forward progress.

The following basic example demonstrates a case in which a [JoinBlock<T1,T2,T3>](#) object requires multiple data to compute a value. This example creates a [JoinBlock<T1,T2,T3>](#) object that requires two [Int32](#) values and a [Char](#) value to perform an arithmetic operation.

```
// Create a JoinBlock<int, int, char> object that requires
// two numbers and an operator.
var joinBlock = new JoinBlock<int, int, char>();

// Post two values to each target of the join.

joinBlock.Target1.Post(3);
joinBlock.Target1.Post(6);

joinBlock.Target2.Post(5);
joinBlock.Target2.Post(4);

joinBlock.Target3.Post('+');
joinBlock.Target3.Post('-');

// Receive each group of values and apply the operator part
// to the number parts.

for (int i = 0; i < 2; i++)
{
    var data = joinBlock.Receive();
    switch (data.Item3)
    {
        case '+':
            Console.WriteLine("{0} + {1} = {2}",
                data.Item1, data.Item2, data.Item1 + data.Item2);
            break;
        case '-':
            Console.WriteLine("{0} - {1} = {2}",
                data.Item1, data.Item2, data.Item1 - data.Item2);
            break;
        default:
            Console.WriteLine("Unknown operator '{0}'.", data.Item3);
            break;
    }
}

/* Output:
   3 + 5 = 8
   6 - 4 = 2
*/
```

```

' Create a JoinBlock<int, int, char> object that requires
' two numbers and an operator.
Dim joinBlock = New JoinBlock(Of Integer, Integer, Char)()

' Post two values to each target of the join.

joinBlock.Target1.Post(3)
joinBlock.Target1.Post(6)

joinBlock.Target2.Post(5)
joinBlock.Target2.Post(4)

joinBlock.Target3.Post("+"c)
joinBlock.Target3.Post("-"c)

' Receive each group of values and apply the operator part
' to the number parts.

For i As Integer = 0 To 1
    Dim data = joinBlock.Receive()
    Select Case data.Item3
        Case "+"c
            Console.WriteLine("{0} + {1} = {2}", data.Item1, data.Item2, data.Item1 + data.Item2)
        Case "-"c
            Console.WriteLine("{0} - {1} = {2}", data.Item1, data.Item2, data.Item1 - data.Item2)
        Case Else
            Console.WriteLine("Unknown operator '{0}'.", data.Item3)
    End Select
Next i

'       Output:
'       3 + 5 = 8
'       6 - 4 = 2
'

```

For a complete example that uses [JoinBlock<T1,T2>](#) objects in non-greedy mode to cooperatively share a resource, see [How to: Use JoinBlock to Read Data From Multiple Sources](#).

BatchedJoinBlock(T1, T2, ...)

The [BatchedJoinBlock<T1,T2>](#) and [BatchedJoinBlock<T1,T2,T3>](#) classes collect batches of input elements and propagate out `System.Tuple(IList(T1), IList(T2))` or `System.Tuple(IList(T1), IList(T2), IList(T3))` objects that contain those elements. Think of [BatchedJoinBlock<T1,T2>](#) as a combination of [BatchBlock<T>](#) and [JoinBlock<T1,T2>](#). Specify the size of each batch when you create a [BatchedJoinBlock<T1,T2>](#) object.

[BatchedJoinBlock<T1,T2>](#) also provides properties, [Target1](#) and [Target2](#), that implement [ITargetBlock<TInput>](#). When the specified count of input elements are received from across all targets, the [BatchedJoinBlock<T1,T2>](#) object asynchronously propagates out a `System.Tuple(IList(T1), IList(T2))` object that contains those elements.

The following basic example creates a [BatchedJoinBlock<T1,T2>](#) object that holds results, [Int32](#) values, and errors that are [Exception](#) objects. This example performs multiple operations and writes results to the [Target1](#) property, and errors to the [Target2](#) property, of the [BatchedJoinBlock<T1,T2>](#) object. Because the count of successful and failed operations is unknown in advance, the `IList<T>` objects enable each target to receive zero or more values.

```

// For demonstration, create a Func<int, int> that
// returns its argument, or throws ArgumentException
// if the argument is less than zero.
Func<int, int> DoWork = n =>
{
    if (n < 0)
        throw new ArgumentException();
    return n;
};

// Create a BatchedJoinBlock<int, Exception> object that holds
// seven elements per batch.
var batchedJoinBlock = new BatchedJoinBlock<int, Exception>(7);

// Post several items to the block.
foreach (int i in new int[] { 5, 6, -7, -22, 13, 55, 0 })
{
    try
    {
        // Post the result of the worker to the
        // first target of the block.
        batchedJoinBlock.Target1.Post(DoWork(i));
    }
    catch (ArgumentException e)
    {
        // If an error occurred, post the Exception to the
        // second target of the block.
        batchedJoinBlock.Target2.Post(e);
    }
}

// Read the results from the block.
var results = batchedJoinBlock.Receive();

// Print the results to the console.

// Print the results.
foreach (int n in results.Item1)
{
    Console.WriteLine(n);
}
// Print failures.
foreach (Exception e in results.Item2)
{
    Console.WriteLine(e.Message);
}

/* Output:
5
6
13
55
0
Specified argument was out of the range of valid values.
Specified argument was out of the range of valid values.
*/

```

```

' For demonstration, create a Func<int, int> that
' returns its argument, or throws ArgumentException
' if the argument is less than zero.
Dim DoWork As Func(Of Integer, Integer) = Function(n)
    If n < 0 Then
        Throw New ArgumentException()
    End If
    Return n
End Function

' Create a BatchedJoinBlock<int, Exception> object that holds
' seven elements per batch.
Dim batchedJoinBlock = New BatchedJoinBlock(Of Integer, Exception)(7)

' Post several items to the block.
For Each i As Integer In New Integer() {5, 6, -7, -22, 13, 55, 0}
    Try
        ' Post the result of the worker to the
        ' first target of the block.
        batchedJoinBlock.Target1.Post(DoWork(i))
    Catch e As ArgumentException
        ' If an error occurred, post the Exception to the
        ' second target of the block.
        batchedJoinBlock.Target2.Post(e)
    End Try
Next i

' Read the results from the block.
Dim results = batchedJoinBlock.Receive()

' Print the results to the console.

' Print the results.
For Each n As Integer In results.Item1
    Console.WriteLine(n)
Next n
' Print failures.
For Each e As Exception In results.Item2
    Console.WriteLine(e.Message)
Next e

' Output:
' 5
' 6
' 13
' 55
' 0
' Specified argument was out of the range of valid values.
' Specified argument was out of the range of valid values.
'

```

For a complete example that uses `BatchedJoinBlock<T1,T2>` to capture both the results and any exceptions that occur while the program reads from a database, see [Walkthrough: Using BatchBlock and BatchedJoinBlock to Improve Efficiency](#).

Configuring Dataflow Block Behavior

You can enable additional options by providing a `System.Threading.Tasks.Dataflow.DataflowBlockOptions` object to the constructor of dataflow block types. These options control behavior such the scheduler that manages the underlying task and the degree of parallelism. The `DataflowBlockOptions` also has derived types that specify behavior that is specific to certain dataflow block types. The following table summarizes which options type is associated with each dataflow block type.

DATAFLOW BLOCK TYPE	DATAFLOWBLOCKOPTIONS TYPE
BufferBlock<T>	DataflowBlockOptions
BroadcastBlock<T>	DataflowBlockOptions
WriteOnceBlock<T>	DataflowBlockOptions
ActionBlock<TInput>	ExecutionDataflowBlockOptions
TransformBlock<TInput,TOutput>	ExecutionDataflowBlockOptions
TransformManyBlock<TInput,TOutput>	ExecutionDataflowBlockOptions
BatchBlock<T>	GroupingDataflowBlockOptions
JoinBlock<T1,T2>	GroupingDataflowBlockOptions
BatchedJoinBlock<T1,T2>	GroupingDataflowBlockOptions

The following sections provide additional information about the important kinds of dataflow block options that are available through the [System.Threading.Tasks.Dataflow.DataflowBlockOptions](#), [System.Threading.Tasks.Dataflow.ExecutionDataflowBlockOptions](#), and [System.Threading.Tasks.Dataflow.GroupingDataflowBlockOptions](#) classes.

Specifying the Task Scheduler

Every predefined dataflow block uses the TPL task scheduling mechanism to perform activities such as propagating data to a target, receiving data from a source, and running user-defined delegates when data becomes available. [TaskScheduler](#) is an abstract class that represents a task scheduler that queues tasks onto threads. The default task scheduler, [Default](#), uses the [ThreadPool](#) class to queue and execute work. You can override the default task scheduler by setting the [TaskScheduler](#) property when you construct a dataflow block object.

When the same task scheduler manages multiple dataflow blocks, it can enforce policies across them. For example, if multiple dataflow blocks are each configured to target the exclusive scheduler of the same [ConcurrentExclusiveSchedulerPair](#) object, all work that runs across these blocks is serialized. Similarly, if these blocks are configured to target the concurrent scheduler of the same [ConcurrentExclusiveSchedulerPair](#) object, and that scheduler is configured to have a maximum concurrency level, all work from these blocks is limited to that number of concurrent operations. For an example that uses the [ConcurrentExclusiveSchedulerPair](#) class to enable read operations to occur in parallel, but write operations to occur exclusively of all other operations, see [How to: Specify a Task Scheduler in a Dataflow Block](#). For more information about task schedulers in the TPL, see the [TaskScheduler](#) class topic.

Specifying the Degree of Parallelism

By default, the three execution block types that the TPL Dataflow Library provides, [ActionBlock<TInput>](#), [TransformBlock<TInput,TOutput>](#), and [TransformManyBlock<TInput,TOutput>](#), process one message at a time. These dataflow block types also process messages in the order in which they are received. To enable these dataflow blocks to process messages concurrently, set the [ExecutionDataflowBlockOptions.MaxDegreeOfParallelism](#) property when you construct the dataflow block object.

The default value of [MaxDegreeOfParallelism](#) is 1, which guarantees that the dataflow block processes one message at a time. Setting this property to a value that is larger than 1 enables the dataflow block to process

multiple messages concurrently. Setting this property to [DataflowBlockOptions.Unbounded](#) enables the underlying task scheduler to manage the maximum degree of concurrency.

IMPORTANT

When you specify a maximum degree of parallelism that is larger than 1, multiple messages are processed simultaneously, and therefore messages might not be processed in the order in which they are received. The order in which the messages are output from the block is, however, the same one in which they are received.

Because the [MaxDegreeOfParallelism](#) property represents the maximum degree of parallelism, the dataflow block might execute with a lesser degree of parallelism than you specify. The dataflow block might use a lesser degree of parallelism to meet its functional requirements or because there is a lack of available system resources. A dataflow block never chooses more parallelism than you specify.

The value of the [MaxDegreeOfParallelism](#) property is exclusive to each dataflow block object. For example, if four dataflow block objects each specify 1 for the maximum degree of parallelism, all four dataflow block objects can potentially run in parallel.

For an example that sets the maximum degree of parallelism to enable lengthy operations to occur in parallel, see [How to: Specify the Degree of Parallelism in a Dataflow Block](#).

Specifying the Number of Messages per Task

The predefined dataflow block types use tasks to process multiple input elements. This helps minimize the number of task objects that are required to process data, which enables applications to run more efficiently. However, when the tasks from one set of dataflow blocks are processing data, the tasks from other dataflow blocks might need to wait for processing time by queuing messages. To enable better fairness among dataflow tasks, set the [MaxMessagesPerTask](#) property. When [MaxMessagesPerTask](#) is set to [DataflowBlockOptions.Unbounded](#), which is the default, the task used by a dataflow block processes as many messages as are available. When [MaxMessagesPerTask](#) is set to a value other than [Unbounded](#), the dataflow block processes at most this number of messages per [Task](#) object. Although setting the [MaxMessagesPerTask](#) property can increase fairness among tasks, it can cause the system to create more tasks than are necessary, which can decrease performance.

Enabling Cancellation

The TPL provides a mechanism that enables tasks to coordinate cancellation in a cooperative manner. To enable dataflow blocks to participate in this cancellation mechanism, set the [CancellationToken](#) property. When this [CancellationToken](#) object is set to the canceled state, all dataflow blocks that monitor this token finish execution of their current item but do not start processing subsequent items. These dataflow blocks also clear any buffered messages, release connections to any source and target blocks, and transition to the canceled state. By transitioning to the canceled state, the [Completion](#) property has the [Status](#) property set to [Canceled](#), unless an exception occurred during processing. In that case, [Status](#) is set to [Faulted](#).

For an example that demonstrates how to use cancellation in a Windows Forms application, see [How to: Cancel a Dataflow Block](#). For more information about cancellation in the TPL, see [Task Cancellation](#).

Specifying Greedy Versus Non-Greedy Behavior

Several grouping dataflow block types can operate in either *greedy* or *non-greedy* mode. By default, the predefined dataflow block types operate in greedy mode.

For join block types such as [JoinBlock<T1,T2>](#), greedy mode means that the block immediately accepts data even if the corresponding data with which to join is not yet available. Non-greedy mode means that the block postpones all incoming messages until one is available on each of its targets to complete the join. If any of the postponed messages are no longer available, the join block releases all postponed messages and restarts the process. For the [BatchBlock<T>](#) class, greedy and non-greedy behavior is similar, except that under non-greedy

mode, a `BatchBlock<T>` object postpones all incoming messages until enough are available from distinct sources to complete a batch.

To specify non-greedy mode for a dataflow block, set `Greedy` to `False`. For an example that demonstrates how to use non-greedy mode to enable multiple join blocks to share a data source more efficiently, see [How to: Use JoinBlock to Read Data From Multiple Sources](#).

Custom Dataflow Blocks

Although the TPL Dataflow Library provides many predefined block types, you can create additional block types that perform custom behavior. Implement the `ISourceBlock<TOutput>` or `ITargetBlock<TInput>` interfaces directly or use the `Encapsulate` method to build a complex block that encapsulates the behavior of existing block types. For examples that show how to implement custom dataflow block functionality, see [Walkthrough: Creating a Custom Dataflow Block Type](#).

Related Topics

TITLE	DESCRIPTION
How to: Write Messages to and Read Messages from a Dataflow Block	Demonstrates how to write messages to and read messages from a <code>BufferBlock<T></code> object.
How to: Implement a Producer-Consumer Dataflow Pattern	Describes how to use the dataflow model to implement a producer-consumer pattern, where the producer sends messages to a dataflow block, and the consumer reads messages from that block.
How to: Perform Action When a Dataflow Block Receives Data	Describes how to provide delegates to the execution dataflow block types, <code>ActionBlock<TInput></code> , <code>TransformBlock<TInput,TOutput></code> , and <code>TransformManyBlock<TInput,TOutput></code> .
Walkthrough: Creating a Dataflow Pipeline	Describes how to create a dataflow pipeline that downloads text from the web and performs operations on that text.
How to: Unlink Dataflow Blocks	Demonstrates how to use the <code>LinkTo</code> method to unlink a target block from its source after the source offers a message to the target.
Walkthrough: Using Dataflow in a Windows Forms Application	Demonstrates how to create a network of dataflow blocks that perform image processing in a Windows Forms application.
How to: Cancel a Dataflow Block	Demonstrates how to use cancellation in a Windows Forms application.
How to: Use JoinBlock to Read Data From Multiple Sources	Explains how to use the <code>JoinBlock<T1,T2></code> class to perform an operation when data is available from multiple sources, and how to use non-greedy mode to enable multiple join blocks to share a data source more efficiently.
How to: Specify the Degree of Parallelism in a Dataflow Block	Describes how to set the <code>MaxDegreeOfParallelism</code> property to enable an execution dataflow block to process more than one message at a time.

TITLE	DESCRIPTION
How to: Specify a Task Scheduler in a Dataflow Block	Demonstrates how to associate a specific task scheduler when you use dataflow in your application.
Walkthrough: Using BatchBlock and BatchedJoinBlock to Improve Efficiency	Describes how to use the <code>BatchBlock<T></code> class to improve the efficiency of database insert operations, and how to use the <code>BatchedJoinBlock<T1,T2></code> class to capture both the results and any exceptions that occur while the program reads from a database.
Walkthrough: Creating a Custom Dataflow Block Type	Demonstrates two ways to create a dataflow block type that implements custom behavior.
Task Parallel Library (TPL)	Introduces the TPL, a library that simplifies parallel and concurrent programming in .NET Framework applications.

How to: Write and read messages from a Dataflow block

9/20/2022 • 6 minutes to read • [Edit Online](#)

This article describes how to use the Task Parallel Library (TPL) Dataflow Library to write messages to and read messages from a dataflow block. The TPL Dataflow Library provides both synchronous and asynchronous methods for writing messages to and reading messages from a dataflow block. This article shows how to uses the `System.Threading.Tasks.Dataflow.BufferBlock<T>` class. The `BufferBlock<T>` class buffers messages and behaves as both a message source and a message target.

NOTE

The TPL Dataflow Library (the `System.Threading.Tasks.Dataflow` namespace) is not distributed with .NET. To install the `System.Threading.Tasks.Dataflow` namespace in Visual Studio, open your project, choose **Manage NuGet Packages** from the **Project** menu, and search online for the `System.Threading.Tasks.Dataflow` package. Alternatively, to install it using the [.NET Core CLI](#), run `dotnet add package System.Threading.Tasks.Dataflow`.

Writing and reading synchronously

The following example uses the `Post` method to write to a `BufferBlock<T>` dataflow block and the `Receive` method to read from the same object.

```
var bufferBlock = new BufferBlock<int>();

// Post several messages to the block.
for (int i = 0; i < 3; i++)
{
    bufferBlock.Post(i);
}

// Receive the messages back from the block.
for (int i = 0; i < 3; i++)
{
    Console.WriteLine(bufferBlock.Receive());
}

// Output:
// 0
// 1
// 2
```

```

Dim bufferBlock = New BufferBlock(Of Integer)()

' Post several messages to the block.
For i As Integer = 0 To 2
    bufferBlock.Post(i)
Next i

' Receive the messages back from the block.
For i As Integer = 0 To 2
    Console.WriteLine(bufferBlock.Receive())
Next i

' Output:
'   0
'   1
'   2

```

You can also use the [TryReceive](#) method to read from a dataflow block, as shown in the following example. The [TryReceive](#) method does not block the current thread and is useful when you occasionally poll for data.

```

// Post more messages to the block.
for (int i = 0; i < 3; i++)
{
    bufferBlock.Post(i);
}

// Receive the messages back from the block.
while (bufferBlock.TryReceive(out int value))
{
    Console.WriteLine(value);
}

// Output:
//   0
//   1
//   2

```

```

' Post more messages to the block.
For i As Integer = 0 To 2
    bufferBlock.Post(i)
Next i

' Receive the messages back from the block.
Dim value As Integer
Do While bufferBlock.TryReceive(value)
    Console.WriteLine(value)
Loop

' Output:
'   0
'   1
'   2

```

Because the [Post](#) method acts synchronously, the [BufferBlock<T>](#) object in the previous examples receives all data before the second loop reads data. The following example extends the first example by using [Task.WhenAll\(Task\[\]\)](#) to read from and write to the message block concurrently. Because [WhenAll](#) awaits all the asynchronous operations that are executing concurrently, the values are not written to the [BufferBlock<T>](#) object in any specific order.

```
// Write to and read from the message block concurrently.
var post01 = Task.Run(() =>
{
    bufferBlock.Post(0);
    bufferBlock.Post(1);
});
var receive = Task.Run(() =>
{
    for (int i = 0; i < 3; i++)
    {
        Console.WriteLine(bufferBlock.Receive());
    }
});
var post2 = Task.Run(() =>
{
    bufferBlock.Post(2);
});

await Task.WhenAll(post01, receive, post2);

// Output:
// 0
// 1
// 2
```

```
' Write to and read from the message block concurrently.
Dim post01 = Task.Run(Sub()
    bufferBlock.Post(0)
    bufferBlock.Post(1)
End Sub)
Dim receive = Task.Run(Sub()
    For i As Integer = 0 To 2
        Console.WriteLine(bufferBlock.Receive())
    Next i
End Sub)
Dim post2 = Task.Run(Sub() bufferBlock.Post(2))
Task.WaitAll(post01, receive, post2)

' Output:
' 0
' 1
' 2
```

Writing and reading asynchronously

The following example uses the [SendAsync](#) method to asynchronously write to a `BufferBlock<T>` object and the [ReceiveAsync](#) method to asynchronously read from the same object. This example uses the `async` and `await` operators ([Async](#) and [Await](#) in Visual Basic) to asynchronously send data to and read data from the target block. The [SendAsync](#) method is useful when you must enable a dataflow block to postpone messages. The [ReceiveAsync](#) method is useful when you want to act on data when that data becomes available. For more information about how messages propagate among message blocks, see the section [Message Passing in Dataflow](#).

```

// Post more messages to the block asynchronously.
for (int i = 0; i < 3; i++)
{
    await bufferBlock.SendAsync(i);
}

// Asynchronously receive the messages back from the block.
for (int i = 0; i < 3; i++)
{
    Console.WriteLine(await bufferBlock.ReceiveAsync());
}

// Output:
// 0
// 1
// 2

```

```

' Post more messages to the block asynchronously.
For i As Integer = 0 To 2
    await bufferBlock.SendAsync(i)
Next i

' Asynchronously receive the messages back from the block.
For i As Integer = 0 To 2
    Console.WriteLine(await bufferBlock.ReceiveAsync())
Next i

' Output:
' 0
' 1
' 2

```

A complete example

The following example shows all of the code for this article.

```

using System;
using System.Threading.Tasks;
using System.Threading.Tasks.Dataflow;

// Demonstrates how to write to and read from a dataflow block.
class DataflowReadWrite
{
    // Demonstrates asynchronous dataflow operations.
    static async Task AsyncSendReceive(BufferBlock<int> bufferBlock)
    {
        // Post more messages to the block asynchronously.
        for (int i = 0; i < 3; i++)
        {
            await bufferBlock.SendAsync(i);
        }

        // Asynchronously receive the messages back from the block.
        for (int i = 0; i < 3; i++)
        {
            Console.WriteLine(await bufferBlock.ReceiveAsync());
        }

        // Output:
        // 0
        // 1
        // 2
    }
}

```

```
static async Task Main()
{
    var bufferBlock = new BufferBlock<int>();

    // Post several messages to the block.
    for (int i = 0; i < 3; i++)
    {
        bufferBlock.Post(i);
    }

    // Receive the messages back from the block.
    for (int i = 0; i < 3; i++)
    {
        Console.WriteLine(bufferBlock.Receive());
    }

    // Output:
    // 0
    // 1
    // 2

    // Post more messages to the block.
    for (int i = 0; i < 3; i++)
    {
        bufferBlock.Post(i);
    }

    // Receive the messages back from the block.
    while (bufferBlock.TryReceive(out int value))
    {
        Console.WriteLine(value);
    }

    // Output:
    // 0
    // 1
    // 2

    // Write to and read from the message block concurrently.
    var post01 = Task.Run(() =>
    {
        bufferBlock.Post(0);
        bufferBlock.Post(1);
    });
    var receive = Task.Run(() =>
    {
        for (int i = 0; i < 3; i++)
        {
            Console.WriteLine(bufferBlock.Receive());
        }
    });
    var post2 = Task.Run(() =>
    {
        bufferBlock.Post(2);
    });

    await Task.WhenAll(post01, receive, post2);

    // Output:
    // 0
    // 1
    // 2

    // Demonstrate asynchronous dataflow operations.
    await AsyncSendReceive(bufferBlock);
}

}
```

```

Imports System.Threading.Tasks
Imports System.Threading.Tasks.Dataflow

' Demonstrates how to write to and read from a dataflow block.
Friend Class DataflowReadWrite
    ' Demonstrates asynchronous dataflow operations.
    Private Shared Async Function AsyncSendReceive(ByVal bufferBlock As BufferBlock(Of Integer)) As Task
        ' Post more messages to the block asynchronously.
        For i As Integer = 0 To 2
            await bufferBlock.SendAsync(i)
        Next i

        ' Asynchronously receive the messages back from the block.
        For i As Integer = 0 To 2
            Console.WriteLine(await bufferBlock.ReceiveAsync())
        Next i

        ' Output:
        ' 0
        ' 1
        ' 2
    End Function

    Shared Sub Main(ByVal args() As String)
        Dim bufferBlock = New BufferBlock(Of Integer)()

        ' Post several messages to the block.
        For i As Integer = 0 To 2
            bufferBlock.Post(i)
        Next i

        ' Receive the messages back from the block.
        For i As Integer = 0 To 2
            Console.WriteLine(bufferBlock.Receive())
        Next i

        ' Output:
        ' 0
        ' 1
        ' 2

        ' Post more messages to the block.
        For i As Integer = 0 To 2
            bufferBlock.Post(i)
        Next i

        ' Receive the messages back from the block.
        Dim value As Integer
        Do While bufferBlock.TryReceive(value)
            Console.WriteLine(value)
        Loop

        ' Output:
        ' 0
        ' 1
        ' 2

        ' Write to and read from the message block concurrently.
        Dim post01 = Task.Run(Sub()
                                bufferBlock.Post(0)
                                bufferBlock.Post(1)
                            End Sub)
        Dim receive = Task.Run(Sub()
                                For i As Integer = 0 To 2
                                    Console.WriteLine(bufferBlock.Receive())
                                Next i
                            End Sub)
        Dim post2 = Task.Run(Sub() bufferBlock.Post(2))
    End Sub
End Class

```

```
    Task.WaitAll(post01, receive, post2)

    ' Output:
    ' 0
    ' 1
    ' 2

    ' Demonstrate asynchronous dataflow operations.
    AsyncSendReceive(bufferBlock).Wait()
End Sub

End Class
```

Next steps

This example shows how to read from and write to a message block directly. You can also connect dataflow blocks to form *pipelines*, which are linear sequences of dataflow blocks, or *networks*, which are graphs of dataflow blocks. In a pipeline or network, sources asynchronously propagate data to targets as that data becomes available. For an example that creates a basic dataflow pipeline, see [Walkthrough: Creating a Dataflow Pipeline](#). For an example that creates a more complex dataflow network, see [Walkthrough: Using Dataflow in a Windows Forms Application](#).

See also

- [Dataflow \(Task Parallel Library\)](#)

How to: Implement a producer-consumer dataflow pattern

9/20/2022 • 3 minutes to read • [Edit Online](#)

In this article, you'll learn how to use the TPL dataflow library to implement a producer-consumer pattern. In this pattern, the *producer* sends messages to a message block, and the *consumer* reads messages from that block.

NOTE

The TPL Dataflow Library (the `System.Threading.Tasks.Dataflow` namespace) is not distributed with .NET. To install the `System.Threading.Tasks.Dataflow` namespace in Visual Studio, open your project, choose **Manage NuGet Packages** from the **Project** menu, and search online for the `System.Threading.Tasks.Dataflow` package. Alternatively, to install it using the [.NET Core CLI](#), run `dotnet add package System.Threading.Tasks.Dataflow`.

Example

The following example demonstrates a basic producer-consumer model that uses dataflow. The `Produce` method writes arrays that contain random bytes of data to a `System.Threading.Tasks.Dataflow.ITargetBlock<TInput>` object and the `Consume` method reads bytes from a `System.Threading.Tasks.Dataflow.ISourceBlock<TOutput>` object. By acting on the `ISourceBlock<TOutput>` and `ITargetBlock<TInput>` interfaces, instead of their derived types, you can write reusable code that can act on a variety of dataflow block types. This example uses the `BufferBlock<T>` class. Because the `BufferBlock<T>` class acts as both a source block and as a target block, the producer and the consumer can use a shared object to transfer data.

The `Produce` method calls the `Post` method in a loop to synchronously write data to the target block. After the `Produce` method writes all data to the target block, it calls the `Complete` method to indicate that the block will never have additional data available. The `Consume` method uses the `async` and `await` operators (`Async` and `Await` in Visual Basic) to asynchronously compute the total number of bytes that are received from the `ISourceBlock<TOutput>` object. To act asynchronously, the `Consume` method calls the `OutputAvailableAsync` method to receive a notification when the source block has data available and when the source block will never have additional data available.

```
using System;
using System.Threading.Tasks;
using System.Threading.Tasks.Dataflow;

class DataflowProducerConsumer
{
    static void Produce(ITargetBlock<byte[]> target)
    {
        var rand = new Random();

        for (int i = 0; i < 100; ++ i)
        {
            var buffer = new byte[1024];
            rand.NextBytes(buffer);
            target.Post(buffer);
        }

        target.Complete();
    }

    static async Task<int> ConsumeAsync(ISourceBlock<byte[]> source)
    {
        int bytesProcessed = 0;

        while (await source.OutputAvailableAsync())
        {
            byte[] data = await source.ReceiveAsync();
            bytesProcessed += data.Length;
        }

        return bytesProcessed;
    }

    static async Task Main()
    {
        var buffer = new BufferBlock<byte[]>();
        var consumerTask = ConsumeAsync(buffer);
        Produce(buffer);

        var bytesProcessed = await consumerTask;

        Console.WriteLine($"Processed {bytesProcessed:#,#} bytes.");
    }
}

// Sample output:
//      Processed 102,400 bytes.
```

```

Imports System.Threading.Tasks
Imports System.Threading.Tasks.Dataflow

Friend Class DataflowProducerConsumer
    Private Shared Sub Produce(ByVal target As ITargetBlock(Of Byte()))
        Dim rand As New Random()

        For i As Integer = 0 To 99
            Dim buffer(1023) As Byte
            rand.NextBytes(buffer)
            target.Post(buffer)
        Next i

        target.Complete()
    End Sub

    Private Shared Async Function ConsumeAsync(
        ByVal source As ISourceBlock(Of Byte()) As Task(Of Integer)
        Dim bytesProcessed As Integer = 0

        Do While Await source.OutputAvailableAsync()
            Dim data() As Byte = Await source.ReceiveAsync()
            bytesProcessed += data.Length
        Loop

        Return bytesProcessed
    End Function

    Shared Sub Main()
        Dim buffer = New BufferBlock(Of Byte)()
        Dim consumer = ConsumeAsync(buffer)
        Produce(buffer)

        Dim result = consumer.GetAwaiter().GetResult()

        Console.WriteLine($"Processed {result:#,##} bytes.")
    End Sub
End Class

' Sample output:
'     Processed 102,400 bytes.

```

Robust programming

The preceding example uses just one consumer to process the source data. If you have multiple consumers in your application, use the [TryReceive](#) method to read data from the source block, as shown in the following example.

```

static async Task<int> ConsumeAsync(IReceivableSourceBlock<byte[]> source)
{
    int bytesProcessed = 0;
    while (await source.OutputAvailableAsync())
    {
        while (source.TryReceive(out byte[] data))
        {
            bytesProcessed += data.Length;
        }
    }
    return bytesProcessed;
}

```

```
Private Shared Async Function ConsumeAsync(  
    ByVal source As IReceivableSourceBlock(Of Byte()) As Task(Of Integer)  
    Dim bytesProcessed As Integer = 0  
  
    Do While Await source.OutputAvailableAsync()  
        Dim data() As Byte  
        Do While source.TryReceive(data)  
            bytesProcessed += data.Length  
        Loop  
    Loop  
  
    Return bytesProcessed  
End Function
```

The [TryReceive](#) method returns `False` when no data is available. When multiple consumers must access the source block concurrently, this mechanism guarantees that data is still available after the call to [OutputAvailableAsync](#).

See also

- [Dataflow](#)

How to: Perform Action When a Dataflow Block Receives Data

9/20/2022 • 6 minutes to read • [Edit Online](#)

Execution dataflow block types call a user-provided delegate when they receive data. The `System.Threading.Tasks.Dataflow.ActionBlock<TInput>`, `System.Threading.Tasks.Dataflow.TransformBlock<TInput,TOutput>`, and `System.Threading.Tasks.Dataflow.TransformManyBlock<TInput,TOutput>` classes are execution dataflow block types. You can use the `delegate` keyword (`Sub` in Visual Basic), `Action<T>`, `Func<T,TResult>`, or a lambda expression when you provide a work function to an execution dataflow block. This document describes how to use `Func<T,TResult>` and lambda expressions to perform action in execution blocks.

NOTE

The TPL Dataflow Library (the `System.Threading.Tasks.Dataflow` namespace) is not distributed with .NET. To install the `System.Threading.Tasks.Dataflow` namespace in Visual Studio, open your project, choose **Manage NuGet Packages** from the **Project** menu, and search online for the `System.Threading.Tasks.Dataflow` package. Alternatively, to install it using the [.NET Core CLI](#), run `dotnet add package System.Threading.Tasks.Dataflow`.

Example

The following example uses dataflow to read a file from disk and computes the number of bytes in that file that are equal to zero. It uses `TransformBlock<TInput,TOutput>` to read the file and compute the number of zero bytes, and `ActionBlock<TInput>` to print the number of zero bytes to the console. The `TransformBlock<TInput,TOutput>` object specifies a `Func<T,TResult>` object to perform work when the blocks receive data. The `ActionBlock<TInput>` object uses a lambda expression to print to the console the number of zero bytes that are read.

```
using System;
using System.IO;
using System.Linq;
using System.Threading.Tasks;
using System.Threading.Tasks.Dataflow;

// Demonstrates how to provide delegates to execution dataflow blocks.
class DataflowExecutionBlocks
{
    // Computes the number of zero bytes that the provided file
    // contains.
    static int CountBytes(string path)
    {
        byte[] buffer = new byte[1024];
        int totalZeroBytesRead = 0;
        using (var fileStream = File.OpenRead(path))
        {
            int bytesRead = 0;
            do
            {
                bytesRead = fileStream.Read(buffer, 0, buffer.Length);
                totalZeroBytesRead += buffer.Count(b => b == 0);
            } while (bytesRead > 0);
        }
        return totalZeroBytesRead;
    }
}
```

```

    }

    static void Main(string[] args)
    {
        // Create a temporary file on disk.
        string tempFile = Path.GetTempFileName();

        // Write random data to the temporary file.
        using (var fileStream = File.OpenWrite(tempFile))
        {
            Random rand = new Random();
            byte[] buffer = new byte[1024];
            for (int i = 0; i < 512; i++)
            {
                rand.NextBytes(buffer);
                fileStream.Write(buffer, 0, buffer.Length);
            }
        }

        // Create an ActionBlock<int> object that prints to the console
        // the number of bytes read.
        var printResult = new ActionBlock<int>(zeroBytesRead =>
        {
            Console.WriteLine("{0} contains {1} zero bytes.",
                Path.GetFileName(tempFile), zeroBytesRead);
        });

        // Create a TransformBlock<string, int> object that calls the
        // CountBytes function and returns its result.
        var countBytes = new TransformBlock<string, int>(
            new Func<string, int>(CountBytes));

        // Link the TransformBlock<string, int> object to the
        // ActionBlock<int> object.
        countBytes.LinkTo(printResult);

        // Create a continuation task that completes the ActionBlock<int>
        // object when the TransformBlock<string, int> finishes.
        countBytes.Completion.ContinueWith(delegate { printResult.Complete(); });

        // Post the path to the temporary file to the
        // TransformBlock<string, int> object.
        countBytes.Post(tempFile);

        // Requests completion of the TransformBlock<string, int> object.
        countBytes.Complete();

        // Wait for the ActionBlock<int> object to print the message.
        printResult.Completion.Wait();

        // Delete the temporary file.
        File.Delete(tempFile);
    }
}

/* Sample output:
tmp4FBE.tmp contains 2081 zero bytes.
*/

```

```

Imports System.IO
Imports System.Linq
Imports System.Threading.Tasks
Imports System.Threading.Tasks.Dataflow

' Demonstrates how to provide delegates to execution dataflow blocks.
Friend Class DataflowExecutionBlocks
    ' Computes the number of zero bytes that the provided file

```

```

' contains.

Private Shared Function CountBytes(ByVal path As String) As Integer
    Dim buffer(1023) As Byte
    Dim totalZeroBytesRead As Integer = 0
    Using fileStream = File.OpenRead(path)
        Dim bytesRead As Integer = 0
        Do
            bytesRead = fileStream.Read(buffer, 0, buffer.Length)
            totalZeroBytesRead += buffer.Count(Function(b) b = 0)
        Loop While bytesRead > 0
    End Using

    Return totalZeroBytesRead
End Function

Shared Sub Main(ByVal args() As String)
    ' Create a temporary file on disk.
    Dim tempFile As String = Path.GetTempFileName()

    ' Write random data to the temporary file.
    Using fileStream = File.OpenWrite(tempFile)
        Dim rand As New Random()
        Dim buffer(1023) As Byte
        For i As Integer = 0 To 511
            rand.NextBytes(buffer)
            fileStream.Write(buffer, 0, buffer.Length)
        Next i
    End Using

    ' Create an ActionBlock<int> object that prints to the console
    ' the number of bytes read.
    Dim printResult = New ActionBlock(Of Integer)(Sub(zeroBytesRead) Console.WriteLine("{0} contains {1} zero bytes.", Path.GetFileName(tempFile), zeroBytesRead))

    ' Create a TransformBlock<string, int> object that calls the
    ' CountBytes function and returns its result.
    Dim countBytes = New TransformBlock(Of String, Integer)(New Func(Of String, Integer)(AddressOf
DataflowExecutionBlocks.CountBytes))

    ' Link the TransformBlock<string, int> object to the
    ' ActionBlock<int> object.
    countBytes.LinkTo(printResult)

    ' Create a continuation task that completes the ActionBlock<int>
    ' object when the TransformBlock<string, int> finishes.
    countBytes.Completion.ContinueWith(Sub() printResult.Complete())

    ' Post the path to the temporary file to the
    ' TransformBlock<string, int> object.
    countBytes.Post(tempFile)

    ' Requests completion of the TransformBlock<string, int> object.
    countBytes.Complete()

    ' Wait for the ActionBlock<int> object to print the message.
    printResult.Completion.Wait()

    ' Delete the temporary file.
    File.Delete(tempFile)
End Sub
End Class

' Sample output:
'tmp4FBE.tmp contains 2081 zero bytes.
'

```

Although you can provide a lambda expression to a [TransformBlock<TInput,TOutput>](#) object, this example uses

`Func<T,TResult>` to enable other code to use the `CountBytes` method. The `ActionBlock<TInput>` object uses a lambda expression because the work to be performed is specific to this task and is not likely to be useful from other code. For more information about how lambda expressions work in the Task Parallel Library, see [Lambda Expressions in PLINQ and TPL](#).

The section Summary of Delegate Types in the [Dataflow](#) document summarizes the delegate types that you can provide to `ActionBlock<TInput>`, `TransformBlock<TInput,TOutput>`, and `TransformManyBlock<TInput,TOutput>` objects. The table also specifies whether the delegate type operates synchronously or asynchronously.

Robust Programming

This example provides a delegate of type `Func<T,TResult>` to the `TransformBlock<TInput,TOutput>` object to perform the task of the dataflow block synchronously. To enable the dataflow block to behave asynchronously, provide a delegate of type `Func<T, Task<TResult>>` to the dataflow block. When a dataflow block behaves asynchronously, the task of the dataflow block is complete only when the returned `Task<TResult>` object finishes. The following example modifies the `CountBytes` method and uses the `async` and `await` operators ([Async](#) and [Await](#) in Visual Basic) to asynchronously compute the total number of bytes that are zero in the provided file. The `ReadAsync` method performs file read operations asynchronously.

```
// Asynchronously computes the number of zero bytes that the provided file
// contains.
static async Task<int> CountBytesAsync(string path)
{
    byte[] buffer = new byte[1024];
    int totalZeroBytesRead = 0;
    using (var fileStream = new FileStream(
        path, FileMode.Open, FileAccess.Read, FileShare.Read, 0x1000, true))
    {
        int bytesRead = 0;
        do
        {
            // Asynchronously read from the file stream.
            bytesRead = await fileStream.ReadAsync(buffer, 0, buffer.Length);
            totalZeroBytesRead += buffer.Count(b => b == 0);
        } while (bytesRead > 0);
    }

    return totalZeroBytesRead;
}
```

```
' Asynchronously computes the number of zero bytes that the provided file
' contains.
Private Shared async Function CountBytesAsync(ByVal path As String) As Task(Of Integer)
    Dim buffer(1023) As Byte
    Dim totalZeroBytesRead As Integer = 0
    Using fileStream = New FileStream(path, FileMode.Open, FileAccess.Read, FileShare.Read, &H1000, True)
        Dim bytesRead As Integer = 0
        Do
            ' Asynchronously read from the file stream.
            bytesRead = await fileStream.ReadAsync(buffer, 0, buffer.Length)
            totalZeroBytesRead += buffer.Count(Function(b) b = 0)
        Loop While bytesRead > 0
    End Using

    Return totalZeroBytesRead
End Function
```

You can also use asynchronous lambda expressions to perform action in an execution dataflow block. The following example modifies the `TransformBlock<TInput,TOutput>` object that is used in the previous example so

that it uses a lambda expression to perform the work asynchronously.

```
// Create a TransformBlock<string, int> object that calls the
// CountBytes function and returns its result.
var countBytesAsync = new TransformBlock<string, int>(async path =>
{
    byte[] buffer = new byte[1024];
    int totalZeroBytesRead = 0;
    using (var fileStream = new FileStream(
        path, FileMode.Open, FileAccess.Read, FileShare.Read, 0x1000, true))
    {
        int bytesRead = 0;
        do
        {
            // Asynchronously read from the file stream.
            bytesRead = await fileStream.ReadAsync(buffer, 0, buffer.Length);
            totalZeroBytesRead += buffer.Count(b => b == 0);
        } while (bytesRead > 0);
    }

    return totalZeroBytesRead;
});
```

```
' Create a TransformBlock<string, int> object that calls the
' CountBytes function and returns its result.
Dim countBytesAsync = New TransformBlock(Of String, Integer)(async Function(path)
    ' Asynchronously read from the file stream.
    Dim buffer(1023) As Byte
    Dim totalZeroBytesRead As Integer = 0
    Using fileStream = New FileStream(path,
        FileMode.Open, FileAccess.Read, FileShare.Read, &H1000, True)
        Dim bytesRead As Integer = 0
        Do
            bytesRead = Await
                fileStream.ReadAsync(buffer, 0, buffer.Length)
            totalZeroBytesRead +=
                buffer.Count(Function(b) b = 0)
        Loop While bytesRead > 0
    End Using
    Return totalZeroBytesRead
End Function)
```

See also

- [Dataflow](#)

Walkthrough: Creating a Dataflow Pipeline

9/20/2022 • 12 minutes to read • [Edit Online](#)

Although you can use the [DataflowBlock.Receive](#), [DataflowBlock.ReceiveAsync](#), and [DataflowBlock.TryReceive](#) methods to receive messages from source blocks, you can also connect message blocks to form a *dataflow pipeline*. A dataflow pipeline is a series of components, or *dataflow blocks*, each of which performs a specific task that contributes to a larger goal. Every dataflow block in a dataflow pipeline performs work when it receives a message from another dataflow block. An analogy to this is an assembly line for automobile manufacturing. As each vehicle passes through the assembly line, one station assembles the frame, the next one installs the engine, and so on. Because an assembly line enables multiple vehicles to be assembled at the same time, it provides better throughput than assembling complete vehicles one at a time.

This document demonstrates a dataflow pipeline that downloads the book *The Iliad of Homer* from a website and searches the text to match individual words with words that reverse the first word's characters. The formation of the dataflow pipeline in this document consists of the following steps:

1. Create the dataflow blocks that participate in the pipeline.
2. Connect each dataflow block to the next block in the pipeline. Each block receives as input the output of the previous block in the pipeline.
3. For each dataflow block, create a continuation task that sets the next block to the completed state after the previous block finishes.
4. Post data to the head of the pipeline.
5. Mark the head of the pipeline as completed.
6. Wait for the pipeline to complete all work.

Prerequisites

Read [Dataflow](#) before you start this walkthrough.

Creating a Console Application

In Visual Studio, create a Visual C# or Visual Basic Console Application project. Install the `System.Threading.Tasks.Dataflow` NuGet package.

NOTE

The TPL Dataflow Library (the `System.Threading.Tasks.Dataflow` namespace) is not distributed with .NET. To install the `System.Threading.Tasks.Dataflow` namespace in Visual Studio, open your project, choose **Manage NuGet Packages** from the **Project** menu, and search online for the `System.Threading.Tasks.Dataflow` package. Alternatively, to install it using the [.NET Core CLI](#), run `dotnet add package System.Threading.Tasks.Dataflow`.

Add the following code to your project to create the basic application.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net.Http;
using System.Threading.Tasks.Dataflow;

// Demonstrates how to create a basic dataflow pipeline.
// This program downloads the book "The Iliad of Homer" by Homer from the Web
// and finds all reversed words that appear in that book.
static class Program
{
    static void Main()
    {
    }
}
```

```
Imports System.Net.Http
Imports System.Threading.Tasks.Dataflow

' Demonstrates how to create a basic dataflow pipeline.
' This program downloads the book "The Iliad of Homer" by Homer from the Web
' and finds all reversed words that appear in that book.
Module DataflowReversedWords

    Sub Main()
    End Sub

End Module
```

Creating the Dataflow Blocks

Add the following code to the `Main` method to create the dataflow blocks that participate in the pipeline. The table that follows summarizes the role of each member of the pipeline.

```

// Create the members of the pipeline.
//

// Downloads the requested resource as a string.
var downloadString = new TransformBlock<string, string>(async uri =>
{
    Console.WriteLine("Downloading '{0}'...", uri);

    return await new HttpClient(new HttpClientHandler{ AutomaticDecompression =
System.Net.DecompressionMethods.GZip }).GetStringAsync(uri);
});

// Separates the specified text into an array of words.
var createWordList = new TransformBlock<string, string[]>(text =>
{
    Console.WriteLine("Creating word list...");

    // Remove common punctuation by replacing all non-letter characters
    // with a space character.
    char[] tokens = text.Select(c => char.IsLetter(c) ? c : ' ').ToArray();
    text = new string(tokens);

    // Separate the text into an array of words.
    return text.Split(new[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);
});

// Removes short words and duplicates.
var filterWordList = new TransformBlock<string[], string[]>(words =>
{
    Console.WriteLine("Filtering word list...");

    return words
        .Where(word => word.Length > 3)
        .Distinct()
        .ToArray();
});

// Finds all words in the specified collection whose reverse also
// exists in the collection.
var findReversedWords = new TransformManyBlock<string[], string>(words =>
{
    Console.WriteLine("Finding reversed words...");

    var wordsSet = new HashSet<string>(words);

    return from word in words.AsParallel()
           let reverse = new string(word.Reverse().ToArray())
           where word != reverse && wordsSet.Contains(reverse)
           select word;
});

// Prints the provided reversed words to the console.
var printReversedWords = new ActionBlock<string>(reversedWord =>
{
    Console.WriteLine("Found reversed words {0}/{1}",
        reversedWord, new string(reversedWord.Reverse().ToArray()));
});

```

```

' Create the members of the pipeline.
'

' Downloads the requested resource as a string.
Dim downloadString = New TransformBlock(Of String, String)(
    Async Function(uri)
        Console.WriteLine("Downloading '{0}'...", uri)

        Return Await New HttpClient().GetStringAsync(uri)
    End Function)

' Separates the specified text into an array of words.
Dim createWordList = New TransformBlock(Of String, String())(
    Function(text)
        Console.WriteLine("Creating word list...")

        ' Remove common punctuation by replacing all non-letter characters
        ' with a space character.
        Dim tokens() As Char = text.Select(Function(c) If(Char.IsLetter(c), c, " "c)).ToArray()
        text = New String(tokens)

        ' Separate the text into an array of words.
        Return text.Split(New Char() {" "c}, StringSplitOptions.RemoveEmptyEntries)
    End Function)

' Removes short words and duplicates.
Dim filterWordList = New TransformBlock(Of String(), String())(
    Function(words)
        Console.WriteLine("Filtering word list...")

        Return words.Where(Function(word) word.Length > 3).Distinct().ToArray()
    End Function)

' Finds all words in the specified collection whose reverse also
' exists in the collection.
Dim findReversedWords = New TransformManyBlock(Of String(), String)(
    Function(words)

        Dim wordsSet = New HashSet(Of String)(words)

        Return From word In words.AsParallel()
            Let reverse = New String(word.Reverse().ToArray())
            Where word <> reverse AndAlso wordsSet.Contains(reverse)
            Select word
    End Function)

' Prints the provided reversed words to the console.
Dim printReversedWords = New ActionBlock(Of String)(
    Sub(reversedWord)
        Console.WriteLine("Found reversed words {0}/{1}", reversedWord, New
String(reversedWord.Reverse().ToArray()))
    End Sub)

```

MEMBER	TYPE	DESCRIPTION
downloadString	TransformBlock<TInput,TOutput>	Downloads the book text from the Web.
createWordList	TransformBlock<TInput,TOutput>	Separates the book text into an array of words.

MEMBER	TYPE	DESCRIPTION
<code>filterWordList</code>	<code>TransformBlock<TInput,TOutput></code>	Removes short words and duplicates from the word array.
<code>findReversedWords</code>	<code>TransformManyBlock<TInput,TOutput></code>	Finds all words in the filtered word array collection whose reverse also occurs in the word array.
<code>printReversedWords</code>	<code>ActionBlock<TInput></code>	Displays words and the corresponding reverse words to the console.

Although you could combine multiple steps in the dataflow pipeline in this example into one step, the example illustrates the concept of composing multiple independent dataflow tasks to perform a larger task. The example uses `TransformBlock<TInput,TOutput>` to enable each member of the pipeline to perform an operation on its input data and send the results to the next step in the pipeline. The `findReversedWords` member of the pipeline is a `TransformManyBlock<TInput,TOutput>` object because it produces multiple independent outputs for each input. The tail of the pipeline, `printReversedWords`, is an `ActionBlock<TInput>` object because it performs an action on its input, and does not produce a result.

Forming the Pipeline

Add the following code to connect each block to the next block in the pipeline.

When you call the `LinkTo` method to connect a source dataflow block to a target dataflow block, the source dataflow block propagates data to the target block as data becomes available. If you also provide `DataflowLinkOptions` with `PropagateCompletion` set to true, successful or unsuccessful completion of one block in the pipeline will cause completion of the next block in the pipeline.

```
//  
// Connect the dataflow blocks to form a pipeline.  
  
var linkOptions = new DataflowLinkOptions { PropagateCompletion = true };  
  
downloadString.LinkTo(createWordList, linkOptions);  
createWordList.LinkTo(filterWordList, linkOptions);  
filterWordList.LinkTo(findReversedWords, linkOptions);  
findReversedWords.LinkTo(printReversedWords, linkOptions);
```

```
'  
' Connect the dataflow blocks to form a pipeline.  
'  
  
Dim linkOptions = New DataflowLinkOptions With {.PropagateCompletion = True}  
  
downloadString.LinkTo(createWordList, linkOptions)  
createWordList.LinkTo(filterWordList, linkOptions)  
filterWordList.LinkTo(findReversedWords, linkOptions)  
findReversedWords.LinkTo(printReversedWords, linkOptions)
```

Posting Data to the Pipeline

Add the following code to post the URL of the book *The Iliad of Homer* to the head of the dataflow pipeline.

```
// Process "The Iliad of Homer" by Homer.  
downloadString.Post("http://www.gutenberg.org/cache/epub/16452/pg16452.txt");
```

```
' Process "The Iliad of Homer" by Homer.  
downloadString.Post("http://www.gutenberg.org/cache/epub/16452/pg16452.txt")
```

This example uses [DataflowBlock.Post](#) to synchronously send data to the head of the pipeline. Use the [DataflowBlock.SendAsync](#) method when you must asynchronously send data to a dataflow node.

Completing Pipeline Activity

Add the following code to mark the head of the pipeline as completed. The head of the pipeline propagates its completion after it processes all buffered messages.

```
// Mark the head of the pipeline as complete.  
downloadString.Complete();
```

```
' Mark the head of the pipeline as complete.  
downloadString.Complete()
```

This example sends one URL through the dataflow pipeline to be processed. If you send more than one input through a pipeline, call the [IDataflowBlock.Complete](#) method after you submit all the input. You can omit this step if your application has no well-defined point at which data is no longer available or the application does not have to wait for the pipeline to finish.

Waiting for the Pipeline to Finish

Add the following code to wait for the pipeline to finish. The overall operation is finished when the tail of the pipeline finishes.

```
// Wait for the last block in the pipeline to process all messages.  
printReversedWords.Completion.Wait();
```

```
' Wait for the last block in the pipeline to process all messages.  
printReversedWords.Completion.Wait()
```

You can wait for dataflow completion from any thread or from multiple threads at the same time.

The Complete Example

The following example shows the complete code for this walkthrough.

```
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Net.Http;  
using System.Threading.Tasks.Dataflow;  
  
// Demonstrates how to create a basic dataflow pipeline.  
// This program downloads the book "The Iliad of Homer" by Homer from the Web  
// and finds all reversed words that appear in that book.  
static class DataflowReversedWords
```

```

{
    static void Main()
    {
        //
        // Create the members of the pipeline.
        //

        // Downloads the requested resource as a string.
        var downloadString = new TransformBlock<string, string>(async uri =>
        {
            Console.WriteLine("Downloading '{0}'...", uri);

            return await new HttpClient(new HttpClientHandler{ AutomaticDecompression =
System.Net.DecompressionMethods.GZip }).GetStringAsync(uri);
        });

        // Separates the specified text into an array of words.
        var createWordList = new TransformBlock<string, string[]>(text =>
        {
            Console.WriteLine("Creating word list...");

            // Remove common punctuation by replacing all non-letter characters
            // with a space character.
            char[] tokens = text.Select(c => char.IsLetter(c) ? c : ' ').ToArray();
            text = new string(tokens);

            // Separate the text into an array of words.
            return text.Split(new[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);
        });

        // Removes short words and duplicates.
        var filterWordList = new TransformBlock<string[], string[]>(words =>
        {
            Console.WriteLine("Filtering word list...");

            return words
                .Where(word => word.Length > 3)
                .Distinct()
                .ToArray();
        });

        // Finds all words in the specified collection whose reverse also
        // exists in the collection.
        var findReversedWords = new TransformManyBlock<string[], string>(words =>
        {
            Console.WriteLine("Finding reversed words...");

            var wordsSet = new HashSet<string>(words);

            return from word in words.AsParallel()
                   let reverse = new string(word.Reverse().ToArray())
                   where word != reverse && wordsSet.Contains(reverse)
                   select word;
        });

        // Prints the provided reversed words to the console.
        var printReversedWords = new ActionBlock<string>(reversedWord =>
        {
            Console.WriteLine("Found reversed words {0}/{1}",
                reversedWord, new string(reversedWord.Reverse().ToArray()));
        });

        //
        // Connect the dataflow blocks to form a pipeline.
        //

        var linkOptions = new DataflowLinkOptions { PropagateCompletion = true };

        downloadString.LinkTo(createWordList, linkOptions);
    }
}

```

```

        createWordList.LinkTo(filterWordList, linkOptions);
        filterWordList.LinkTo(findReversedWords, linkOptions);
        findReversedWords.LinkTo(printReversedWords, linkOptions);

        // Process "The Iliad of Homer" by Homer.
        downloadString.Post("http://www.gutenberg.org/cache/epub/16452/pg16452.txt");

        // Mark the head of the pipeline as complete.
        downloadString.Complete();

        // Wait for the last block in the pipeline to process all messages.
        printReversedWords.Completion.Wait();
    }
}

/* Sample output:
   Downloading 'http://www.gutenberg.org/cache/epub/16452/pg16452.txt'...
   Creating word list...
   Filtering word list...
   Finding reversed words...
   Found reversed words doom/mood
   Found reversed words draw/ward
   Found reversed words aera/area
   Found reversed words seat/taes
   Found reversed words live/evil
   Found reversed words port/trop
   Found reversed words sleek/keels
   Found reversed words area/aera
   Found reversed words tops/spot
   Found reversed words evil/live
   Found reversed words mood/doom
   Found reversed words speed/deeps
   Found reversed words moor/room
   Found reversed words trop/port
   Found reversed words spot/tops
   Found reversed words spots/stops
   Found reversed words stops/spots
   Found reversed words reed/deer
   Found reversed words keels/sleek
   Found reversed words deeps/speed
   Found reversed words deer/reed
   Found reversed words taes/seat
   Found reversed words room/moor
   Found reversed words ward/draw
*/

```

```

Imports System.Net.Http
Imports System.Threading.Tasks.Dataflow

' Demonstrates how to create a basic dataflow pipeline.
' This program downloads the book "The Iliad of Homer" by Homer from the Web
' and finds all reversed words that appear in that book.
Module DataflowReversedWords

    Sub Main()
        '
        ' Create the members of the pipeline.
        '

        ' Downloads the requested resource as a string.
        Dim downloadString = New TransformBlock(Of String, String)(
            Async Function(uri)
                Console.WriteLine("Downloading '{0}'...", uri)

                Return Await New HttpClient().GetStringAsync(uri)
            End Function)

        '
        ' Separates the specified text into an array of words.
        '

```

```

Dim createWordList = New TransformBlock(Of String, String())(
    Function(text)
        Console.WriteLine("Creating word list...")

        ' Remove common punctuation by replacing all non-letter characters
        ' with a space character.
        Dim tokens() As Char = text.Select(Function(c) If(Char.IsLetter(c), c, " "c)).ToArray()
        text = New String(tokens)

        ' Separate the text into an array of words.
        Return text.Split(New Char() {" "c}, StringSplitOptions.RemoveEmptyEntries)
    End Function)

    ' Removes short words and duplicates.
Dim filterWordList = New TransformBlock(Of String(), String())(
    Function(words)
        Console.WriteLine("Filtering word list...")

        Return words.Where(Function(word) word.Length > 3).Distinct().ToArray()
    End Function)

    ' Finds all words in the specified collection whose reverse also
    ' exists in the collection.
Dim findReversedWords = New TransformManyBlock(Of String(), String)(
    Function(words)

        Dim wordsSet = New HashSet(Of String)(words)

        Return From word In words.AsParallel()
            Let reverse = New String(word.Reverse().ToArray())
            Where word <> reverse AndAlso wordsSet.Contains(reverse)
            Select word
    End Function)

    ' Prints the provided reversed words to the console.
Dim printReversedWords = New ActionBlock(Of String)(
    Sub(reversedWord)
        Console.WriteLine("Found reversed words {0}/{1}", reversedWord, New
String(reversedWord.Reverse().ToArray()))
    End Sub)

    ' Connect the dataflow blocks to form a pipeline.
    ' 

Dim linkOptions = New DataflowLinkOptions With {.PropagateCompletion = True}

downloadString.LinkTo(createWordList, linkOptions)
createWordList.LinkTo(filterWordList, linkOptions)
filterWordList.LinkTo(findReversedWords, linkOptions)
findReversedWords.LinkTo(printReversedWords, linkOptions)

    ' Process "The Iliad of Homer" by Homer.
    downloadString.Post("http://www.gutenberg.org/cache/epub/16452/pg16452.txt")

    ' Mark the head of the pipeline as complete.
    downloadString.Complete()

    ' Wait for the last block in the pipeline to process all messages.
    printReversedWords.Completion.Wait()
End Sub

End Module

    ' Sample output:
'Downloading 'http://www.gutenberg.org/cache/epub/16452/pg16452.txt'...
'Creating word list...
'Filtering word list...
'Finding reversed words...

```

```
'Found reversed words aera/area
'Found reversed words doom/mood
'Found reversed words draw/ward
'Found reversed words live/evil
'Found reversed words seat/taes
'Found reversed words area/aera
'Found reversed words port/trop
'Found reversed words sleek/keels
'Found reversed words tops/spot
'Found reversed words evil/live
'Found reversed words speed/deeps
'Found reversed words mood/doom
'Found reversed words moor/room
'Found reversed words spot/tops
'Found reversed words spots/stops
'Found reversed words trop/port
'Found reversed words stops/spots
'Found reversed words reed/deer
'Found reversed words deeps/speed
'Found reversed words deer/reed
'Found reversed words taes/seat
'Found reversed words keels/sleek
'Found reversed words room/moor
'Found reversed words ward/draw
```

Next Steps

This example sends one URL to process through the dataflow pipeline. If you send more than one input value through a pipeline, you can introduce a form of parallelism into your application that resembles how parts might move through an automobile factory. When the first member of the pipeline sends its result to the second member, it can process another item in parallel as the second member processes the first result.

The parallelism that is achieved by using dataflow pipelines is known as *coarse-grained parallelism* because it typically consists of fewer, larger tasks. You can also use a more *fine-grained parallelism* of smaller, short-running tasks in a dataflow pipeline. In this example, the `findReversedWords` member of the pipeline uses [PLINQ](#) to process multiple items in the work list in parallel. The use of fine-grained parallelism in a coarse-grained pipeline can improve overall throughput.

You can also connect a source dataflow block to multiple target blocks to create a *dataflow network*. The overloaded version of the [LinkTo](#) method takes a `Predicate<T>` object that defines whether the target block accepts each message based on its value. Most dataflow block types that act as sources offer messages to all connected target blocks, in the order in which they were connected, until one of the blocks accepts that message. By using this filtering mechanism, you can create systems of connected dataflow blocks that direct certain data through one path and other data through another path. For an example that uses filtering to create a dataflow network, see [Walkthrough: Using Dataflow in a Windows Forms Application](#).

See also

- [Dataflow](#)

How to: Unlink Dataflow Blocks

9/20/2022 • 4 minutes to read • [Edit Online](#)

This document describes how to unlink a target dataflow block from its source.

NOTE

The TPL Dataflow Library (the `System.Threading.Tasks.Dataflow` namespace) is not distributed with .NET. To install the `System.Threading.Tasks.Dataflow` namespace in Visual Studio, open your project, choose **Manage NuGet Packages** from the **Project** menu, and search online for the `System.Threading.Tasks.Dataflow` package. Alternatively, to install it using the [.NET Core CLI](#), run `dotnet add package System.Threading.Tasks.Dataflow`.

Example

The following example creates three `TransformBlock<TInput,TOutput>` objects, each of which calls the

`TrySolution` method to compute a value. This example requires only the result from the first call to

`TrySolution` to finish.

```
using System;
using System.Threading;
using System.Threading.Tasks.Dataflow;

// Demonstrates how to unlink dataflow blocks.
class DataflowReceiveAny
{
    // Receives the value from the first provided source that has
    // a message.
    public static T ReceiveFromAny<T>(params ISourceBlock<T>[] sources)
    {
        // Create a WriteOnceBlock<T> object and link it to each source block.
        var writeOnceBlock = new WriteOnceBlock<T>(e => e);
        foreach (var source in sources)
        {
            // Setting MaxMessages to one instructs
            // the source block to unlink from the WriteOnceBlock<T> object
            // after offering the WriteOnceBlock<T> object one message.
            source.LinkTo(writeOnceBlock, new DataflowLinkOptions { MaxMessages = 1 });
        }
        // Return the first value that is offered to the WriteOnceBlock object.
        return writeOnceBlock.Receive();
    }

    // Demonstrates a function that takes several seconds to produce a result.
    static int TrySolution(int n, CancellationToken ct)
    {
        // Simulate a lengthy operation that completes within three seconds
        // or when the provided CancellationToken object is cancelled.
        SpinWait.SpinUntil(() => ct.IsCancellationRequested,
            new Random().Next(3000));

        // Return a value.
        return n + 42;
    }

    static void Main(string[] args)
    {
        // Create a shared CancellationTokenSource object to enable the
        // TrySolution method to be cancelled.
    }
}
```

```
var cts = new CancellationTokenSource();

// Create three TransformBlock<int, int> objects.
// Each TransformBlock<int, int> object calls the TrySolution method.
Func<int, int> action = n => TrySolution(n, cts.Token);
var trySolution1 = new TransformBlock<int, int>(action);
var trySolution2 = new TransformBlock<int, int>(action);
var trySolution3 = new TransformBlock<int, int>(action);

// Post data to each TransformBlock<int, int> object.
trySolution1.Post(11);
trySolution2.Post(21);
trySolution3.Post(31);

// Call the ReceiveFromAny<T> method to receive the result from the
// first TransformBlock<int, int> object to finish.
int result = ReceiveFromAny(trySolution1, trySolution2, trySolution3);

// Cancel all calls to TrySolution that are still active.
cts.Cancel();

// Print the result to the console.
Console.WriteLine("The solution is {0}.", result);

cts.Dispose();
}

}

/* Sample output:
The solution is 53.
*/
```

```

Imports System.Threading
Imports System.Threading.Tasks.Dataflow

' Demonstrates how to unlink dataflow blocks.
Friend Class DataflowReceiveAny
    ' Receives the value from the first provided source that has
    ' a message.
    Public Shared Function ReceiveFromAny(Of T)(ParamArray ByVal sources() As ISourceBlock(Of T)) As T
        ' Create a WriteOnceBlock<T> object and link it to each source block.
        Dim writeOnceBlock = New WriteOnceBlock(Of T)(Function(e) e)
        For Each source In sources
            ' Setting MaxMessages to one instructs
            ' the source block to unlink from the WriteOnceBlock<T> object
            ' after offering the WriteOnceBlock<T> object one message.
            source.LinkTo(writeOnceBlock, New DataflowLinkOptions With {.MaxMessages = 1})
        Next source
        ' Return the first value that is offered to the WriteOnceBlock object.
        Return writeOnceBlock.Receive()
    End Function

    ' Demonstrates a function that takes several seconds to produce a result.
    Private Shared Function TrySolution(ByVal n As Integer, ByVal ct As CancellationToken) As Integer
        ' Simulate a lengthy operation that completes within three seconds
        ' or when the provided CancellationToken object is cancelled.
        SpinWait.SpinUntil(Function() ct.IsCancellationRequested, New Random().Next(3000))

        ' Return a value.
        Return n + 42
    End Function

    Shared Sub Main(ByVal args() As String)
        ' Create a shared CancellationTokenSource object to enable the
        ' TrySolution method to be cancelled.
        Dim cts = New CancellationTokenSource()

        ' Create three TransformBlock<int, int> objects.
        ' Each TransformBlock<int, int> object calls the TrySolution method.
        Dim action As Func(Of Integer, Integer) = Function(n) TrySolution(n, cts.Token)
        Dim trySolution1 = New TransformBlock(Of Integer, Integer)(action)
        Dim trySolution2 = New TransformBlock(Of Integer, Integer)(action)
        Dim trySolution3 = New TransformBlock(Of Integer, Integer)(action)

        ' Post data to each TransformBlock<int, int> object.
        trySolution1.Post(11)
        trySolution2.Post(21)
        trySolution3.Post(31)

        ' Call the ReceiveFromAny<T> method to receive the result from the
        ' first TransformBlock<int, int> object to finish.
        Dim result As Integer = ReceiveFromAny(trySolution1, trySolution2, trySolution3)

        ' Cancel all calls to TrySolution that are still active.
        cts.Cancel()

        ' Print the result to the console.
        Console.WriteLine("The solution is {0}.", result)

        cts.Dispose()
    End Sub
End Class

' Sample output:
'The solution is 53.
'
```

To receive the value from the first [TransformBlock<TInput,TOutput>](#) object that finishes, this example defines the

`ReceiveFromAny(T)` method. The `ReceiveFromAny(T)` method accepts an array of `ISourceBlock<TOutput>` objects and links each of these objects to a `WriteOnceBlock<T>` object. When you use the `LinkTo` method to link a source dataflow block to a target block, the source propagates messages to the target as data becomes available. Because the `WriteOnceBlock<T>` class accepts only the first message that it is offered, the `ReceiveFromAny(T)` method produces its result by calling the `Receive` method. This produces the first message that is offered to the `WriteOnceBlock<T>` object. The `LinkTo` method has an overloaded version that takes an `DataflowLinkOptions` object with a `MaxMessages` property that, when it is set to `1`, instructs the source block to unlink from the target after the target receives one message from the source. It is important for the `WriteOnceBlock<T>` object to unlink from its sources because the relationship between the array of sources and the `WriteOnceBlock<T>` object is no longer required after the `WriteOnceBlock<T>` object receives a message.

To enable the remaining calls to `TrySolution` to end after one of them computes a value, the `TrySolution` method takes a `CancellationToken` object that is canceled after the call to `ReceiveFromAny(T)` returns. The `SpinUntil` method returns when this `CancellationToken` object is canceled.

See also

- [Dataflow](#)

Walkthrough: Using Dataflow in a Windows Forms Application

9/20/2022 • 16 minutes to read • [Edit Online](#)

This document demonstrates how to create a network of dataflow blocks that perform image processing in a Windows Forms application.

This example loads image files from the specified folder, creates a composite image, and displays the result. The example uses the dataflow model to route images through the network. In the dataflow model, independent components of a program communicate with one another by sending messages. When a component receives a message, it performs some action and then passes the result to another component. Compare this with the control flow model, in which an application uses control structures, for example, conditional statements, loops, and so on, to control the order of operations in a program.

Prerequisites

Read [Dataflow](#) before you start this walkthrough.

NOTE

The TPL Dataflow Library (the `System.Threading.Tasks.Dataflow` namespace) is not distributed with .NET. To install the `System.Threading.Tasks.Dataflow` namespace in Visual Studio, open your project, choose **Manage NuGet Packages** from the **Project** menu, and search online for the `System.Threading.Tasks.Dataflow` package. Alternatively, to install it using the [.NET Core CLI](#), run `dotnet add package System.Threading.Tasks.Dataflow`.

Sections

This walkthrough contains the following sections:

- [Creating the Windows Forms Application](#)
- [Creating the Dataflow Network](#)
- [Connecting the Dataflow Network to the User Interface](#)
- [The Complete Example](#)

Creating the Windows Forms Application

This section describes how to create the basic Windows Forms application and add controls to the main form.

To Create the Windows Forms Application

1. In Visual Studio, create a Visual C# or Visual Basic **Windows Forms Application** project. In this document, the project is named `CompositeImages`.
2. On the form designer for the main form, `Form1.cs` (`Form1.vb` for Visual Basic), add a **ToolStrip** control.
3. Add a **ToolStripButton** control to the **ToolStrip** control. Set the **DisplayStyle** property to **Text** and the **Text** property to **Choose Folder**.
4. Add a second **ToolStripButton** control to the **ToolStrip** control. Set the **DisplayStyle** property to **Text**, the

`Text` property to `Cancel`, and the `Enabled` property to `False`.

5. Add a `PictureBox` object to the main form. Set the `Dock` property to `Fill`.

Creating the Dataflow Network

This section describes how to create the dataflow network that performs image processing.

To Create the Dataflow Network

1. Add a reference to `System.Threading.Tasks.Dataflow.dll` to your project.
2. Ensure that `Form1.cs` (`Form1.vb` for Visual Basic) contains the following `using` (`Using` in Visual Basic) statements:

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Drawing.Imaging;
using System.IO;
using System.Linq;
using System.Threading;
using System.Threading.Tasks;
using System.Threading.Tasks.Dataflow;
using System.Windows.Forms;
```

3. Add the following data members to the `Form1` class:

```
// The head of the dataflow network.
ITargetBlock<string> headBlock = null;

// Enables the user interface to signal cancellation to the network.
CancellationTokenSource cancellationTokenSource;
```

4. Add the following method, `CreateImageProcessingNetwork`, to the `Form1` class. This method creates the image processing network.

```
// Creates the image processing dataflow network and returns the
// head node of the network.
ITargetBlock<string> CreateImageProcessingNetwork()
{
    //
    // Create the dataflow blocks that form the network.
    //

    // Create a dataflow block that takes a folder path as input
    // and returns a collection of Bitmap objects.
    var loadBitmaps = new TransformBlock<string, IEnumerable<Bitmap>>(path =>
    {
        try
        {
            return LoadBitmaps(path);
        }
        catch (OperationCanceledException)
        {
            // Handle cancellation by passing the empty collection
            // to the next stage of the network.
            return Enumerable.Empty<Bitmap>();
        }
    });
}

// Create a dataflow block that takes a collection of Bitmap objects
```

```

// and returns a single composite bitmap.
var createCompositeBitmap = new TransformBlock<IEnumerable<Bitmap>, Bitmap>(bitmaps =>
{
    try
    {
        return CreateCompositeBitmap(bitmaps);
    }
    catch (OperationCanceledException)
    {
        // Handle cancellation by passing null to the next stage
        // of the network.
        return null;
    }
});

// Create a dataflow block that displays the provided bitmap on the form.
var displayCompositeBitmap = new ActionBlock<Bitmap>(bitmap =>
{
    // Display the bitmap.
    pictureBox1.SizeMode = PictureBoxSizeMode.StretchImage;
    pictureBox1.Image = bitmap;

    // Enable the user to select another folder.
    toolStripButton1.Enabled = true;
    toolStripButton2.Enabled = false;
    Cursor = DefaultCursor;
},
// Specify a task scheduler from the current synchronization context
// so that the action runs on the UI thread.
new ExecutionDataflowBlockOptions
{
    TaskScheduler = TaskScheduler.FromCurrentSynchronizationContext()
});

// Create a dataflow block that responds to a cancellation request by
// displaying an image to indicate that the operation is cancelled and
// enables the user to select another folder.
var operationCancelled = new ActionBlock<object>(delegate
{
    // Display the error image to indicate that the operation
    // was cancelled.
    pictureBox1.SizeMode = PictureBoxSizeMode.CenterImage;
    pictureBox1.Image = pictureBox1.ErrorImage;

    // Enable the user to select another folder.
    toolStripButton1.Enabled = true;
    toolStripButton2.Enabled = false;
    Cursor = DefaultCursor;
},
// Specify a task scheduler from the current synchronization context
// so that the action runs on the UI thread.
new ExecutionDataflowBlockOptions
{
    TaskScheduler = TaskScheduler.FromCurrentSynchronizationContext()
});

//
// Connect the network.
//

// Link loadBitmaps to createCompositeBitmap.
// The provided predicate ensures that createCompositeBitmap accepts the
// collection of bitmaps only if that collection has at least one member.
loadBitmaps.LinkTo(createCompositeBitmap, bitmaps => bitmaps.Count() > 0);

// Also link loadBitmaps to operationCancelled.
// When createCompositeBitmap rejects the message, loadBitmaps
// offers the message to operationCancelled.
// operationCancelled accepts all messages because we do not provide a

```

```

    // operationCancelled accepts all messages because we do not provide a
    // predicate.
    loadBitmaps.LinkTo(operationCancelled);

    // Link createCompositeBitmap to displayCompositeBitmap.
    // The provided predicate ensures that displayCompositeBitmap accepts the
    // bitmap only if it is non-null.
    createCompositeBitmap.LinkTo(displayCompositeBitmap, bitmap => bitmap != null);

    // Also link createCompositeBitmap to operationCancelled.
    // When displayCompositeBitmap rejects the message, createCompositeBitmap
    // offers the message to operationCancelled.
    // operationCancelled accepts all messages because we do not provide a
    // predicate.
    createCompositeBitmap.LinkTo(operationCancelled);

    // Return the head of the network.
    return loadBitmaps;
}

```

5. Implement the `LoadBitmaps` method.

```

// Loads all bitmap files that exist at the provided path.
IEnumerable<Bitmap> LoadBitmaps(string path)
{
    List<Bitmap> bitmaps = new List<Bitmap>();

    // Load a variety of image types.
    foreach (string bitmapType in
        new string[] { "*.bmp", "*gif", "*jpg", "*png", "*tif" })
    {
        // Load each bitmap for the current extension.
        foreach (string fileName in Directory.GetFiles(path, bitmapType))
        {
            // Throw OperationCanceledException if cancellation is requested.
            cancellationTokenSource.Token.ThrowIfCancellationRequested();

            try
            {
                // Add the Bitmap object to the collection.
                bitmaps.Add(new Bitmap(fileName));
            }
            catch (Exception)
            {
                // TODO: A complete application might handle the error.
            }
        }
    }
    return bitmaps;
}

```

6. Implement the `CreateCompositeBitmap` method.

```

// Creates a composite bitmap from the provided collection of Bitmap objects.
// This method computes the average color of each pixel among all bitmaps
// to create the composite image.
Bitmap CreateCompositeBitmap(IEnumerable<Bitmap> bitmaps)
{
    Bitmap[] bitmapArray = bitmaps.ToArray();

    // Compute the maximum width and height components of all
    // bitmaps in the collection.
    Rectangle largest = new Rectangle();
    foreach (var bitmap in bitmapArray)
    {
        if (bitmap.Width > largest.Width)

```

```

        largest.Width = bitmap.Width;
        if (bitmap.Height > largest.Height)
            largest.Height = bitmap.Height;
    }

    // Create a 32-bit Bitmap object with the greatest dimensions.
    Bitmap result = new Bitmap(largest.Width, largest.Height,
        PixelFormat.Format32bppArgb);

    // Lock the result Bitmap.
    var resultBitmapData = result.LockBits(
        new Rectangle(new Point(), result.Size), ImageLockMode.WriteOnly,
        result.PixelFormat);

    // Lock each source bitmap to create a parallel list of BitmapData objects.
    var bitmapDataList = (from bitmap in bitmapArray
        select bitmap.LockBits(
            new Rectangle(new Point(), bitmap.Size),
            ImageLockMode.ReadOnly, PixelFormat.Format32bppArgb))
        .ToList();

    // Compute each column in parallel.
    Parallel.For(0, largest.Width, new ParallelOptions
    {
        CancellationToken = cancellationTokenSource.Token
    },
    i =>
    {
        // Compute each row.
        for (int j = 0; j < largest.Height; j++)
        {
            // Counts the number of bitmaps whose dimensions
            // contain the current location.
            int count = 0;

            // The sum of all alpha, red, green, and blue components.
            int a = 0, r = 0, g = 0, b = 0;

            // For each bitmap, compute the sum of all color components.
            foreach (var bitmapData in bitmapDataList)
            {
                // Ensure that we stay within the bounds of the image.
                if (bitmapData.Width > i && bitmapData.Height > j)
                {
                    unsafe
                    {
                        byte* row = (byte*)(bitmapData.Scan0 + (j * bitmapData.Stride));
                        byte* pix = (byte*)(row + (4 * i));
                        a += *pix; pix++;
                        r += *pix; pix++;
                        g += *pix; pix++;
                        b += *pix;
                    }
                    count++;
                }
            }

            //prevent divide by zero in bottom right pixelless corner
            if (count == 0)
                break;

            unsafe
            {
                // Compute the average of each color component.
                a /= count;
                r /= count;
                g /= count;
                b /= count;
            }
        }
    }
}

```

```

        // Set the result pixel.
        byte* row = (byte*)(resultBitmapData.Scan0 + (j * resultBitmapData.Stride));
        byte* pix = (byte*)(row + (4 * i));
        *pix = (byte)a; pix++;
        *pix = (byte)r; pix++;
        *pix = (byte)g; pix++;
        *pix = (byte)b;
    }
}

// Unlock the source bitmaps.
for (int i = 0; i < bitmapArray.Length; i++)
{
    bitmapArray[i].UnlockBits(bitmapDataList[i]);
}

// Unlock the result bitmap.
result.UnlockBits(resultBitmapData);

// Return the result.
return result;
}

```

NOTE

The C# version of the `CreateCompositeBitmap` method uses pointers to enable efficient processing of the `System.Drawing.Bitmap` objects. Therefore, you must enable the **Allow unsafe code** option in your project in order to use the `unsafe` keyword. For more information about how to enable unsafe code in a Visual C# project, see [Build Page, Project Designer \(C#\)](#).

The following table describes the members of the network.

MEMBER	TYPE	DESCRIPTION
<code>loadBitmaps</code>	<code>TransformBlock<TInput,TOutput></code>	Takes a folder path as input and produces a collection of <code>Bitmap</code> objects as output.
<code>createCompositeBitmap</code>	<code>TransformBlock<TInput,TOutput></code>	Takes a collection of <code>Bitmap</code> objects as input and produces a composite bitmap as output.
<code>displayCompositeBitmap</code>	<code>ActionBlock<TInput></code>	Displays the composite bitmap on the form.
<code>operationCancelled</code>	<code>ActionBlock<TInput></code>	Displays an image to indicate that the operation is canceled and enables the user to select another folder.

To connect the dataflow blocks to form a network, this example uses the `LinkTo` method. The `LinkTo` method contains an overloaded version that takes a `Predicate<T>` object that determines whether the target block accepts or rejects a message. This filtering mechanism enables message blocks to receive only certain values. In this example, the network can branch in one of two ways. The main branch loads the images from disk, creates the composite image, and displays that image on the form. The alternate branch cancels the current operation. The `Predicate<T>` objects enable the dataflow blocks along the main branch to switch to the alternative branch by rejecting certain messages. For example, if the user cancels the operation, the dataflow block `createCompositeBitmap` produces `null` (`Nothing` in Visual Basic) as its output. The dataflow block

`displayCompositeBitmap` rejects `null` input values, and therefore, the message is offered to `operationCancelled`. The dataflow block `operationCancelled` accepts all messages and therefore, displays an image to indicate that the operation is canceled.

The following illustration shows the image processing network:



Because the `displayCompositeBitmap` and `operationCancelled` dataflow blocks act on the user interface, it is important that these actions occur on the user-interface thread. To accomplish this, during construction, these objects each provide an `ExecutionDataflowBlockOptions` object that has the `TaskScheduler` property set to `TaskScheduler.FromCurrentSynchronizationContext`. The `TaskScheduler.FromCurrentSynchronizationContext` method creates a `TaskScheduler` object that performs work on the current synchronization context. Because the `CreateImageProcessingNetwork` method is called from the handler of the **Choose Folder** button, which runs on the user-interface thread, the actions for the `displayCompositeBitmap` and `operationCancelled` dataflow blocks also run on the user-interface thread.

This example uses a shared cancellation token instead of setting the `CancellationToken` property because the `CancellationToken` property permanently cancels dataflow block execution. A cancellation token enables this example to reuse the same dataflow network multiple times, even when the user cancels one or more operations. For an example that uses `CancellationToken` to permanently cancel the execution of a dataflow block, see [How to: Cancel a Dataflow Block](#).

Connecting the Dataflow Network to the User Interface

This section describes how to connect the dataflow network to the user interface. The creation of the composite image and cancellation of the operation are initiated from the **Choose Folder** and **Cancel** buttons. When the user chooses either of these buttons, the appropriate action is initiated in an asynchronous manner.

To Connect the Dataflow Network to the User Interface

1. On the form designer for the main form, create an event handler for the `Click` event for the **Choose Folder** button.
2. Implement the `Click` event for the **Choose Folder** button.

```

// Event handler for the Choose Folder button.
private void toolStripButton1_Click(object sender, EventArgs e)
{
    // Create a FolderBrowserDialog object to enable the user to
    // select a folder.
    FolderBrowserDialog dlg = new FolderBrowserDialog
    {
        ShowNewFolderButton = false
    };

    // Set the selected path to the common Sample Pictures folder
    // if it exists.
    string initialDirectory = Path.Combine(
        Environment.GetFolderPath(Environment.SpecialFolder.CommonPictures),
        "Sample Pictures");
    if (Directory.Exists(initialDirectory))
    {
        dlg.SelectedPath = initialDirectory;
    }

    // Show the dialog and process the dataflow network.
    if (dlg.ShowDialog() == DialogResult.OK)
    {
        // Create a new CancellationTokenSource object to enable
        // cancellation.
        cancellationTokenSource = new CancellationTokenSource();

        // Create the image processing network if needed.
        headBlock ??= CreateImageProcessingNetwork();

        // Post the selected path to the network.
        headBlock.Post(dlg.SelectedPath);

        // Enable the Cancel button and disable the Choose Folder button.
        toolStripButton1.Enabled = false;
        toolStripButton2.Enabled = true;

        // Show a wait cursor.
        Cursor = Cursors.WaitCursor;
    }
}

```

3. On the form designer for the main form, create an event handler for the [Click](#) event for the **Cancel** button.
4. Implement the [Click](#) event for the **Cancel** button.

```

// Event handler for the Cancel button.
private void toolStripButton2_Click(object sender, EventArgs e)
{
    // Signal the request for cancellation. The current component of
    // the dataflow network will respond to the cancellation request.
    cancellationTokenSource.Cancel();
}

```

The Complete Example

The following example shows the complete code for this walkthrough.

```

using System;
using System.Collections.Generic;
using System.Drawing;

```

```

using System.Drawing.Imaging;
using System.IO;
using System.Linq;
using System.Threading;
using System.Threading.Tasks;
using System.Threading.Tasks.Dataflow;
using System.Windows.Forms;

namespace CompositeImages
{
    public partial class Form1 : Form
    {
        // The head of the dataflow network.
        ITargetBlock<string> headBlock = null;

        // Enables the user interface to signal cancellation to the network.
        CancellationTokenSource cancellationTokenSource;

        public Form1()
        {
            InitializeComponent();
        }

        // Creates the image processing dataflow network and returns the
        // head node of the network.
        ITargetBlock<string> CreateImageProcessingNetwork()
        {
            //
            // Create the dataflow blocks that form the network.
            //

            // Create a dataflow block that takes a folder path as input
            // and returns a collection of Bitmap objects.
            var loadBitmaps = new TransformBlock<string, IEnumerable<Bitmap>>(path =>
            {
                try
                {
                    return LoadBitmaps(path);
                }
                catch (OperationCanceledException)
                {
                    // Handle cancellation by passing the empty collection
                    // to the next stage of the network.
                    return Enumerable.Empty<Bitmap>();
                }
            });
        });

        // Create a dataflow block that takes a collection of Bitmap objects
        // and returns a single composite bitmap.
        var createCompositeBitmap = new TransformBlock<IEnumerable<Bitmap>, Bitmap>(bitmaps =>
        {
            try
            {
                return CreateCompositeBitmap(bitmaps);
            }
            catch (OperationCanceledException)
            {
                // Handle cancellation by passing null to the next stage
                // of the network.
                return null;
            }
        });
    });

    // Create a dataflow block that displays the provided bitmap on the form.
    var displayCompositeBitmap = new ActionBlock<Bitmap>(bitmap =>
    {
        // Display the bitmap.
        pictureBox1.SizeMode = PictureBoxSizeMode.StretchImage;
        pictureBox1.Image = bitmap;
    });
}

```

```

        // Enable the user to select another folder.
        toolStripButton1.Enabled = true;
        toolStripButton2.Enabled = false;
        Cursor = DefaultCursor;
    },
    // Specify a task scheduler from the current synchronization context
    // so that the action runs on the UI thread.
    new ExecutionDataflowBlockOptions
    {
        TaskScheduler = TaskScheduler.FromCurrentSynchronizationContext()
    });

    // Create a dataflow block that responds to a cancellation request by
    // displaying an image to indicate that the operation is cancelled and
    // enables the user to select another folder.
    var operationCancelled = new ActionBlock<object>(delegate
    {
        // Display the error image to indicate that the operation
        // was cancelled.
        pictureBox1.SizeMode = PictureBoxSizeMode.CenterImage;
        pictureBox1.Image = pictureBox1.ErrorImage;

        // Enable the user to select another folder.
        toolStripButton1.Enabled = true;
        toolStripButton2.Enabled = false;
        Cursor = DefaultCursor;
    },
    // Specify a task scheduler from the current synchronization context
    // so that the action runs on the UI thread.
    new ExecutionDataflowBlockOptions
    {
        TaskScheduler = TaskScheduler.FromCurrentSynchronizationContext()
    });

    //
    // Connect the network.
    //

    // Link loadBitmaps to createCompositeBitmap.
    // The provided predicate ensures that createCompositeBitmap accepts the
    // collection of bitmaps only if that collection has at least one member.
    loadBitmaps.LinkTo(createCompositeBitmap, bitmaps => bitmaps.Count() > 0);

    // Also link loadBitmaps to operationCancelled.
    // When createCompositeBitmap rejects the message, loadBitmaps
    // offers the message to operationCancelled.
    // operationCancelled accepts all messages because we do not provide a
    // predicate.
    loadBitmaps.LinkTo(operationCancelled);

    // Link createCompositeBitmap to displayCompositeBitmap.
    // The provided predicate ensures that displayCompositeBitmap accepts the
    // bitmap only if it is non-null.
    createCompositeBitmap.LinkTo(displayCompositeBitmap, bitmap => bitmap != null);

    // Also link createCompositeBitmap to operationCancelled.
    // When displayCompositeBitmap rejects the message, createCompositeBitmap
    // offers the message to operationCancelled.
    // operationCancelled accepts all messages because we do not provide a
    // predicate.
    createCompositeBitmap.LinkTo(operationCancelled);

    // Return the head of the network.
    return loadBitmaps;
}

// Loads all bitmap files that exist at the provided path.
TEnumerable<Bitmap> LoadBitmaps(string path)

```

```

    LoadBitmaps - Loads bitmaps from the specified path.
    {
        List<Bitmap> bitmaps = new List<Bitmap>();

        // Load a variety of image types.
        foreach (string bitmapType in
            new string[] { ".bmp", ".gif", ".jpg", ".png", ".tif" })
        {
            // Load each bitmap for the current extension.
            foreach (string fileName in Directory.GetFiles(path, bitmapType))
            {

                // Throw OperationCanceledException if cancellation is requested.
                cancellationTokenSource.Token.ThrowIfCancellationRequested();

                try
                {
                    // Add the Bitmap object to the collection.
                    bitmaps.Add(new Bitmap(fileName));
                }
                catch (Exception)
                {
                    // TODO: A complete application might handle the error.
                }
            }
        }
        return bitmaps;
    }

// Creates a composite bitmap from the provided collection of Bitmap objects.
// This method computes the average color of each pixel among all bitmaps
// to create the composite image.
Bitmap CreateCompositeBitmap(IEnumerable<Bitmap> bitmaps)
{
    Bitmap[] bitmapArray = bitmaps.ToArray();

    // Compute the maximum width and height components of all
    // bitmaps in the collection.
    Rectangle largest = new Rectangle();
    foreach (var bitmap in bitmapArray)
    {
        if (bitmap.Width > largest.Width)
            largest.Width = bitmap.Width;
        if (bitmap.Height > largest.Height)
            largest.Height = bitmap.Height;
    }

    // Create a 32-bit Bitmap object with the greatest dimensions.
    Bitmap result = new Bitmap(largest.Width, largest.Height,
        PixelFormat.Format32bppArgb);

    // Lock the result Bitmap.
    var resultBitmapData = result.LockBits(
        new Rectangle(new Point(), result.Size), ImageLockMode.WriteOnly,
        result.PixelFormat);

    // Lock each source bitmap to create a parallel list of BitmapData objects.
    var bitmapDataList = (from bitmap in bitmapArray
        select bitmap.LockBits(
            new Rectangle(new Point(), bitmap.Size),
            ImageLockMode.ReadOnly, PixelFormat.Format32bppArgb))
        .ToList();

    // Compute each column in parallel.
    Parallel.For(0, largest.Width, new ParallelOptions
    {
        CancellationToken = cancellationTokenSource.Token
    },
    i =>
    {
        // Compute each row.
    });
}

```

```

// Compute each row.
for (int j = 0; j < largest.Height; j++)
{
    // Counts the number of bitmaps whose dimensions
    // contain the current location.
    int count = 0;

    // The sum of all alpha, red, green, and blue components.
    int a = 0, r = 0, g = 0, b = 0;

    // For each bitmap, compute the sum of all color components.
    foreach (var bitmapData in bitmapDataList)
    {
        // Ensure that we stay within the bounds of the image.
        if (bitmapData.Width > i && bitmapData.Height > j)
        {
            unsafe
            {
                byte* row = (byte*)(bitmapData.Scan0 + (j * bitmapData.Stride));
                byte* pix = (byte*)(row + (4 * i));
                a += *pix; pix++;
                r += *pix; pix++;
                g += *pix; pix++;
                b += *pix;
            }
            count++;
        }
    }

    //prevent divide by zero in bottom right pixelless corner
    if (count == 0)
        break;

    unsafe
    {
        // Compute the average of each color component.
        a /= count;
        r /= count;
        g /= count;
        b /= count;

        // Set the result pixel.
        byte* row = (byte*)(resultBitmapData.Scan0 + (j * resultBitmapData.Stride));
        byte* pix = (byte*)(row + (4 * i));
        *pix = (byte)a; pix++;
        *pix = (byte)r; pix++;
        *pix = (byte)g; pix++;
        *pix = (byte)b;
    }
}

// Unlock the source bitmaps.
for (int i = 0; i < bitmapArray.Length; i++)
{
    bitmapArray[i].UnlockBits(bitmapDataList[i]);
}

// Unlock the result bitmap.
result.UnlockBits(resultBitmapData);

// Return the result.
return result;
}

// Event handler for the Choose Folder button.
private void toolStripButton1_Click(object sender, EventArgs e)
{
    // Create a FolderBrowserDialog object to enable the user to
    ...
}

```

```

// select a folder.
FolderBrowserDialog dlg = new FolderBrowserDialog
{
    ShowNewFolderButton = false
};

// Set the selected path to the common Sample Pictures folder
// if it exists.
string initialDirectory = Path.Combine(
    Environment.GetFolderPath(Environment.SpecialFolder.CommonPictures),
    "Sample Pictures");
if (Directory.Exists(initialDirectory))
{
    dlg.SelectedPath = initialDirectory;
}

// Show the dialog and process the dataflow network.
if (dlg.ShowDialog() == DialogResult.OK)
{
    // Create a new CancellationTokenSource object to enable
    // cancellation.
    cancellationTokenSource = new CancellationTokenSource();

    // Create the image processing network if needed.
    headBlock ??= CreateImageProcessingNetwork();

    // Post the selected path to the network.
    headBlock.Post(dlg.SelectedPath);

    // Enable the Cancel button and disable the Choose Folder button.
    toolStripButton1.Enabled = false;
    toolStripButton2.Enabled = true;

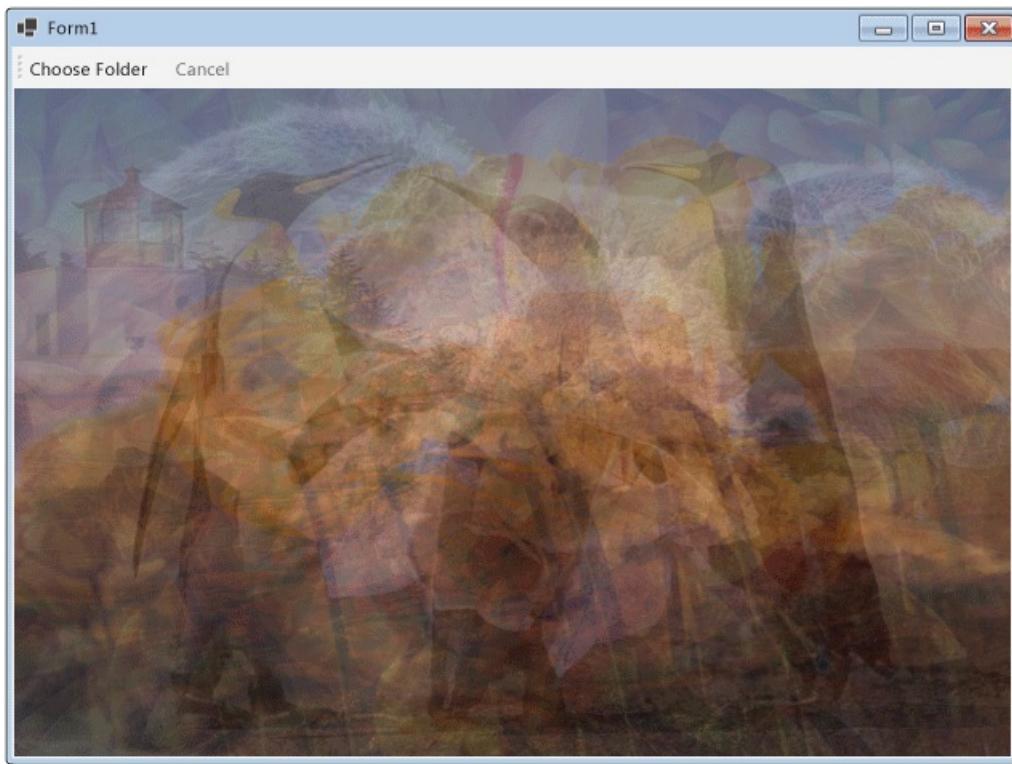
    // Show a wait cursor.
    Cursor = Cursors.WaitCursor;
}
}

// Event handler for the Cancel button.
private void toolStripButton2_Click(object sender, EventArgs e)
{
    // Signal the request for cancellation. The current component of
    // the dataflow network will respond to the cancellation request.
    cancellationTokenSource.Cancel();
}

~Form1()
{
    cancellationTokenSource.Dispose();
}
}

```

The following illustration shows typical output for the common \Sample Pictures\ folder.



See also

- [Dataflow](#)

How to: Cancel a Dataflow Block

9/20/2022 • 15 minutes to read • [Edit Online](#)

This document demonstrates how to enable cancellation in your application. This example uses Windows Forms to show where work items are active in a dataflow pipeline and also the effects of cancellation.

NOTE

The TPL Dataflow Library (the `System.Threading.Tasks.Dataflow` namespace) is not distributed with .NET. To install the `System.Threading.Tasks.Dataflow` namespace in Visual Studio, open your project, choose **Manage NuGet Packages** from the **Project** menu, and search online for the `System.Threading.Tasks.Dataflow` package. Alternatively, to install it using the [.NET Core CLI](#), run `dotnet add package System.Threading.Tasks.Dataflow`.

To Create the Windows Forms Application

1. Create a C# or Visual Basic **Windows Forms Application** project. In the following steps, the project is named `CancellationWinForms`.
2. On the form designer for the main form, `Form1.cs` (`Form1.vb` for Visual Basic), add a **ToolStrip** control.
3. Add a **ToolStripButton** control to the **ToolStrip** control. Set the **DisplayStyle** property to **Text** and the **Text** property to **Add Work Items**.
4. Add a second **ToolStripButton** control to the **ToolStrip** control. Set the **DisplayStyle** property to **Text**, the **Text** property to **Cancel**, and the **Enabled** property to `False`.
5. Add four **ToolStripProgressBar** objects to the **ToolStrip** control.

Creating the Dataflow Pipeline

This section describes how to create the dataflow pipeline that processes work items and updates the progress bars.

To Create the Dataflow Pipeline

1. In your project, add a reference to `System.Threading.Tasks.Dataflow.dll`.
2. Ensure that `Form1.cs` (`Form1.vb` for Visual Basic) contains the following `using` statements (`Imports` in Visual Basic).

```
using System;
using System.Threading;
using System.Threading.Tasks;
using System.Threading.Tasks.Dataflow;
using System.Windows.Forms;
```

```
Imports System.Threading
Imports System.Threading.Tasks
Imports System.Threading.Tasks.Dataflow
```

3. Add the `WorkItem` class as an inner type of the `Form1` class.

```
// A placeholder type that performs work.
class WorkItem
{
    // Performs work for the provided number of milliseconds.
    public void DoWork(int milliseconds)
    {
        // For demonstration, suspend the current thread.
        Thread.Sleep(milliseconds);
    }
}
```

```
' A placeholder type that performs work.
Private Class WorkItem
    ' Performs work for the provided number of milliseconds.
    Public Sub DoWork(ByVal milliseconds As Integer)
        ' For demonstration, suspend the current thread.
        Thread.Sleep(milliseconds)
    End Sub
End Class
```

4. Add the following data members to the `Form1` class.

```
// Enables the user interface to signal cancellation.
CancellationTokenSource cancellationSource;

// The first node in the dataflow pipeline.
TransformBlock<WorkItem, WorkItem> startWork;

// The second, and final, node in the dataflow pipeline.
ActionBlock<WorkItem> completeWork;

// Increments the value of the provided progress bar.
ActionBlock<ToolStripProgressBar> incrementProgress;

// Decrements the value of the provided progress bar.
ActionBlock<ToolStripProgressBar> decrementProgress;

// Enables progress bar actions to run on the UI thread.
TaskScheduler uiTaskScheduler;
```

```
' Enables the user interface to signal cancellation.
Private cancellationSource As CancellationTokenSource

' The first node in the dataflow pipeline.
Private startWork As TransformBlock(Of WorkItem, WorkItem)

' The second, and final, node in the dataflow pipeline.
Private completeWork As ActionBlock(Of WorkItem)

' Increments the value of the provided progress bar.
Private incrementProgress As ActionBlock(Of ToolStripProgressBar)

' Decrements the value of the provided progress bar.
Private decrementProgress As ActionBlock(Of ToolStripProgressBar)

' Enables progress bar actions to run on the UI thread.
Private uiTaskScheduler As TaskScheduler
```

5. Add the following method, `CreatePipeline`, to the `Form1` class.

```
// Creates the blocks that participate in the dataflow pipeline.
```

```
// Creates the blocks that participate in the execution pipeline.
private void CreatePipeline()
{
    // Create the cancellation source.
    cancellationSource = new CancellationTokenSource();

    // Create the first node in the pipeline.
    startWork = new TransformBlock<WorkItem, WorkItem>(workItem =>
    {
        // Perform some work.
        workItem.DoWork(250);

        // Decrement the progress bar that tracks the count of
        // active work items in this stage of the pipeline.
        decrementProgress.Post(toolStripProgressBar1);

        // Increment the progress bar that tracks the count of
        // active work items in the next stage of the pipeline.
        incrementProgress.Post(toolStripProgressBar2);

        // Send the work item to the next stage of the pipeline.
        return workItem;
    },
    new ExecutionDataflowBlockOptions
    {
        CancellationToken = cancellationSource.Token
    });

    // Create the second, and final, node in the pipeline.
    completeWork = new ActionBlock<WorkItem>(workItem =>
    {
        // Perform some work.
        workItem.DoWork(1000);

        // Decrement the progress bar that tracks the count of
        // active work items in this stage of the pipeline.
        decrementProgress.Post(toolStripProgressBar2);

        // Increment the progress bar that tracks the overall
        // count of completed work items.
        incrementProgress.Post(toolStripProgressBar3);
    },
    new ExecutionDataflowBlockOptions
    {
        CancellationToken = cancellationSource.Token,
        MaxDegreeOfParallelism = 2
    });

    // Connect the two nodes of the pipeline. When the first node completes,
    // set the second node also to the completed state.
    startWork.LinkTo(
        completeWork, new DataflowLinkOptions { PropagateCompletion = true });

    // Create the dataflow action blocks that increment and decrement
    // progress bars.
    // These blocks use the task scheduler that is associated with
    // the UI thread.

    incrementProgress = new ActionBlock<ToolStripProgressBar>(
        progressBar => progressBar.Value++,
        new ExecutionDataflowBlockOptions
        {
            CancellationToken = cancellationSource.Token,
            TaskScheduler = uiTaskScheduler
        });

    decrementProgress = new ActionBlock<ToolStripProgressBar>(
        progressBar => progressBar.Value--,
        new ExecutionDataflowBlockOptions
        {
            CancellationToken = cancellationSource.Token,
            TaskScheduler = uiTaskScheduler
        });
}
```

```
        {
            CancellationToken = cancellationSource.Token,
            TaskScheduler = uiTaskScheduler
        });
    }
}
```

```

' Creates the blocks that participate in the dataflow pipeline.
Private Sub CreatePipeline()
    ' Create the cancellation source.
    cancellationSource = New CancellationTokenSource()

    ' Create the first node in the pipeline.
    startWork = New TransformBlock(Of WorkItem, WorkItem)(Function(workItem)
        ' Perform some work.
        ' Decrement the progress bar that
        tracks the count of
            ' active work items in this stage of
            the pipeline.
        ' Increment the progress bar that
        tracks the count of
            ' active work items in the next stage
            of the pipeline.
        ' Send the work item to the next stage
        of the pipeline.
        workItem.DoWork(250)

decrementProgress.Post(toolStripProgressBar1)

incrementProgress.Post(toolStripProgressBar2)
    Return workItem
End Function,
New ExecutionDataflowBlockOptions With {.CancellationToken = cancellationSource.Token})

' Create the second, and final, node in the pipeline.
completeWork = New ActionBlock(Of WorkItem)(Sub(workItem)
    ' Perform some work.
    ' Decrement the progress bar that tracks the
    count of
        ' active work items in this stage of the
        pipeline.
    ' Increment the progress bar that tracks the
    overall
        ' count of completed work items.
        workItem.DoWork(1000)
        decrementProgress.Post(toolStripProgressBar2)
        incrementProgress.Post(toolStripProgressBar3)
    End Sub,
New ExecutionDataflowBlockOptions With {.CancellationToken = cancellationSource.Token,
    .MaxDegreeOfParallelism = 2})

' Connect the two nodes of the pipeline. When the first node completes,
' set the second node also to the completed state.
startWork.LinkTo(
    completeWork, New DataflowLinkOptions With {.PropagateCompletion = true})

' Create the dataflow action blocks that increment and decrement
' progress bars.
' These blocks use the task scheduler that is associated with
' the UI thread.

incrementProgress = New ActionBlock(Of ToolStripProgressBar)(
    Sub(progressBar) progressBar.Value += 1,
    New ExecutionDataflowBlockOptions With {.CancellationToken = cancellationSource.Token,
        .TaskScheduler = uiTaskScheduler})

decrementProgress = New ActionBlock(Of ToolStripProgressBar)(
    Sub(progressBar) progressBar.Value -= 1,
    New ExecutionDataflowBlockOptions With {.CancellationToken = cancellationSource.Token,
        .TaskScheduler = uiTaskScheduler})

End Sub

```

Because the `incrementProgress` and `decrementProgress` dataflow blocks act on the user interface, it is important that these actions occur on the user-interface thread. To accomplish this, during construction these objects each provide an `ExecutionDataflowBlockOptions` object that has the `TaskScheduler` property set to `TaskScheduler.FromCurrentSynchronizationContext`. The `TaskScheduler.FromCurrentSynchronizationContext` method creates a `TaskScheduler` object that performs work on the current synchronization context. Because the `Form1` constructor is called from the user-interface thread, the actions for the `incrementProgress` and `decrementProgress` dataflow blocks also run on the user-interface thread.

This example sets the `CancellationToken` property when it constructs the members of the pipeline. Because the `CancellationToken` property permanently cancels dataflow block execution, the whole pipeline must be recreated after the user cancels the operation and then wants to add more work items to the pipeline. For an example that demonstrates an alternative way to cancel a dataflow block so that other work can be performed after an operation is canceled, see [Walkthrough: Using Dataflow in a Windows Forms Application](#).

Connecting the Dataflow Pipeline to the User Interface

This section describes how to connect the dataflow pipeline to the user interface. Both creating the pipeline and adding work items to the pipeline are controlled by the event handler for the **Add Work Items** button. Cancellation is initiated by the **Cancel** button. When the user clicks either of these buttons, the appropriate action is initiated in an asynchronous manner.

To Connect the Dataflow Pipeline to the User Interface

1. On the form designer for the main form, create an event handler for the `Click` event for the **Add Work Items** button.
2. Implement the `Click` event for the **Add Work Items** button.

```
// Event handler for the Add Work Items button.
private void toolStripButton1_Click(object sender, EventArgs e)
{
    // The Cancel button is disabled when the pipeline is not active.
    // Therefore, create the pipeline and enable the Cancel button
    // if the Cancel button is disabled.
    if (!toolStripButton2.Enabled)
    {
        CreatePipeline();

        // Enable the Cancel button.
        toolStripButton2.Enabled = true;
    }

    // Post several work items to the head of the pipeline.
    for (int i = 0; i < 5; i++)
    {
        toolStripProgressBar1.Value++;
        startWork.Post(new WorkItem());
    }
}
```

```

' Event handler for the Add Work Items button.
Private Sub toolStripButton1_Click(ByVal sender As Object, ByVal e As EventArgs) Handles
toolStripButton1.Click
    ' The Cancel button is disabled when the pipeline is not active.
    ' Therefore, create the pipeline and enable the Cancel button
    ' if the Cancel button is disabled.
    If Not toolStripButton2.Enabled Then
        CreatePipeline()

        ' Enable the Cancel button.
        toolStripButton2.Enabled = True
    End If

    ' Post several work items to the head of the pipeline.
    For i As Integer = 0 To 4
        toolStripProgressBar1.Value += 1
        startWork.Post(New WorkItem())
    Next i
End Sub

```

3. On the form designer for the main form, create an event handler for the [Click](#) event handler for the **Cancel** button.
4. Implement the [Click](#) event handler for the **Cancel** button.

```

// Event handler for the Cancel button.
private async void toolStripButton2_Click(object sender, EventArgs e)
{
    // Disable both buttons.
    toolStripButton1.Enabled = false;
    toolStripButton2.Enabled = false;

    // Trigger cancellation.
    cancellationSource.Cancel();

    try
    {
        // Asynchronously wait for the pipeline to complete processing and for
        // the progress bars to update.
        await Task.WhenAll(
            completeWork.Completion,
            incrementProgress.Completion,
            decrementProgress.Completion);
    }
    catch (OperationCanceledException)
    {
    }

    // Increment the progress bar that tracks the number of cancelled
    // work items by the number of active work items.
    toolStripProgressBar4.Value += toolStripProgressBar1.Value;
    toolStripProgressBar4.Value += toolStripProgressBar2.Value;

    // Reset the progress bars that track the number of active work items.
    toolStripProgressBar1.Value = 0;
    toolStripProgressBar2.Value = 0;

    // Enable the Add Work Items button.
    toolStripButton1.Enabled = true;
}

```

```

' Event handler for the Cancel button.
Private Async Sub toolStripButton2_Click(ByVal sender As Object, ByVal e As EventArgs) Handles
toolStripButton2.Click
    ' Disable both buttons.
    toolStripButton1.Enabled = False
    toolStripButton2.Enabled = False

    ' Trigger cancellation.
    cancellationSource.Cancel()

    Try
        ' Asynchronously wait for the pipeline to complete processing and for
        ' the progress bars to update.
        Await Task.WhenAll(completeWork.Completion, incrementProgress.Completion,
decrementProgress.Completion)
    Catch e1 As OperationCanceledException
    End Try

    ' Increment the progress bar that tracks the number of cancelled
    ' work items by the number of active work items.
    toolStripProgressBar4.Value += toolStripProgressBar1.Value
    toolStripProgressBar4.Value += toolStripProgressBar2.Value

    ' Reset the progress bars that track the number of active work items.
    toolStripProgressBar1.Value = 0
    toolStripProgressBar2.Value = 0

    ' Enable the Add Work Items button.
    toolStripButton1.Enabled = True
End Sub

```

Example

The following example shows the complete code for Form1.cs (Form1.vb for Visual Basic).

```

using System;
using System.Threading;
using System.Threading.Tasks;
using System.Threading.Tasks.Dataflow;
using System.Windows.Forms;

namespace CancellationWinForms
{
    public partial class Form1 : Form
    {
        // A placeholder type that performs work.
        class WorkItem
        {
            // Performs work for the provided number of milliseconds.
            public void DoWork(int milliseconds)
            {
                // For demonstration, suspend the current thread.
                Thread.Sleep(milliseconds);
            }
        }

        // Enables the user interface to signal cancellation.
        CancellationTokenSource cancellationSource;

        // The first node in the dataflow pipeline.
        TransformBlock<WorkItem, WorkItem> startWork;

        // The second, and final, node in the dataflow pipeline.
        ActionBlock<WorkItem> completeWork;
    }
}

```

```

// Increments the value of the provided progress bar.
ActionBlock<ToolStripProgressBar> incrementProgress;

// Decrements the value of the provided progress bar.
ActionBlock<ToolStripProgressBar> decrementProgress;

// Enables progress bar actions to run on the UI thread.
TaskScheduler uiTaskScheduler;

public Form1()
{
    InitializeComponent();

    // Create the UI task scheduler from the current synchronization
    // context.
    uiTaskScheduler = TaskScheduler.FromCurrentSynchronizationContext();
}

// Creates the blocks that participate in the dataflow pipeline.
private void CreatePipeline()
{
    // Create the cancellation source.
    cancellationSource = new CancellationTokenSource();

    // Create the first node in the pipeline.
    startWork = new TransformBlock<WorkItem, WorkItem>(workItem =>
    {
        // Perform some work.
        workItem.DoWork(250);

        // Decrement the progress bar that tracks the count of
        // active work items in this stage of the pipeline.
        decrementProgress.Post(toolStripProgressBar1);

        // Increment the progress bar that tracks the count of
        // active work items in the next stage of the pipeline.
        incrementProgress.Post(toolStripProgressBar2);

        // Send the work item to the next stage of the pipeline.
        return workItem;
    },
    new ExecutionDataflowBlockOptions
    {
        CancellationToken = cancellationSource.Token
    });

    // Create the second, and final, node in the pipeline.
    completeWork = new ActionBlock<WorkItem>(workItem =>
    {
        // Perform some work.
        workItem.DoWork(1000);

        // Decrement the progress bar that tracks the count of
        // active work items in this stage of the pipeline.
        decrementProgress.Post(toolStripProgressBar2);

        // Increment the progress bar that tracks the overall
        // count of completed work items.
        incrementProgress.Post(toolStripProgressBar3);
    },
    new ExecutionDataflowBlockOptions
    {
        CancellationToken = cancellationSource.Token,
        MaxDegreeOfParallelism = 2
    });

    // Connect the two nodes of the pipeline. When the first node completes,
    // set the second node also to the completed state.
    startWork.LinkTo(

```

```

        completeWork, new DataflowLinkOptions { PropagateCompletion = true });

    // Create the dataflow action blocks that increment and decrement
    // progress bars.
    // These blocks use the task scheduler that is associated with
    // the UI thread.

    incrementProgress = new ActionBlock<ToolStripProgressBar>(
        progressBar => progressBar.Value++,
        new ExecutionDataflowBlockOptions
    {
        CancellationToken = cancellationSource.Token,
        TaskScheduler = uiTaskScheduler
    });

    decrementProgress = new ActionBlock<ToolStripProgressBar>(
        progressBar => progressBar.Value--,
        new ExecutionDataflowBlockOptions
    {
        CancellationToken = cancellationSource.Token,
        TaskScheduler = uiTaskScheduler
    });
}

// Event handler for the Add Work Items button.
private void toolStripButton1_Click(object sender, EventArgs e)
{
    // The Cancel button is disabled when the pipeline is not active.
    // Therefore, create the pipeline and enable the Cancel button
    // if the Cancel button is disabled.
    if (!toolStripButton2.Enabled)
    {
        CreatePipeline();

        // Enable the Cancel button.
        toolStripButton2.Enabled = true;
    }

    // Post several work items to the head of the pipeline.
    for (int i = 0; i < 5; i++)
    {
        toolStripProgressBar1.Value++;
        startWork.Post(new WorkItem());
    }
}

// Event handler for the Cancel button.
private async void toolStripButton2_Click(object sender, EventArgs e)
{
    // Disable both buttons.
    toolStripButton1.Enabled = false;
    toolStripButton2.Enabled = false;

    // Trigger cancellation.
    cancellationSource.Cancel();

    try
    {
        // Asynchronously wait for the pipeline to complete processing and for
        // the progress bars to update.
        await Task.WhenAll(
            completeWork.Completion,
            incrementProgress.Completion,
            decrementProgress.Completion);
    }
    catch (OperationCanceledException)
    {
    }
}

```

```

        // Increment the progress bar that tracks the number of cancelled
        // work items by the number of active work items.
        toolStripProgressBar4.Value += toolStripProgressBar1.Value;
        toolStripProgressBar4.Value += toolStripProgressBar2.Value;

        // Reset the progress bars that track the number of active work items.
        toolStripProgressBar1.Value = 0;
        toolStripProgressBar2.Value = 0;

        // Enable the Add Work Items button.
        toolStripButton1.Enabled = true;
    }

    ~Form1()
    {
        cancellationSource.Dispose();
    }
}

```

```

Imports System.Threading
Imports System.Threading.Tasks
Imports System.Threading.Tasks.Dataflow

Namespace CancellationWinForms
    Partial Public Class Form1
        Inherits Form
        ' A placeholder type that performs work.
        Private Class WorkItem
            ' Performs work for the provided number of milliseconds.
            Public Sub DoWork(ByVal milliseconds As Integer)
                ' For demonstration, suspend the current thread.
                Thread.Sleep(milliseconds)
            End Sub
        End Class

        ' Enables the user interface to signal cancellation.
        Private cancellationSource As CancellationTokenSource

        ' The first node in the dataflow pipeline.
        Private startWork As TransformBlock(Of WorkItem, WorkItem)

        ' The second, and final, node in the dataflow pipeline.
        Private completeWork As ActionBlock(Of WorkItem)

        ' Increments the value of the provided progress bar.
        Private incrementProgress As ActionBlock(Of ToolStripProgressBar)

        ' Decrement the value of the provided progress bar.
        Private decrementProgress As ActionBlock(Of ToolStripProgressBar)

        ' Enables progress bar actions to run on the UI thread.
        Private uiTaskScheduler As TaskScheduler

        Public Sub New()
            InitializeComponent()

            ' Create the UI task scheduler from the current synchronization
            ' context.
            uiTaskScheduler = TaskScheduler.FromCurrentSynchronizationContext()
        End Sub

        ' Creates the blocks that participate in the dataflow pipeline.
        Private Sub CreatePipeline()
            ' Create the cancellation source.
            cancellationSource = New CancellationTokenSource()

```

```

' Create the first node in the pipeline.
startWork = New TransformBlock(Of WorkItem, WorkItem)(Function(workItem)
    ' Perform some work.
    ' Decrement the progress bar that
    tracks the count of
    ' active work items in this stage of
    the pipeline.
    ' Increment the progress bar that
    tracks the count of
    ' active work items in the next stage
    of the pipeline.
    ' Send the work item to the next stage
    of the pipeline.
    workItem.DoWork(250)

decrementProgress.Post(toolStripProgressBar1)

incrementProgress.Post(toolStripProgressBar2)
    Return workItem
End Function,
New ExecutionDataflowBlockOptions With {.CancellationToken = cancellationSource.Token})
```

' Create the second, and final, node in the pipeline.

```

completeWork = New ActionBlock(Of WorkItem)(Sub(workItem)
    ' Perform some work.
    ' Decrement the progress bar that tracks the
    count of
    ' active work items in this stage of the
    pipeline.
    ' Increment the progress bar that tracks the
    overall
    ' count of completed work items.
    workItem.DoWork(1000)
    decrementProgress.Post(toolStripProgressBar2)
    incrementProgress.Post(toolStripProgressBar3)
End Sub,
New ExecutionDataflowBlockOptions With {.CancellationToken = cancellationSource.Token,
    .MaxDegreeOfParallelism = 2})
```

' Connect the two nodes of the pipeline. When the first node completes,

' set the second node also to the completed state.

```

startWork.LinkTo(
    completeWork, New DataflowLinkOptions With {.PropagateCompletion = true})
```

' Create the dataflow action blocks that increment and decrement

' progress bars.

' These blocks use the task scheduler that is associated with

' the UI thread.

```

incrementProgress = New ActionBlock(Of ToolStripProgressBar)(
    Sub(progressBar) progressBar.Value += 1,
    New ExecutionDataflowBlockOptions With {.CancellationToken = cancellationSource.Token,
        .TaskScheduler = uiTaskScheduler})
```

```

decrementProgress = New ActionBlock(Of ToolStripProgressBar)(
    Sub(progressBar) progressBar.Value -= 1,
    New ExecutionDataflowBlockOptions With {.CancellationToken = cancellationSource.Token,
        .TaskScheduler = uiTaskScheduler})
```

End Sub

' Event handler for the Add Work Items button.

```

Private Sub toolStripButton1_Click(ByVal sender As Object, ByVal e As EventArgs) Handles
toolStripButton1.Click
    ' The Cancel button is disabled when the pipeline is not active.
    ' Therefore, create the pipeline and enable the Cancel button
    ' if the Cancel button is disabled.
    If Not toolStripButton2.Enabled Then
```

```

    If Not toolStripButton2.Enabled Then
        CreatePipeline()

        ' Enable the Cancel button.
        toolStripButton2.Enabled = True
    End If

        ' Post several work items to the head of the pipeline.
        For i As Integer = 0 To 4
            toolStripProgressBar1.Value += 1
            startWork.Post(New WorkItem())
        Next i
    End Sub

        ' Event handler for the Cancel button.
    Private Async Sub toolStripButton2_Click(ByVal sender As Object, ByVal e As EventArgs) Handles
toolStripButton2.Click
        ' Disable both buttons.
        toolStripButton1.Enabled = False
        toolStripButton2.Enabled = False

        ' Trigger cancellation.
        cancellationSource.Cancel()

        Try
            ' Asynchronously wait for the pipeline to complete processing and for
            ' the progress bars to update.
            Await Task.WhenAll(completeWork.Completion, incrementProgress.Completion,
decrementProgress.Completion)
        Catch e1 As OperationCanceledException
        End Try

        ' Increment the progress bar that tracks the number of cancelled
        ' work items by the number of active work items.
        toolStripProgressBar4.Value += toolStripProgressBar1.Value
        toolStripProgressBar4.Value += toolStripProgressBar2.Value

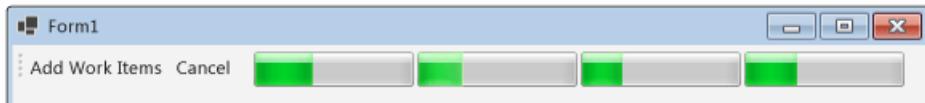
        ' Reset the progress bars that track the number of active work items.
        toolStripProgressBar1.Value = 0
        toolStripProgressBar2.Value = 0

        ' Enable the Add Work Items button.
        toolStripButton1.Enabled = True
    End Sub

    Protected Overrides Sub Finalize()
        cancellationSource.Dispose()
        MyBase.Finalize()
    End Sub
End Class
End Namespace

```

The following illustration shows the running application.



See also

- [Dataflow](#)

Walkthrough: Creating a Custom Dataflow Block Type

9/20/2022 • 20 minutes to read • [Edit Online](#)

Although the TPL Dataflow Library provides several dataflow block types that enable a variety of functionality, you can also create custom block types. This document describes how to create a dataflow block type that implements custom behavior.

Prerequisites

Read [Dataflow](#) before you read this document.

NOTE

The TPL Dataflow Library (the `System.Threading.Tasks.Dataflow` namespace) is not distributed with .NET. To install the `System.Threading.Tasks.Dataflow` namespace in Visual Studio, open your project, choose **Manage NuGet Packages** from the **Project** menu, and search online for the `System.Threading.Tasks.Dataflow` package. Alternatively, to install it using the [.NET Core CLI](#), run `dotnet add package System.Threading.Tasks.Dataflow`.

Defining the Sliding Window Dataflow Block

Consider a dataflow application that requires that input values be buffered and then output in a sliding window manner. For example, for the input values {0, 1, 2, 3, 4, 5} and a window size of three, a sliding window dataflow block produces the output arrays {0, 1, 2}, {1, 2, 3}, {2, 3, 4}, and {3, 4, 5}. The following sections describe two ways to create a dataflow block type that implements this custom behavior. The first technique uses the [Encapsulate](#) method to combine the functionality of an `ISourceBlock<TOutput>` object and an `ITargetBlock<TInput>` object into one propagator block. The second technique defines a class that derives from `IPropagatorBlock<TInput,TOutput>` and combines existing functionality to perform custom behavior.

Using the Encapsulate Method to Define the Sliding Window Dataflow Block

The following example uses the [Encapsulate](#) method to create a propagator block from a target and a source. A propagator block enables a source block and a target block to act as a receiver and sender of data.

This technique is useful when you require custom dataflow functionality, but you do not require a type that provides additional methods, properties, or fields.

```
// Creates a IPropagatorBlock<T, T[]> object propagates data in a
// sliding window fashion.
public static IPropagatorBlock<T, T[]> CreateSlidingWindow<T>(int windowSize)
{
    // Create a queue to hold messages.
    var queue = new Queue<T>();

    // The source part of the propagator holds arrays of size windowSize
    // and propagates data out to any connected targets.
    var source = new BufferBlock<T[]>();

    // The target part receives data and adds them to the queue.
    var target = new ActionBlock<T>(item =>
    {
        // Add the item to the queue.
        queue.Enqueue(item);
        // Remove the oldest item when the queue size exceeds the window size.
        if (queue.Count > windowSize)
            queue.Dequeue();
        // Post the data in the queue to the source block when the queue size
        // equals the window size.
        if (queue.Count == windowSize)
            source.Post(queue.ToArray());
    });

    // When the target is set to the completed state, propagate out any
    // remaining data and set the source to the completed state.
    target.Completion.ContinueWith(delegate
    {
        if (queue.Count > 0 && queue.Count < windowSize)
            source.Post(queue.ToArray());
        source.Complete();
    });
}

// Return a IPropagatorBlock<T, T[]> object that encapsulates the
// target and source blocks.
return DataflowBlock.Encapsulate(target, source);
}
```

```

' Creates a IPropagatorBlock<T, T[]> object propagates data in a
' sliding window fashion.
Public Shared Function CreateSlidingWindow(Of T)(ByVal windowSize As Integer) As IPropagatorBlock(Of T, T())
    ' Create a queue to hold messages.
    Dim queue = New Queue(Of T)()

    ' The source part of the propagator holds arrays of size windowSize
    ' and propagates data out to any connected targets.
    Dim source = New BufferBlock(Of T)()

    ' The target part receives data and adds them to the queue.
    Dim target = New ActionBlock(Of T)(Sub(item)
        ' Add the item to the queue.
        ' Remove the oldest item when the queue size exceeds the window
        size.
        ' Post the data in the queue to the source block when the queue
        size
        ' equals the window size.
        queue.Enqueue(item)
        If queue.Count > windowSize Then
            queue.Dequeue()
        End If
        If queue.Count = windowSize Then
            source.Post(queue.ToArray())
        End If
    End Sub)

    ' When the target is set to the completed state, propagate out any
    ' remaining data and set the source to the completed state.
    target.Completion.ContinueWith(Sub()
        If queue.Count > 0 AndAlso queue.Count < windowSize Then
            source.Post(queue.ToArray())
        End If
        source.Complete()
    End Sub)

    ' Return a IPropagatorBlock<T, T[]> object that encapsulates the
    ' target and source blocks.
    Return DataflowBlock.Encapsulate(target, source)
End Function

```

Deriving from IPropagatorBlock to Define the Sliding Window Dataflow Block

The following example shows the `SlidingWindowBlock` class. This class derives from `IPropagatorBlock<TInput,TOutput>` so that it can act as both a source and a target of data. As in the previous example, the `SlidingWindowBlock` class is built on existing dataflow block types. However, the `SlidingWindowBlock` class also implements the methods that are required by the `ISourceBlock<TOutput>`, `ITargetBlock<TInput>`, and `IDataflowBlock` interfaces. These methods all forward work to the predefined dataflow block type members. For example, the `Post` method defers work to the `m_target` data member, which is also an `ITargetBlock<TInput>` object.

This technique is useful when you require custom dataflow functionality, and also require a type that provides additional methods, properties, or fields. For example, the `SlidingWindowBlock` class also derives from `IReceivableSourceBlock<TOutput>` so that it can provide the `TryReceive` and `TryReceiveAll` methods. The `SlidingWindowBlock` class also demonstrates extensibility by providing the `WindowSize` property, which retrieves the number of elements in the sliding window.

```

// Propagates data in a sliding window fashion.
public class SlidingWindowBlock<T> : IPropagatorBlock<T, T[]>,
    IRreceivableSourceBlock<T>

```

```

{
    // The size of the window.
    private readonly int m_windowSize;
    // The target part of the block.
    private readonly ITargetBlock<T> m_target;
    // The source part of the block.
    private readonly IReceivableSourceBlock<T[]> m_source;

    // Constructs a SlidingWindowBlock object.
    public SlidingWindowBlock(int windowHeight)
    {
        // Create a queue to hold messages.
        var queue = new Queue<T>();

        // The source part of the propagator holds arrays of size windowHeight
        // and propagates data out to any connected targets.
        var source = new BufferBlock<T[]>();

        // The target part receives data and adds them to the queue.
        var target = new ActionBlock<T>(item =>
        {
            // Add the item to the queue.
            queue.Enqueue(item);
            // Remove the oldest item when the queue size exceeds the window size.
            if (queue.Count > windowHeight)
                queue.Dequeue();
            // Post the data in the queue to the source block when the queue size
            // equals the window size.
            if (queue.Count == windowHeight)
                source.Post(queue.ToArray());
        });
    }

    // When the target is set to the completed state, propagate out any
    // remaining data and set the source to the completed state.
    target.Completion.ContinueWith(delegate
    {
        if (queue.Count > 0 && queue.Count < windowHeight)
            source.Post(queue.ToArray());
        source.Complete();
    });

    m_windowSize = windowHeight;
    m_target = target;
    m_source = source;
}

// Retrieves the size of the window.
public int WindowSize { get { return m_windowSize; } }

#region IReceivableSourceBlock<TOutput> members

// Attempts to synchronously receive an item from the source.
public bool TryReceive(Predicate<T[]> filter, out T[] item)
{
    return m_source.TryReceive(filter, out item);
}

// Attempts to remove all available elements from the source into a new
// array that is returned.
public bool TryReceiveAll(out IList<T[]> items)
{
    return m_source.TryReceiveAll(out items);
}

#endregion

#region ISourceBlock<TOutput> members

// Links this dataflow block to the provided target

```

```

// LINKS THIS DATAFLOW BLOCK TO THE PROVIDED TARGET.
public IDisposable LinkTo(ITargetBlock<T[]> target, DataflowLinkOptions linkOptions)
{
    return m_source.LinkTo(target, linkOptions);
}

// Called by a target to reserve a message previously offered by a source
// but not yet consumed by this target.
bool ISourceBlock<T[]>.ReserveMessage(DataflowMessageHeader messageHeader,
    ITargetBlock<T[]> target)
{
    return m_source.ReserveMessage(messageHeader, target);
}

// Called by a target to consume a previously offered message from a source.
T[] ISourceBlock<T[]>.ConsumeMessage(DataflowMessageHeader messageHeader,
    ITargetBlock<T[]> target, out bool messageConsumed)
{
    return m_source.ConsumeMessage(messageHeader,
        target, out messageConsumed);
}

// Called by a target to release a previously reserved message from a source.
void ISourceBlock<T[]>.ReleaseReservation(DataflowMessageHeader messageHeader,
    ITargetBlock<T[]> target)
{
    m_source.ReleaseReservation(messageHeader, target);
}

#endregion

#region ITargetBlock<TInput> Members

// Asynchronously passes a message to the target block, giving the target the
// opportunity to consume the message.
DataflowMessageStatus ITargetBlock<T>.OfferMessage(DataflowMessageHeader messageHeader,
    T messageValue, ISourceBlock<T> source, bool consumeToAccept)
{
    return m_target.OfferMessage(messageHeader,
        messageValue, source, consumeToAccept);
}

#endregion

#region IDataflowBlock Members

// Gets a Task that represents the completion of this dataflow block.
public Task Completion { get { return m_source.Completion; } }

// Signals to this target block that it should not accept any more messages,
// nor consume postponed messages.
public void Complete()
{
    m_target.Complete();
}

public void Fault(Exception error)
{
    m_target.Fault(error);
}

#endregion
}

```

```

' Propagates data in a sliding window fashion.
Public Class SlidingWindowBlock(Of T)
    Implements IPropagatorBlock(Of T, T()), IReceivableSourceBlock(Of T())
    ' The size of the window.

```

```

    THE SIZE OF THE WINDOW.

Private ReadOnly m_windowSize As Integer
' The target part of the block.
Private ReadOnly m_target As ITargetBlock(Of T)
' The source part of the block.
Private ReadOnly m_source As IReceivableSourceBlock(Of T())

' Constructs a SlidingWindowBlock object.
Public Sub New(ByVal windowSize As Integer)
    ' Create a queue to hold messages.
    Dim queue = New Queue(Of T)()

    ' The source part of the propagator holds arrays of size windowSize
    ' and propagates data out to any connected targets.
    Dim source = New BufferBlock(Of T())()

    ' The target part receives data and adds them to the queue.
    Dim target = New ActionBlock(Of T)(Sub(item)
        ' Add the item to the queue.
        ' Remove the oldest item when the queue size exceeds the
        window size.
        ' Post the data in the queue to the source block when the
        queue size
        ' equals the window size.
        queue.Enqueue(item)
        If queue.Count > windowSize Then
            queue.Dequeue()
        End If
        If queue.Count = windowSize Then
            source.Post(queue.ToArray())
        End If
    End Sub)

    ' When the target is set to the completed state, propagate out any
    ' remaining data and set the source to the completed state.
    target.Completion.ContinueWith(Sub()
        If queue.Count > 0 AndAlso queue.Count < windowSize Then
            source.Post(queue.ToArray())
        End If
        source.Complete()
    End Sub)

    m_windowSize = windowSize
    m_target = target
    m_source = source
End Sub

' Retrieves the size of the window.
Public ReadOnly Property WindowSize() As Integer
    Get
        Return m_windowSize
    End Get
End Property

'#Region "IReceivableSourceBlock<TOutput> members"

' Attempts to synchronously receive an item from the source.
Public Function TryReceive(ByVal filter As Predicate(Of T()), <System.Runtime.InteropServices.Out()>
ByRef item() As T) As Boolean Implements IReceivableSourceBlock(Of T()).TryReceive
    Return m_source.TryReceive(filter, item)
End Function

' Attempts to remove all available elements from the source into a new
' array that is returned.
Public Function TryReceiveAll(<System.Runtime.InteropServices.Out()> ByRef items As IList(Of T()))
As Boolean Implements IReceivableSourceBlock(Of T()).TryReceiveAll
    Return m_source.TryReceiveAll(items)
End Function

'#End Region

```

```

#End Region

#Region "ISourceBlock<TOutput> members"

    ' Links this dataflow block to the provided target.
    Public Function LinkTo(ByVal target As ITargetBlock(Of T()), ByVal linkOptions As DataflowLinkOptions) As IDisposable Implements ISourceBlock(Of T()).LinkTo
        Return m_source.LinkTo(target, linkOptions)
    End Function

    ' Called by a target to reserve a message previously offered by a source
    ' but not yet consumed by this target.
    Private Function ReserveMessage(ByVal messageHeader As DataflowMessageHeader, ByVal target As ITargetBlock(Of T())) As Boolean Implements ISourceBlock(Of T()).ReserveMessage
        Return m_source.ReserveMessage(messageHeader, target)
    End Function

    ' Called by a target to consume a previously offered message from a source.
    Private Function ConsumeMessage(ByVal messageHeader As DataflowMessageHeader, ByVal target As ITargetBlock(Of T()), ByRef messageConsumed As Boolean) As T() Implements ISourceBlock(Of T()).ConsumeMessage
        Return m_source.ConsumeMessage(messageHeader, target, messageConsumed)
    End Function

    ' Called by a target to release a previously reserved message from a source.
    Private Sub ReleaseReservation(ByVal messageHeader As DataflowMessageHeader, ByVal target As ITargetBlock(Of T())) Implements ISourceBlock(Of T()).ReleaseReservation
        m_source.ReleaseReservation(messageHeader, target)
    End Sub

#End Region

#Region "ITargetBlock<TInput> members"

    ' Asynchronously passes a message to the target block, giving the target the
    ' opportunity to consume the message.
    Private Function OfferMessage(ByVal messageHeader As DataflowMessageHeader, ByVal messageValue As T, ByVal source As ISourceBlock(Of T), ByVal consumeToAccept As Boolean) As DataflowMessageStatus Implements ITargetBlock(Of T).OfferMessage
        Return m_target.OfferMessage(messageHeader, messageValue, source, consumeToAccept)
    End Function

#End Region

#Region "IDataflowBlock members"

    ' Gets a Task that represents the completion of this dataflow block.
    Public ReadOnly Property Completion() As Task Implements IDataflowBlock.Completion
        Get
            Return m_source.Completion
        End Get
    End Property

    ' Signals to this target block that it should not accept any more messages,
    ' nor consume postponed messages.
    Public Sub Complete() Implements IDataflowBlock.Complete
        m_target.Complete()
    End Sub

    Public Sub Fault(ByVal [error] As Exception) Implements IDataflowBlock.Fault
        m_target.Fault([error])
    End Sub

#End Region
End Class

```

The Complete Example

The following example shows the complete code for this walkthrough. It also demonstrates how to use the both sliding window blocks in a method that writes to the block, reads from it, and prints the results to the console.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Threading.Tasks.Dataflow;

// Demonstrates how to create a custom dataflow block type.
class Program
{
    // Creates a IPropagatorBlock<T, T[]> object propagates data in a
    // sliding window fashion.
    public static IPropagatorBlock<T, T[]> CreateSlidingWindow<T>(int windowHeight)
    {
        // Create a queue to hold messages.
        var queue = new Queue<T>();

        // The source part of the propagator holds arrays of size windowHeight
        // and propagates data out to any connected targets.
        var source = new BufferBlock<T[]>();

        // The target part receives data and adds them to the queue.
        var target = new ActionBlock<T>(item =>
        {
            // Add the item to the queue.
            queue.Enqueue(item);
            // Remove the oldest item when the queue size exceeds the window size.
            if (queue.Count > windowHeight)
                queue.Dequeue();
            // Post the data in the queue to the source block when the queue size
            // equals the window size.
            if (queue.Count == windowHeight)
                source.Post(queue.ToArray());
        });
        // When the target is set to the completed state, propagate out any
        // remaining data and set the source to the completed state.
        target.Completion.ContinueWith(delegate
        {
            if (queue.Count > 0 && queue.Count < windowHeight)
                source.Post(queue.ToArray());
            source.Complete();
        });
        // Return a IPropagatorBlock<T, T[]> object that encapsulates the
        // target and source blocks.
        return DataflowBlock.Encapsulate(target, source);
    }

    // Propagates data in a sliding window fashion.
    public class SlidingWindowBlock<T> : IPropagatorBlock<T, T[]>,
                                            IReceivableSourceBlock<T[]>
    {
        // The size of the window.
        private readonly int m_windowSize;
        // The target part of the block.
        private readonly ITargetBlock<T> m_target;
        // The source part of the block.
        private readonly IReceivableSourceBlock<T[]> m_source;

        // Constructs a SlidingWindowBlock object.
        public SlidingWindowBlock(int windowHeight)
        {
            // Create a queue to hold messages.
            var queue = new Queue<T>();
        }
    }
}
```

```

// The source part of the propagator holds arrays of size windowSize
// and propagates data out to any connected targets.
var source = new BufferBlock<T[]>();

// The target part receives data and adds them to the queue.
var target = new ActionBlock<T>(item =>
{
    // Add the item to the queue.
    queue.Enqueue(item);
    // Remove the oldest item when the queue size exceeds the window size.
    if (queue.Count > windowSize)
        queue.Dequeue();
    // Post the data in the queue to the source block when the queue size
    // equals the window size.
    if (queue.Count == windowSize)
        source.Post(queue.ToArray());
});

// When the target is set to the completed state, propagate out any
// remaining data and set the source to the completed state.
target.Completion.ContinueWith(delegate
{
    if (queue.Count > 0 && queue.Count < windowSize)
        source.Post(queue.ToArray());
    source.Complete();
});

_mWindowSize = windowSize;
_mTarget = target;
_mSource = source;
}

// Retrieves the size of the window.
public int WindowSize { get { return mWindowSize; } }

#region IReceivableSourceBlock<TOutput> members

// Attempts to synchronously receive an item from the source.
public bool TryReceive(Predicate<T[]> filter, out T[] item)
{
    return mSource.TryReceive(filter, out item);
}

// Attempts to remove all available elements from the source into a new
// array that is returned.
public bool TryReceiveAll(out IList<T[]> items)
{
    return mSource.TryReceiveAll(out items);
}

#endregion

#region ISourceBlock<TOutput> members

// Links this dataflow block to the provided target.
public IDisposable LinkTo(ITargetBlock<T[]> target, DataflowLinkOptions linkOptions)
{
    return mSource.LinkTo(target, linkOptions);
}

// Called by a target to reserve a message previously offered by a source
// but not yet consumed by this target.
bool ISourceBlock<T[]>.ReserveMessage(DataflowMessageHeader messageHeader,
    ITargetBlock<T[]> target)
{
    return mSource.ReserveMessage(messageHeader, target);
}

```

```

// Called by a target to consume a previously offered message from a source.
T[] ISourceBlock<T[]>.ConsumeMessage(DataflowMessageHeader messageHeader,
    ITargetBlock<T[]> target, out bool messageConsumed)
{
    return m_source.ConsumeMessage(messageHeader,
        target, out messageConsumed);
}

// Called by a target to release a previously reserved message from a source.
void ISourceBlock<T[]>.ReleaseReservation(DataflowMessageHeader messageHeader,
    ITargetBlock<T[]> target)
{
    m_source.ReleaseReservation(messageHeader, target);
}

#endifregion

#region ITargetBlock<TInput> members

// Asynchronously passes a message to the target block, giving the target the
// opportunity to consume the message.
DataflowMessageStatus ITargetBlock<T>.OfferMessage(DataflowMessageHeader messageHeader,
    T messageValue, ISourceBlock<T> source, bool consumeToAccept)
{
    return m_target.OfferMessage(messageHeader,
        messageValue, source, consumeToAccept);
}

#endifregion

#region IDataflowBlock members

// Gets a Task that represents the completion of this dataflow block.
public Task Completion { get { return m_source.Completion; } }

// Signals to this target block that it should not accept any more messages,
// nor consume postponed messages.
public void Complete()
{
    m_target.Complete();
}

public void Fault(Exception error)
{
    m_target.Fault(error);
}

#endifregion
}

// Demonstrates usage of the sliding window block by sending the provided
// values to the provided propagator block and printing the output of
// that block to the console.
static void DemonstrateSlidingWindow<T>(IPropagatorBlock<T, T[]> slidingWindow,
    IEnumerable<T> values)
{
    // Create an action block that prints arrays of data to the console.
    string windowComma = string.Empty;
    var printWindow = new ActionBlock<T[]>(window =>
    {
        Console.Write(windowComma);
        Console.WriteLine("{");

        string comma = string.Empty;
        foreach (T item in window)
        {
            Console.Write(comma);
            Console.Write(item);
            comma = ",";
        }
    });
}

```

```

        }

        Console.WriteLine("}");

        windowComma = ", ";
    });

    // Link the printer block to the sliding window block.
    slidingWindow.LinkTo(printWindow);

    // Set the printer block to the completed state when the sliding window
    // block completes.
    slidingWindow.Completion.ContinueWith(delegate { printWindow.Complete(); });

    // Print an additional newline to the console when the printer block completes.
    var completion = printWindow.Completion.ContinueWith(delegate { Console.WriteLine(); });

    // Post the provided values to the sliding window block and then wait
    // for the sliding window block to complete.
    foreach (T value in values)
    {
        slidingWindow.Post(value);
    }
    slidingWindow.Complete();

    // Wait for the printer to complete and perform its final action.
    completion.Wait();
}

static void Main(string[] args)
{
    Console.Write("Using the DataflowBlockExtensions.Encapsulate method ");
    Console.WriteLine("(T=int, windowSize=3):");
    DemonstrateSlidingWindow(CreateSlidingWindow<int>(3), Enumerable.Range(0, 10));

    Console.WriteLine();

    var slidingWindow = new SlidingWindowBlock<char>(4);

    Console.Write("Using SlidingWindowBlock<T> ");
    Console.WriteLine("(T=char, windowSize={0}):", slidingWindow.WindowSize);
    DemonstrateSlidingWindow(slidingWindow, from n in Enumerable.Range(65, 10)
                                select (char)n);
}
}

/* Output:
Using the DataflowBlockExtensions.Encapsulate method (T=int, windowSize=3):
{0,1,2}, {1,2,3}, {2,3,4}, {3,4,5}, {4,5,6}, {5,6,7}, {6,7,8}, {7,8,9}

Using SlidingWindowBlock<T> (T=char, windowSize=4):
{A,B,C,D}, {B,C,D,E}, {C,D,E,F}, {D,E,F,G}, {E,F,G,H}, {F,G,H,I}, {G,H,I,J}
*/

```

```

Imports System.Collections.Generic
Imports System.Linq
Imports System.Threading.Tasks
Imports System.Threading.Tasks.Dataflow

' Demonstrates how to create a custom dataflow block type.
Friend Class Program
    ' Creates a IPropagatorBlock<T, T[]> object propagates data in a
    ' sliding window fashion.
    Public Shared Function CreateSlidingWindow(Of T)(ByVal windowSize As Integer) As IPropagatorBlock(Of T,
T())
        ' Create a queue to hold messages.
        Dim queue = New Queue(Of T)()

```

```

' The source part of the propagator holds arrays of size windowSize
' and propagates data out to any connected targets.
Dim source = New BufferBlock(Of T)()

' The target part receives data and adds them to the queue.
Dim target = New ActionBlock(Of T)(Sub(item)
    ' Add the item to the queue.
    ' Remove the oldest item when the queue size exceeds the
window size.
    ' Post the data in the queue to the source block when the
queue size
    ' equals the window size.
    queue.Enqueue(item)
    If queue.Count > windowSize Then
        queue.Dequeue()
    End If
    If queue.Count = windowSize Then
        source.Post(queue.ToArray())
    End If
End Sub)

' When the target is set to the completed state, propagate out any
' remaining data and set the source to the completed state.
target.Completion.ContinueWith(Sub()
    If queue.Count > 0 AndAlso queue.Count < windowSize Then
        source.Post(queue.ToArray())
    End If
    source.Complete()
End Sub)

' Return a IPropagatorBlock<T, T[]> object that encapsulates the
' target and source blocks.
Return DataflowBlock.Encapsulate(target, source)
End Function

' Propagates data in a sliding window fashion.
Public Class SlidingWindowBlock(Of T)
    Implements IPropagatorBlock(Of T, T()), IReceivableSourceBlock(Of T())
    ' The size of the window.
    Private ReadOnly m_windowSize As Integer
    ' The target part of the block.
    Private ReadOnly m_target As ITargetBlock(Of T)
    ' The source part of the block.
    Private ReadOnly m_source As IReceivableSourceBlock(Of T())

    ' Constructs a SlidingWindowBlock object.
    Public Sub New(ByVal windowSize As Integer)
        ' Create a queue to hold messages.
        Dim queue = New Queue(Of T)()

        ' The source part of the propagator holds arrays of size windowSize
        ' and propagates data out to any connected targets.
        Dim source = New BufferBlock(Of T)()

        ' The target part receives data and adds them to the queue.
        Dim target = New ActionBlock(Of T)(Sub(item)
            ' Add the item to the queue.
            ' Remove the oldest item when the queue size exceeds the
window size.
            ' Post the data in the queue to the source block when the
queue size
            ' equals the window size.
            queue.Enqueue(item)
            If queue.Count > windowSize Then
                queue.Dequeue()
            End If
            If queue.Count = windowSize Then
                source.Post(queue.ToArray())
            End If
        End Sub)

        ' When the target is set to the completed state, propagate out any
        ' remaining data and set the source to the completed state.
        target.Completion.ContinueWith(Sub()
            If queue.Count > 0 AndAlso queue.Count < windowSize Then
                source.Post(queue.ToArray())
            End If
            source.Complete()
        End Sub)

        ' Return a IPropagatorBlock<T, T[]> object that encapsulates the
        ' target and source blocks.
        Return DataflowBlock.Encapsulate(target, source)
    End Sub
End Class

```

```

        End If
    End Sub)

    ' When the target is set to the completed state, propagate out any
    ' remaining data and set the source to the completed state.
    target.Completion.ContinueWith(Sub()
        If queue.Count > 0 AndAlso queue.Count < windowSize Then
            source.Post(queue.ToArray())
        End If
        source.Complete()
    End Sub)

    m_windowSize = windowSize
    m_target = target
    m_source = source
End Sub

' Retrieves the size of the window.
Public ReadOnly Property WindowSize() As Integer
    Get
        Return m_windowSize
    End Get
End Property

'#Region "IReceivableSourceBlock<TOutput> members"

    ' Attempts to synchronously receive an item from the source.
    Public Function TryReceive(ByVal filter As Predicate(Of T()), <System.Runtime.InteropServices.Out()>
ByRef item() As T) As Boolean Implements IReceivableSourceBlock(Of T()).TryReceive
        Return m_source.TryReceive(filter, item)
    End Function

    ' Attempts to remove all available elements from the source into a new
    ' array that is returned.
    Public Function TryReceiveAll(<System.Runtime.InteropServices.Out()> ByRef items As IList(Of T()))
As Boolean Implements IReceivableSourceBlock(Of T()).TryReceiveAll
        Return m_source.TryReceiveAll(items)
    End Function

    '#End Region

#Region "ISourceBlock<TOutput> members"

    ' Links this dataflow block to the provided target.
    Public Function LinkTo(ByVal target As ITargetBlock(Of T()), ByVal linkOptions As
DataflowLinkOptions) As IDisposable Implements ISourceBlock(Of T()).LinkTo
        Return m_source.LinkTo(target, linkOptions)
    End Function

    ' Called by a target to reserve a message previously offered by a source
    ' but not yet consumed by this target.
    Private Function ReserveMessage(ByVal messageHeader As DataflowMessageHeader, ByVal target As
ITargetBlock(Of T())) As Boolean Implements ISourceBlock(Of T()).ReserveMessage
        Return m_source.ReserveMessage(messageHeader, target)
    End Function

    ' Called by a target to consume a previously offered message from a source.
    Private Function ConsumeMessage(ByVal messageHeader As DataflowMessageHeader, ByVal target As
ITargetBlock(Of T()), ByRef messageConsumed As Boolean) As T() Implements ISourceBlock(Of
T()).ConsumeMessage
        Return m_source.ConsumeMessage(messageHeader, target, messageConsumed)
    End Function

    ' Called by a target to release a previously reserved message from a source.
    Private Sub ReleaseReservation(ByVal messageHeader As DataflowMessageHeader, ByVal target As
ITargetBlock(Of T())) Implements ISourceBlock(Of T()).ReleaseReservation
        m_source.ReleaseReservation(messageHeader, target)
    End Sub

```

```

#End Region

#Region "ITargetBlock<TInput> members"

    ' Asynchronously passes a message to the target block, giving the target the
    ' opportunity to consume the message.
    Private Function OfferMessage(ByVal messageHeader As DataflowMessageHeader, ByVal messageValue As T,
    ByVal source As ISourceBlock(Of T), ByVal consumeToAccept As Boolean) As DataflowMessageStatus Implements
    ITargetBlock(Of T).OfferMessage
        Return m_target.OfferMessage(messageHeader, messageValue, source, consumeToAccept)
    End Function

#End Region

#Region "IDataflowBlock members"

    ' Gets a Task that represents the completion of this dataflow block.
    Public ReadOnly Property Completion() As Task Implements IDataflowBlock.Completion
        Get
            Return m_source.Completion
        End Get
    End Property

    ' Signals to this target block that it should not accept any more messages,
    ' nor consume postponed messages.
    Public Sub Complete() Implements IDataflowBlock.Complete
        m_target.Complete()
    End Sub

    Public Sub Fault(ByVal [error] As Exception) Implements IDataflowBlock.Fault
        m_target.Fault([error])
    End Sub

#End Region

End Class

' Demonstrates usage of the sliding window block by sending the provided
' values to the provided propagator block and printing the output of
' that block to the console.
Private Shared Sub DemonstrateSlidingWindow(Of T)(ByVal slidingWindow As IPropagatorBlock(Of T, T()),
ByVal values As IEnumerable(Of T))
    ' Create an action block that prints arrays of data to the console.
    Dim windowComma As String = String.Empty
    Dim printWindow = New ActionBlock(Of T())(Sub(window)
                                                Console.WriteLine(windowComma)
                                                Console.Write("{")
                                                Dim comma As String = String.Empty
                                                For Each item As T In window
                                                    Console.Write(comma)
                                                    Console.Write(item)
                                                    comma = ","
                                                Next item
                                                Console.Write("}")
                                                windowComma = ", "
                                            End Sub)

    ' Link the printer block to the sliding window block.
    slidingWindow.LinkTo(printWindow)

    ' Set the printer block to the completed state when the sliding window
    ' block completes.
    slidingWindow.Completion.ContinueWith(Sub() printWindow.Complete())

    ' Print an additional newline to the console when the printer block completes.
    Dim completion = printWindow.Completion.ContinueWith(Sub() Console.WriteLine())

    ' Post the provided values to the sliding window block and then wait
    ' for the sliding window block to complete.
    For Each value As T In values

```

```

    slidingWindow.Post(value)
Next value
slidingWindow.Complete()

' Wait for the printer to complete and perform its final action.
completion.Wait()
End Sub

Shared Sub Main(ByVal args() As String)

    Console.WriteLine("Using the DataflowBlockExtensions.Encapsulate method ")
    Console.WriteLine("(T=int, windowSize=3):")
    DemonstrateSlidingWindow(CreateSlidingWindow(Of Integer)(3), Enumerable.Range(0, 10))

    Console.WriteLine()

    Dim slidingWindow = New SlidingWindowBlock(Of Char)(4)

    Console.WriteLine("Using SlidingWindowBlock<T> ")
    Console.WriteLine("(T=char, windowSize={0}):", slidingWindow.WindowSize)
    DemonstrateSlidingWindow(slidingWindow,
        From n In Enumerable.Range(65, 10) _
        Select ChrW(n))
    End Sub
End Class

' Output:
'Using the DataflowBlockExtensions.Encapsulate method (T=int, windowSize=3):
'{0,1,2}, {1,2,3}, {2,3,4}, {3,4,5}, {4,5,6}, {5,6,7}, {6,7,8}, {7,8,9}
'
'Using SlidingWindowBlock<T> (T=char, windowSize=4):
'{A,B,C,D}, {B,C,D,E}, {C,D,E,F}, {D,E,F,G}, {E,F,G,H}, {F,G,H,I}, {G,H,I,J}
'
```

See also

- [Dataflow](#)

How to: Use JoinBlock to Read Data From Multiple Sources

9/20/2022 • 7 minutes to read • [Edit Online](#)

This document explains how to use the `JoinBlock<T1,T2>` class to perform an operation when data is available from multiple sources. It also demonstrates how to use non-greedy mode to enable multiple join blocks to share a data source more efficiently.

NOTE

The TPL Dataflow Library (the `System.Threading.Tasks.Dataflow` namespace) is not distributed with .NET. To install the `System.Threading.Tasks.Dataflow` namespace in Visual Studio, open your project, choose **Manage NuGet Packages** from the **Project** menu, and search online for the `System.Threading.Tasks.Dataflow` package. Alternatively, to install it using the [.NET Core CLI](#), run `dotnet add package System.Threading.Tasks.Dataflow`.

Example

The following example defines three resource types, `NetworkResource`, `FileResource`, and `MemoryResource`, and performs operations when resources become available. This example requires a `NetworkResource` and `MemoryResource` pair in order to perform the first operation and a `FileResource` and `MemoryResource` pair in order to perform the second operation. To enable these operations to occur when all required resources are available, this example uses the `JoinBlock<T1,T2>` class. When a `JoinBlock<T1,T2>` object receives data from all sources, it propagates that data to its target, which in this example is an `ActionBlock<TInput>` object. Both `JoinBlock<T1,T2>` objects read from a shared pool of `MemoryResource` objects.

```
using System;
using System.Threading;
using System.Threading.Tasks.Dataflow;

// Demonstrates how to use non-greedy join blocks to distribute
// resources among a dataflow network.
class Program
{
    // Represents a resource. A derived class might represent
    // a limited resource such as a memory, network, or I/O
    // device.
    abstract class Resource
    {
    }

    // Represents a memory resource. For brevity, the details of
    // this class are omitted.
    class MemoryResource : Resource
    {
    }

    // Represents a network resource. For brevity, the details of
    // this class are omitted.
    class NetworkResource : Resource
    {
    }

    // Represents a file resource. For brevity, the details of
    // this class are omitted.
    class FileResource : Resource
    {
    }
}
```

```
class FileResource : Resource
{
}

static void Main(string[] args)
{
    // Create three BufferBlock<T> objects. Each object holds a different
    // type of resource.
    var networkResources = new BufferBlock<NetworkResource>();
    var fileResources = new BufferBlock<FileResource>();
    var memoryResources = new BufferBlock<MemoryResource>();

    // Create two non-greedy JoinBlock<T1, T2> objects.
    // The first join works with network and memory resources;
    // the second pool works with file and memory resources.

    var joinNetworkAndMemoryResources =
        new JoinBlock<NetworkResource, MemoryResource>(
            new GroupingDataflowBlockOptions
            {
                Greedy = false
            });

    var joinFileAndMemoryResources =
        new JoinBlock<FileResource, MemoryResource>(
            new GroupingDataflowBlockOptions
            {
                Greedy = false
            });

    // Create two ActionBlock<T> objects.
    // The first block acts on a network resource and a memory resource.
    // The second block acts on a file resource and a memory resource.

    var networkMemoryAction =
        new ActionBlock<Tuple<NetworkResource, MemoryResource>>(
            data =>
            {
                // Perform some action on the resources.

                // Print a message.
                Console.WriteLine("Network worker: using resources...");

                // Simulate a lengthy operation that uses the resources.
                Thread.Sleep(new Random().Next(500, 2000));

                // Print a message.
                Console.WriteLine("Network worker: finished using resources...");

                // Release the resources back to their respective pools.
                networkResources.Post(data.Item1);
                memoryResources.Post(data.Item2);
            });
}

var fileMemoryAction =
    new ActionBlock<Tuple<FileResource, MemoryResource>>(
        data =>
        {
            // Perform some action on the resources.

            // Print a message.
            Console.WriteLine("File worker: using resources...");

            // Simulate a lengthy operation that uses the resources.
            Thread.Sleep(new Random().Next(500, 2000));

            // Print a message.
            Console.WriteLine("File worker: finished using resources...");
        });
}
```

```

        // Release the resources back to their respective pools.
        fileResources.Post(data.Item1);
        memoryResources.Post(data.Item2);
    });

    // Link the resource pools to the JoinBlock<T1, T2> objects.
    // Because these join blocks operate in non-greedy mode, they do not
    // take the resource from a pool until all resources are available from
    // all pools.

    networkResources.LinkTo(joinNetworkAndMemoryResources.Target1);
    memoryResources.LinkTo(joinNetworkAndMemoryResources.Target2);

    fileResources.LinkTo(joinFileAndMemoryResources.Target1);
    memoryResources.LinkTo(joinFileAndMemoryResources.Target2);

    // Link the JoinBlock<T1, T2> objects to the ActionBlock<T> objects.

    joinNetworkAndMemoryResources.LinkTo(networkMemoryAction);
    joinFileAndMemoryResources.LinkTo(fileMemoryAction);

    // Populate the resource pools. In this example, network and
    // file resources are more abundant than memory resources.

    networkResources.Post(new NetworkResource());
    networkResources.Post(new NetworkResource());
    networkResources.Post(new NetworkResource());

    memoryResources.Post(new MemoryResource());

    fileResources.Post(new FileResource());
    fileResources.Post(new FileResource());
    fileResources.Post(new FileResource());

    // Allow data to flow through the network for several seconds.
    Thread.Sleep(10000);
}
}

/* Sample output:
File worker: using resources...
File worker: finished using resources...
Network worker: using resources...
Network worker: finished using resources...
File worker: using resources...
File worker: finished using resources...
Network worker: using resources...
Network worker: finished using resources...
File worker: using resources...
File worker: finished using resources...
File worker: using resources...
File worker: finished using resources...
Network worker: using resources...
Network worker: finished using resources...
Network worker: using resources...
Network worker: finished using resources...
File worker: using resources...
*/

```

```

Imports System.Threading
Imports System.Threading.Tasks.Dataflow

' Demonstrates how to use non-greedy join blocks to distribute
' resources among a dataflow network.
Friend Class Program
    ' Represents a resource. A derived class might represent
    ' a limited resource such as a memory, network, or I/O
    '

```

```

    ' device.

Private MustInherit Class Resource
End Class

' Represents a memory resource. For brevity, the details of
' this class are omitted.
Private Class MemoryResource
    Inherits Resource
End Class

' Represents a network resource. For brevity, the details of
' this class are omitted.
Private Class NetworkResource
    Inherits Resource
End Class

' Represents a file resource. For brevity, the details of
' this class are omitted.
Private Class FileResource
    Inherits Resource
End Class

Shared Sub Main(ByVal args() As String)
    ' Create three BufferBlock<T> objects. Each object holds a different
    ' type of resource.
    Dim networkResources = New BufferBlock(Of NetworkResource)()
    Dim fileResources = New BufferBlock(Of FileResource)()
    Dim memoryResources = New BufferBlock(Of MemoryResource)()

    ' Create two non-greedy JoinBlock<T1, T2> objects.
    ' The first join works with network and memory resources;
    ' the second pool works with file and memory resources.

    Dim joinNetworkAndMemoryResources = New JoinBlock(Of NetworkResource, MemoryResource)(New
GroupingDataflowBlockOptions With {.Greedy = False})

    Dim joinFileAndMemoryResources = New JoinBlock(Of FileResource, MemoryResource)(New
GroupingDataflowBlockOptions With {.Greedy = False})

    ' Create two ActionBlock<T> objects.
    ' The first block acts on a network resource and a memory resource.
    ' The second block acts on a file resource and a memory resource.

    Dim networkMemoryAction = New ActionBlock(Of Tuple(Of NetworkResource, MemoryResource))(Sub(data)

Perform some action on the resources.                                ' Print
a message.                                                       '
                                                               '
Simulate a lengthy operation that uses the resources.          ' Print
a message.                                                       '
                                                               '
Release the resources back to their respective pools.

Console.WriteLine("Network worker: using resources...")

Thread.Sleep(New Random().Next(500, 2000))

Console.WriteLine("Network worker: finished using resources...")

networkResources.Post(data.Item1)

memoryResources.Post(data.Item2)

End Sub)

Dim fileMemoryAction = New ActionBlock(Of Tuple(Of FileResource, MemoryResource))(Sub(data)
    ' Perform some
action on the resources.

```



```
'Network worker: using resources...
'Network worker: finished using resources...
'Network worker: using resources...
'Network worker: finished using resources...
'File worker: using resources...
'
```

To enable efficient use of the shared pool of `MemoryResource` objects, this example specifies a `GroupingDataflowBlockOptions` object that has the `Greedy` property set to `False` to create `JoinBlock<T1,T2>` objects that act in non-greedy mode. A non-greedy join block postpones all incoming messages until one is available from each source. If any of the postponed messages were accepted by another block, the join block restarts the process. Non-greedy mode enables join blocks that share one or more source blocks to make forward progress as the other blocks wait for data. In this example, if a `MemoryResource` object is added to the `memoryResources` pool, the first join block to receive its second data source can make forward progress. If this example were to use greedy mode, which is the default, one join block might take the `MemoryResource` object and wait for the second resource to become available. However, if the other join block has its second data source available, it cannot make forward progress because the `MemoryResource` object has been taken by the other join block.

Robust Programming

The use of non-greedy joins can also help you prevent deadlock in your application. In a software application, *deadlock* occurs when two or more processes each hold a resource and mutually wait for another process to release some other resource. Consider an application that defines two `JoinBlock<T1,T2>` objects. Both objects each read data from two shared source blocks. In greedy mode, if one join block reads from the first source and the second join block reads from the second source, the application might deadlock because both join blocks mutually wait for the other to release its resource. In non-greedy mode, each join block reads from its sources only when all data is available, and therefore, the risk of deadlock is eliminated.

See also

- [Dataflow](#)

How to: Specify the Degree of Parallelism in a Dataflow Block

9/20/2022 • 5 minutes to read • [Edit Online](#)

This document describes how to set the `ExecutionDataflowBlockOptions.MaxDegreeOfParallelism` property to enable an execution dataflow block to process more than one message at a time. Doing this is useful when you have a dataflow block that performs a long-running computation and can benefit from processing messages in parallel. This example uses the `System.Threading.Tasks.Dataflow.ActionBlock<TInput>` class to perform multiple dataflow operations concurrently; however, you can specify the maximum degree of parallelism in any of the predefined execution block types that the TPL Dataflow Library provides, `ActionBlock<TInput>`, `System.Threading.Tasks.Dataflow.TransformBlock<TInput,TOutput>`, and `System.Threading.Tasks.Dataflow.TransformManyBlock<TInput,TOutput>`.

NOTE

The TPL Dataflow Library (the `System.Threading.Tasks.Dataflow` namespace) is not distributed with .NET. To install the `System.Threading.Tasks.Dataflow` namespace in Visual Studio, open your project, choose **Manage NuGet Packages** from the **Project** menu, and search online for the `System.Threading.Tasks.Dataflow` package. Alternatively, to install it using the [.NET Core CLI](#), run `dotnet add package System.Threading.Tasks.Dataflow`.

Example

The following example performs two dataflow computations and prints the elapsed time that is required for each computation. The first computation specifies a maximum degree of parallelism of 1, which is the default. A maximum degree of parallelism of 1 causes the dataflow block to process messages serially. The second computation resembles the first, except that it specifies a maximum degree of parallelism that is equal to the number of available processors. This enables the dataflow block to perform multiple operations in parallel.

```
using System;
using System.Diagnostics;
using System.Threading;
using System.Threading.Tasks.Dataflow;

// Demonstrates how to specify the maximum degree of parallelism
// when using dataflow.
class Program
{
    // Performs several computations by using dataflow and returns the elapsed
    // time required to perform the computations.
    static TimeSpan TimeDataflowComputations(int maxDegreeOfParallelism,
        int messageCount)
    {
        // Create an ActionBlock<int> that performs some work.
        var workerBlock = new ActionBlock<int>(
            // Simulate work by suspending the current thread.
            millisecondsTimeout => Thread.Sleep(millisecondsTimeout),
            // Specify a maximum degree of parallelism.
            new ExecutionDataflowBlockOptions
            {
                MaxDegreeOfParallelism = maxDegreeOfParallelism
            });
        // Compute the time that it takes for several messages to
        // flow through the dataflow block.
    }
}
```

```

// This sample code demonstrates how to specify the maximum degree of parallelism
// when using dataflow.

Stopwatch stopwatch = new Stopwatch();
stopwatch.Start();

for (int i = 0; i < messageCount; i++)
{
    workerBlock.Post(1000);
}
workerBlock.Complete();

// Wait for all messages to propagate through the network.
workerBlock.Completion.Wait();

// Stop the timer and return the elapsed number of milliseconds.
stopwatch.Stop();
return stopwatch.Elapsed;
}

static void Main(string[] args)
{
    int processorCount = Environment.ProcessorCount;
    int messageCount = processorCount;

    // Print the number of processors on this computer.
    Console.WriteLine("Processor count = {0}.", processorCount);

    TimeSpan elapsed;

    // Perform two dataflow computations and print the elapsed
    // time required for each.

    // This call specifies a maximum degree of parallelism of 1.
    // This causes the dataflow block to process messages serially.
    elapsed = TimeDataflowComputations(1, messageCount);
    Console.WriteLine("Degree of parallelism = {0}; message count = {1}; " +
        "elapsed time = {2}ms.", 1, messageCount, (int)elapsed.TotalMilliseconds);

    // Perform the computations again. This time, specify the number of
    // processors as the maximum degree of parallelism. This causes
    // multiple messages to be processed in parallel.
    elapsed = TimeDataflowComputations(processorCount, messageCount);
    Console.WriteLine("Degree of parallelism = {0}; message count = {1}; " +
        "elapsed time = {2}ms.", processorCount, messageCount, (int)elapsed.TotalMilliseconds);
}

/*
 * Sample output:
Processor count = 4.
Degree of parallelism = 1; message count = 4; elapsed time = 4032ms.
Degree of parallelism = 4; message count = 4; elapsed time = 1001ms.
*/

```

```

Imports System.Diagnostics
Imports System.Threading
Imports System.Threading.Tasks.Dataflow

' Demonstrates how to specify the maximum degree of parallelism
' when using dataflow.

Friend Class Program
    ' Performs several computations by using dataflow and returns the elapsed
    ' time required to perform the computations.
    Private Shared Function TimeDataflowComputations(ByVal maxDegreeOfParallelism As Integer, ByVal
messageCount As Integer) As TimeSpan
        ' Create an ActionBlock<int> that performs some work.
        Dim workerBlock = New ActionBlock(Of Integer)(Function(millisecondsTimeout)
Pause(millisecondsTimeout), New ExecutionDataflowBlockOptions() With {.MaxDegreeOfParallelism =
maxDegreeOfParallelism})
        ' Simulate work by suspending the current thread.

```

```

' Specify a maximum degree of parallelism.

' Compute the time that it takes for several messages to
' flow through the dataflow block.

Dim stopwatch As New Stopwatch()
stopwatch.Start()

For i As Integer = 0 To messageCount - 1
    workerBlock.Post(1000)
Next i
workerBlock.Complete()

' Wait for all messages to propagate through the network.
workerBlock.Completion.Wait()

' Stop the timer and return the elapsed number of milliseconds.
stopwatch.Stop()
Return stopwatch.Elapsed
End Function

Private Shared Function Pause(ByVal obj As Object)
    Thread.Sleep(obj)
    Return Nothing
End Function

Shared Sub Main(ByVal args() As String)
    Dim processorCount As Integer = Environment.ProcessorCount
    Dim messageCount As Integer = processorCount

    ' Print the number of processors on this computer.
    Console.WriteLine("Processor count = {0}.", processorCount)

    Dim elapsed As TimeSpan

    ' Perform two dataflow computations and print the elapsed
    ' time required for each.

    ' This call specifies a maximum degree of parallelism of 1.
    ' This causes the dataflow block to process messages serially.
    elapsed = TimeDataflowComputations(1, messageCount)
    Console.WriteLine("Degree of parallelism = {0}; message count = {1}; " & "elapsed time = {2}ms.", 1,
messageCount, CInt(Fix(elapsed.TotalMilliseconds)))

    ' Perform the computations again. This time, specify the number of
    ' processors as the maximum degree of parallelism. This causes
    ' multiple messages to be processed in parallel.
    elapsed = TimeDataflowComputations(processorCount, messageCount)
    Console.WriteLine("Degree of parallelism = {0}; message count = {1}; " & "elapsed time = {2}ms.",
processorCount, messageCount, CInt(Fix(elapsed.TotalMilliseconds)))
End Sub
End Class

' Sample output:
'Processor count = 4.
'Degree of parallelism = 1; message count = 4; elapsed time = 4032ms.
'Degree of parallelism = 4; message count = 4; elapsed time = 1001ms.
'

```

Robust Programming

By default, each predefined dataflow block propagates out messages in the order in which the messages are received. Although multiple messages are processed simultaneously when you specify a maximum degree of parallelism that is greater than 1, they are still propagated out in the order in which they are received.

Because the [MaxDegreeOfParallelism](#) property represents the maximum degree of parallelism, the dataflow

block might execute with a lesser degree of parallelism than you specify. The dataflow block can use a lesser degree of parallelism to meet its functional requirements or to account for a lack of available system resources. A dataflow block never chooses a greater degree of parallelism than you specify.

See also

- [Dataflow](#)

How to: Specify a Task Scheduler in a Dataflow Block

9/20/2022 • 11 minutes to read • [Edit Online](#)

This document demonstrates how to associate a specific task scheduler when you use dataflow in your application. The example uses the [System.Threading.Tasks.ConcurrentExclusiveSchedulerPair](#) class in a Windows Forms application to show when reader tasks are active and when a writer task is active. It also uses the [TaskScheduler.FromCurrentSynchronizationContext](#) method to enable a dataflow block to run on the user-interface thread.

NOTE

The TPL Dataflow Library (the [System.Threading.Tasks.Dataflow](#) namespace) is not distributed with .NET. To install the [System.Threading.Tasks.Dataflow](#) namespace in Visual Studio, open your project, choose **Manage NuGet Packages** from the **Project** menu, and search online for the [System.Threading.Tasks.Dataflow](#) package. Alternatively, to install it using the [.NET Core CLI](#), run `dotnet add package System.Threading.Tasks.Dataflow`.

To Create the Windows Forms Application

1. Create a Visual C# or Visual Basic Windows Forms Application project. In the following steps, the project is named `WriterReadersWinForms`.
2. On the form designer for the main form, Form1.cs (Form1.vb for Visual Basic), add four [CheckBox](#) controls. Set the [Text](#) property to **Reader 1** for `checkBox1`, **Reader 2** for `checkBox2`, **Reader 3** for `checkBox3`, and **Writer** for `checkBox4`. Set the [Enabled](#) property for each control to `False`.
3. Add a [Timer](#) control to the form. Set the [Interval](#) property to `2500`.

Adding Dataflow Functionality

This section describes how to create the dataflow blocks that participate in the application and how to associate each one with a task scheduler.

To Add Dataflow Functionality to the Application

1. In your project, add a reference to `System.Threading.Tasks.Dataflow.dll`.
2. Ensure that Form1.cs (Form1.vb for Visual Basic) contains the following `using` statements (`Imports` in Visual Basic).

```
using System;
using System.Linq;
using System.Threading;
using System.Threading.Tasks;
using System.Threading.Tasks.Dataflow;
using System.Windows.Forms;
```

```
Imports System.Threading
Imports System.Threading.Tasks
Imports System.Threading.Tasks.Dataflow
```

3. Add a `BroadcastBlock<T>` data member to the `Form1` class.

```
// Broadcasts values to an ActionBlock<int> object that is associated
// with each check box.
BroadcastBlock<int> broadcaster = new BroadcastBlock<int>(null);
```

```
' Broadcasts values to an ActionBlock<int> object that is associated
' with each check box.
Private broadcaster As New BroadcastBlock(Of Integer)(Nothing)
```

4. In the `Form1` constructor, after the call to `InitializeComponent`, create an `ActionBlock<TInput>` object that toggles the state of `CheckBox` objects.

```
// Create an ActionBlock<CheckBox> object that toggles the state
// of CheckBox objects.
// Specifying the current synchronization context enables the
// action to run on the user-interface thread.
var toggleCheckBox = new ActionBlock<CheckBox>(checkBox =>
{
    checkBox.Checked = !checkBox.Checked;
},
new ExecutionDataflowBlockOptions
{
    TaskScheduler = TaskScheduler.FromCurrentSynchronizationContext()
});
```

```
' Create an ActionBlock<CheckBox> object that toggles the state
' of CheckBox objects.
' Specifying the current synchronization context enables the
' action to run on the user-interface thread.
Dim toggleCheckBox = New ActionBlock(Of CheckBox)(Sub(checkBox) checkBox.Checked = Not
checkbox.Checked, New ExecutionDataflowBlockOptions With {.TaskScheduler =
TaskScheduler.FromCurrentSynchronizationContext()})
```

5. In the `Form1` constructor, create a `ConcurrentExclusiveSchedulerPair` object and four `ActionBlock<TInput>` objects, one `ActionBlock<TInput>` object for each `CheckBox` object. For each `ActionBlock<TInput>` object, specify an `ExecutionDataflowBlockOptions` object that has the `TaskScheduler` property set to the `ConcurrentScheduler` property for the readers, and the `ExclusiveScheduler` property for the writer.

```

// Create a ConcurrentExclusiveSchedulerPair object.
// Readers will run on the concurrent part of the scheduler pair.
// The writer will run on the exclusive part of the scheduler pair.
var taskSchedulerPair = new ConcurrentExclusiveSchedulerPair();

// Create an ActionBlock<int> object for each reader CheckBox object.
// Each ActionBlock<int> object represents an action that can read
// from a resource in parallel to other readers.
// Specifying the concurrent part of the scheduler pair enables the
// reader to run in parallel to other actions that are managed by
// that scheduler.
var readerActions =
    from checkBox in new CheckBox[] {checkBox1, checkBox2, checkBox3}
    select new ActionBlock<int>(milliseconds =>
{
    // Toggle the check box to the checked state.
    toggleCheckBox.Post(checkBox);

    // Perform the read action. For demonstration, suspend the current
    // thread to simulate a lengthy read operation.
    Thread.Sleep(milliseconds);

    // Toggle the check box to the unchecked state.
    toggleCheckBox.Post(checkBox);
},
new ExecutionDataflowBlockOptions
{
    TaskScheduler = taskSchedulerPair.ConcurrentScheduler
});

// Create an ActionBlock<int> object for the writer CheckBox object.
// This ActionBlock<int> object represents an action that writes to
// a resource, but cannot run in parallel to readers.
// Specifying the exclusive part of the scheduler pair enables the
// writer to run in exclusively with respect to other actions that are
// managed by the scheduler pair.
var writerAction = new ActionBlock<int>(milliseconds =>
{
    // Toggle the check box to the checked state.
    toggleCheckBox.Post(checkBox4);

    // Perform the write action. For demonstration, suspend the current
    // thread to simulate a lengthy write operation.
    Thread.Sleep(milliseconds);

    // Toggle the check box to the unchecked state.
    toggleCheckBox.Post(checkBox4);
},
new ExecutionDataflowBlockOptions
{
    TaskScheduler = taskSchedulerPair.ExclusiveScheduler
});

// Link the broadcaster to each reader and writer block.
// The BroadcastBlock<T> class propagates values that it
// receives to all connected targets.
foreach (var readerAction in readerActions)
{
    broadcaster.LinkTo(readerAction);
}
broadcaster.LinkTo(writerAction);

```

```

' Create a ConcurrentExclusiveSchedulerPair object.
' Readers will run on the concurrent part of the scheduler pair.
' The writer will run on the exclusive part of the scheduler pair.
Dim taskSchedulerPair = New ConcurrentExclusiveSchedulerPair()

' Create an ActionBlock<int> object for each reader CheckBox object.
' Each ActionBlock<int> object represents an action that can read
' from a resource in parallel to other readers.
' Specifying the concurrent part of the scheduler pair enables the
' reader to run in parallel to other actions that are managed by
' that scheduler.
Dim readerActions = From checkBox In New CheckBox() {checkBox1, checkBox2, checkBox3} _
    Select New ActionBlock(Of Integer)(Sub(milliseconds)
        ' Toggle the check box to the checked state.
        ' Perform the read action. For demonstration, suspend the
        current
        ' thread to simulate a lengthy read operation.
        ' Toggle the check box to the unchecked state.
        toggleCheckBox.Post(checkBox)
        Thread.Sleep(milliseconds)
        toggleCheckBox.Post(checkBox)
    End Sub, New ExecutionDataflowBlockOptions
With {.TaskScheduler = taskSchedulerPair.ConcurrentScheduler})

' Create an ActionBlock<int> object for the writer CheckBox object.
' This ActionBlock<int> object represents an action that writes to
' a resource, but cannot run in parallel to readers.
' Specifying the exclusive part of the scheduler pair enables the
' writer to run in exclusively with respect to other actions that are
' managed by the scheduler pair.
Dim writerAction = New ActionBlock(Of Integer)(Sub(milliseconds)
    ' Toggle the check box to the checked state.
    ' Perform the write action. For demonstration,
    suspend the current
    ' thread to simulate a lengthy write operation.
    ' Toggle the check box to the unchecked state.
    toggleCheckBox.Post(checkBox4)
    Thread.Sleep(milliseconds)
    toggleCheckBox.Post(checkBox4)
End Sub, New ExecutionDataflowBlockOptions With
{.TaskScheduler = taskSchedulerPair.ExclusiveScheduler})

' Link the broadcaster to each reader and writer block.
' The BroadcastBlock<T> class propagates values that it
' receives to all connected targets.
For Each readerAction In readerActions
    broadcaster.LinkTo(readerAction)
Next readerAction
broadcaster.LinkTo(writerAction)

```

6. In the `Form1` constructor, start the `Timer` object.

```
// Start the timer.
timer1.Start();
```

```
' Start the timer.
timer1.Start()
```

7. On the form designer for the main form, create an event handler for the `Tick` event for the timer.
8. Implement the `Tick` event for the timer.

```
// Event handler for the timer.
private void timer1_Tick(object sender, EventArgs e)
{
    // Post a value to the broadcaster. The broadcaster
    // sends this message to each target.
    broadcaster.Post(1000);
}
```

```
' Event handler for the timer.
Private Sub timer1_Tick(ByVal sender As Object, ByVal e As EventArgs) Handles timer1.Tick
    ' Post a value to the broadcaster. The broadcaster
    ' sends this message to each target.
    broadcaster.Post(1000)
End Sub
```

Because the `toggleCheckBox` dataflow block acts on the user interface, it is important that this action occur on the user-interface thread. To accomplish this, during construction this object provides an [ExecutionDataflowBlockOptions](#) object that has the [TaskScheduler](#) property set to [TaskScheduler.FromCurrentSynchronizationContext](#). The [FromCurrentSynchronizationContext](#) method creates a [TaskScheduler](#) object that performs work on the current synchronization context. Because the `Form1` constructor is called from the user-interface thread, the action for the `toggleCheckBox` dataflow block also runs on the user-interface thread.

This example also uses the [ConcurrentExclusiveSchedulerPair](#) class to enable some dataflow blocks to act concurrently, and another dataflow block to act exclusive with respect to all other dataflow blocks that run on the same [ConcurrentExclusiveSchedulerPair](#) object. This technique is useful when multiple dataflow blocks share a resource and some require exclusive access to that resource, because it eliminates the requirement to manually synchronize access to that resource. The elimination of manual synchronization can make code more efficient.

Example

The following example shows the complete code for Form1.cs (Form1.vb for Visual Basic).

```
using System;
using System.Linq;
using System.Threading;
using System.Threading.Tasks;
using System.Threading.Tasks.Dataflow;
using System.Windows.Forms;

namespace WriterReadersWinForms
{
    public partial class Form1 : Form
    {
        // Broadcasts values to an ActionBlock<int> object that is associated
        // with each check box.
        BroadcastBlock<int> broadcaster = new BroadcastBlock<int>(null);

        public Form1()
        {
            InitializeComponent();

            // Create an ActionBlock<CheckBox> object that toggles the state
            // of CheckBox objects.
            // Specifying the current synchronization context enables the
            // action to run on the user-interface thread.
            var toggleCheckBox = new ActionBlock<CheckBox>(checkBox =>
            {
```

```

    checkBox.Checked = !checkBox.Checked;
},
new ExecutionDataflowBlockOptions
{
    TaskScheduler = TaskScheduler.FromCurrentSynchronizationContext()
});

// Create a ConcurrentExclusiveSchedulerPair object.
// Readers will run on the concurrent part of the scheduler pair.
// The writer will run on the exclusive part of the scheduler pair.
var taskSchedulerPair = new ConcurrentExclusiveSchedulerPair();

// Create an ActionBlock<int> object for each reader CheckBox object.
// Each ActionBlock<int> object represents an action that can read
// from a resource in parallel to other readers.
// Specifying the concurrent part of the scheduler pair enables the
// reader to run in parallel to other actions that are managed by
// that scheduler.
var readerActions =
    from checkBox in new CheckBox[] {checkBox1, checkBox2, checkBox3}
    select new ActionBlock<int>(milliseconds =>
{
    // Toggle the check box to the checked state.
    toggleCheckBox.Post(checkBox);

    // Perform the read action. For demonstration, suspend the current
    // thread to simulate a lengthy read operation.
    Thread.Sleep(milliseconds);

    // Toggle the check box to the unchecked state.
    toggleCheckBox.Post(checkBox);
},
new ExecutionDataflowBlockOptions
{
    TaskScheduler = taskSchedulerPair.ConcurrentScheduler
});

// Create an ActionBlock<int> object for the writer CheckBox object.
// This ActionBlock<int> object represents an action that writes to
// a resource, but cannot run in parallel to readers.
// Specifying the exclusive part of the scheduler pair enables the
// writer to run in exclusively with respect to other actions that are
// managed by the scheduler pair.
var writerAction = new ActionBlock<int>(milliseconds =>
{
    // Toggle the check box to the checked state.
    toggleCheckBox.Post(checkBox4);

    // Perform the write action. For demonstration, suspend the current
    // thread to simulate a lengthy write operation.
    Thread.Sleep(milliseconds);

    // Toggle the check box to the unchecked state.
    toggleCheckBox.Post(checkBox4);
},
new ExecutionDataflowBlockOptions
{
    TaskScheduler = taskSchedulerPair.ExclusiveScheduler
});

// Link the broadcaster to each reader and writer block.
// The BroadcastBlock<T> class propagates values that it
// receives to all connected targets.
foreach (var readerAction in readerActions)
{
    broadcaster.LinkTo(readerAction);
}
broadcaster.LinkTo(writerAction);

```

```

        // Start the timer.
        timer1.Start();
    }

    // Event handler for the timer.
    private void timer1_Tick(object sender, EventArgs e)
    {
        // Post a value to the broadcaster. The broadcaster
        // sends this message to each target.
        broadcaster.Post(1000);
    }
}
}

```

```

Imports System.Threading
Imports System.Threading.Tasks
Imports System.Threading.Tasks.Dataflow

Namespace WriterReadersWinForms
    Partial Public Class Form1
        Inherits Form
        ' Broadcasts values to an ActionBlock<int> object that is associated
        ' with each check box.
        Private broadcaster As New BroadcastBlock(Of Integer)(Nothing)

        Public Sub New()
            InitializeComponent()

            ' Create an ActionBlock<CheckBox> object that toggles the state
            ' of CheckBox objects.
            ' Specifying the current synchronization context enables the
            ' action to run on the user-interface thread.
            Dim toggleCheckBox = New ActionBlock(Of CheckBox)(Sub(checkBox) checkBox.Checked = Not
checkbox.Checked, New ExecutionDataflowBlockOptions With {.TaskScheduler =
TaskScheduler.FromCurrentSynchronizationContext()})

            ' Create a ConcurrentExclusiveSchedulerPair object.
            ' Readers will run on the concurrent part of the scheduler pair.
            ' The writer will run on the exclusive part of the scheduler pair.
            Dim taskSchedulerPair = New ConcurrentExclusiveSchedulerPair()

            ' Create an ActionBlock<int> object for each reader CheckBox object.
            ' Each ActionBlock<int> object represents an action that can read
            ' from a resource in parallel to other readers.
            ' Specifying the concurrent part of the scheduler pair enables the
            ' reader to run in parallel to other actions that are managed by
            ' that scheduler.
            Dim readerActions = From checkBox In New CheckBox() {checkBox1, checkBox2, checkBox3} _
                Select New ActionBlock(Of Integer)(Sub(milliseconds)
                    ' Toggle the check box to the checked state.
                    ' Perform the read action. For demonstration, suspend
the current
                    ' thread to simulate a lengthy read operation.
                    ' Toggle the check box to the unchecked state.
                    toggleCheckBox.Post(checkBox)
                    Thread.Sleep(milliseconds)
                    toggleCheckBox.Post(checkBox)
                End Sub, New
            ExecutionDataflowBlockOptions With {.TaskScheduler = taskSchedulerPair.ConcurrentScheduler})

            ' Create an ActionBlock<int> object for the writer CheckBox object.
            ' This ActionBlock<int> object represents an action that writes to
            ' a resource, but cannot run in parallel to readers.
            ' Specifying the exclusive part of the scheduler pair enables the
            ' writer to run in exclusively with respect to other actions that are
            ' managed by the scheduler pair.

```

```

Dim writerAction = New ActionBlock(Of Integer)(Sub(milliseconds)
    ' Toggle the check box to the checked state.
    ' Perform the write action. For
demonstration, suspend the current
                                ' thread to simulate a lengthy write
operation.
                                ' Toggle the check box to the unchecked
state.

    toggleCheckBox.Post(checkBox4)
    Thread.Sleep(milliseconds)
    toggleCheckBox.Post(checkBox4)
End Sub, New ExecutionDataflowBlockOptions With
{.TaskScheduler = taskSchedulerPair.ExclusiveScheduler})

    ' Link the broadcaster to each reader and writer block.
    ' The BroadcastBlock<T> class propagates values that it
    ' receives to all connected targets.
For Each readerAction In readerActions
    broadcaster.LinkTo(readerAction)
Next readerAction
broadcaster.LinkTo(writerAction)

    ' Start the timer.
    timer1.Start()
End Sub

    ' Event handler for the timer.
Private Sub timer1_Tick(ByVal sender As Object, ByVal e As EventArgs) Handles timer1.Tick
    ' Post a value to the broadcaster. The broadcaster
    ' sends this message to each target.
    broadcaster.Post(1000)
End Sub
End Class
End Namespace

```

See also

- [Dataflow](#)

Walkthrough: Using BatchBlock and BatchedJoinBlock to Improve Efficiency

9/20/2022 • 29 minutes to read • [Edit Online](#)

The TPL Dataflow Library provides the `System.Threading.Tasks.Dataflow.BatchBlock<T>` and `System.Threading.Tasks.Dataflow.BatchedJoinBlock<T1,T2>` classes so that you can receive and buffer data from one or more sources and then propagate out that buffered data as one collection. This batching mechanism is useful when you collect data from one or more sources and then process multiple data elements as a batch. For example, consider an application that uses dataflow to insert records into a database. This operation can be more efficient if multiple items are inserted at the same time instead of one at a time sequentially. This document describes how to use the `BatchBlock<T>` class to improve the efficiency of such database insert operations. It also describes how to use the `BatchedJoinBlock<T1,T2>` class to capture both the results and any exceptions that occur when the program reads from a database.

NOTE

The TPL Dataflow Library (the `System.Threading.Tasks.Dataflow` namespace) is not distributed with .NET. To install the `System.Threading.Tasks.Dataflow` namespace in Visual Studio, open your project, choose **Manage NuGet Packages** from the **Project** menu, and search online for the `System.Threading.Tasks.Dataflow` package. Alternatively, to install it using the [.NET Core CLI](#), run `dotnet add package System.Threading.Tasks.Dataflow`.

Prerequisites

1. Read the Join Blocks section in the [Dataflow](#) document before you start this walkthrough.
2. Ensure that you have a copy of the Northwind database, `Northwind.sdf`, available on your computer. This file is typically located in the folder `%Program Files%\Microsoft SQL Server Compact Edition\v3.5\Samples\`.

IMPORTANT

In some versions of Windows, you cannot connect to `Northwind.sdf` if Visual Studio is running in a non-administrator mode. To connect to `Northwind.sdf`, start Visual Studio or a Developer Command Prompt for Visual Studio in the **Run as administrator** mode.

This walkthrough contains the following sections:

- [Creating the Console Application](#)
- [Defining the Employee Class](#)
- [Defining Employee Database Operations](#)
- [Adding Employee Data to the Database without Using Buffering](#)
- [Using Buffering to Add Employee Data to the Database](#)
- [Using Buffered Join to Read Employee Data from the Database](#)
- [The Complete Example](#)

Creating the Console Application

1. In Visual Studio, create a Visual C# or Visual Basic **Console Application** project. In this document, the project is named `DataflowBatchDatabase`.
2. In your project, add a reference to `System.Data.SqlServerCe.dll` and a reference to `System.Threading.Tasks.Dataflow.dll`.
3. Ensure that `Form1.cs` (`Form1.vb` for Visual Basic) contains the following `using` (`Imports` in Visual Basic) statements.

```
using System;
using System.Collections.Generic;
using System.Data.SqlServerCe;
using System.Diagnostics;
using System.IO;
using System.Threading.Tasks.Dataflow;
```

```
Imports System.Collections.Generic
Imports System.Data.SqlServerCe
Imports System.Diagnostics
Imports System.IO
Imports System.Threading.Tasks.Dataflow
```

4. Add the following data members to the `Program` class.

```
// The number of employees to add to the database.
// TODO: Change this value to experiment with different numbers of
// employees to insert into the database.
static readonly int insertCount = 256;

// The size of a single batch of employees to add to the database.
// TODO: Change this value to experiment with different batch sizes.
static readonly int insertBatchSize = 96;

// The source database file.
// TODO: Change this value if Northwind.sdf is at a different location
// on your computer.
static readonly string sourceDatabase =
    @"C:\Program Files\Microsoft SQL Server Compact Edition\v3.5\Samples\Northwind.sdf";

// TODO: Change this value if you require a different temporary location.
static readonly string scratchDatabase =
    @"C:\Temp\Northwind.sdf";
```

```

' The number of employees to add to the database.
' TODO: Change this value to experiment with different numbers of
' employees to insert into the database.
Private Shared ReadOnly insertCount As Integer = 256

' The size of a single batch of employees to add to the database.
' TODO: Change this value to experiment with different batch sizes.
Private Shared ReadOnly insertBatchSize As Integer = 96

' The source database file.
' TODO: Change this value if Northwind.sdf is at a different location
' on your computer.
Private Shared ReadOnly sourceDatabase As String = "C:\Program Files\Microsoft SQL Server Compact
Edition\v3.5\Samples\Northwind.sdf"

' TODO: Change this value if you require a different temporary location.
Private Shared ReadOnly scratchDatabase As String = "C:\Temp\Northwind.sdf"

```

Defining the Employee Class

Add to the `Program` class the `Employee` class.

```

// Describes an employee. Each property maps to a
// column in the Employees table in the Northwind database.
// For brevity, the Employee class does not contain
// all columns from the Employees table.
class Employee
{
    public int EmployeeID { get; set; }
    public string LastName { get; set; }
    public string FirstName { get; set; }

    // A random number generator that helps tp generate
    // Employee property values.
    static Random rand = new Random(42);

    // Possible random first names.
    static readonly string[] firstNames = { "Tom", "Mike", "Ruth", "Bob", "John" };
    // Possible random last names.
    static readonly string[] lastNames = { "Jones", "Smith", "Johnson", "Walker" };

    // Creates an Employee object that contains random
    // property values.
    public static Employee Random()
    {
        return new Employee
        {
            EmployeeID = -1,
            LastName = lastNames[rand.Next() % lastNames.Length],
            FirstName = firstNames[rand.Next() % firstNames.Length]
        };
    }
}

```

```

' Describes an employee. Each property maps to a
' column in the Employees table in the Northwind database.
' For brevity, the Employee class does not contain
' all columns from the Employees table.
Private Class Employee
    Public Property EmployeeID() As Integer
    Public Property LastName() As String
    Public Property FirstName() As String

    ' A random number generator that helps tp generate
    ' Employee property values.
    Private Shared rand As New Random(42)

    ' Possible random first names.
    Private Shared ReadOnly firstNames() As String = {"Tom", "Mike", "Ruth", "Bob", "John"}
    ' Possible random last names.
    Private Shared ReadOnly lastNames() As String = {"Jones", "Smith", "Johnson", "Walker"}

    ' Creates an Employee object that contains random
    ' property values.
    Public Shared Function Random() As Employee
        Return New Employee With {.EmployeeID = -1, .LastName = lastNames(rand.Next() Mod lastNames.Length),
        .FirstName = firstNames(rand.Next() Mod firstNames.Length)}
    End Function
End Class

```

The `Employee` class contains three properties, `EmployeeID`, `LastName`, and `FirstName`. These properties correspond to the `Employee ID`, `Last Name`, and `First Name` columns in the `Employees` table in the Northwind database. For this demonstration, the `Employee` class also defines the `Random` method, which creates an `Employee` object that has random values for its properties.

Defining Employee Database Operations

Add to the `Program` class the `InsertEmployees`, `GetEmployeeCount`, and `GetEmployeeID` methods.

```

// Adds new employee records to the database.
static void InsertEmployees(Employee[] employees, string connectionString)
{
    using (SqlCeConnection connection =
        new SqlCeConnection(connectionString))
    {
        try
        {
            // Create the SQL command.
            SqlCeCommand command = new SqlCeCommand(
                "INSERT INTO Employees ([Last Name], [First Name])" +
                "VALUES (@lastName, @firstName)",
                connection);

            connection.Open();
            for (int i = 0; i < employees.Length; i++)
            {
                // Set parameters.
                command.Parameters.Clear();
                command.Parameters.Add("@lastName", employees[i].LastName);
                command.Parameters.Add("@firstName", employees[i].FirstName);

                // Execute the command.
                command.ExecuteNonQuery();
            }
        }
        finally
        {

```

```

        connection.Close();
    }
}

// Retrieves the number of entries in the Employees table in
// the Northwind database.
static int GetEmployeeCount(string connectionString)
{
    int result = 0;
    using (SqlCeConnection sqlConnection =
        new SqlCeConnection(connectionString))
    {
        SqlCeCommand sqlCommand = new SqlCeCommand(
            "SELECT COUNT(*) FROM Employees", sqlConnection);

        sqlConnection.Open();
        try
        {
            result = (int)sqlCommand.ExecuteScalar();
        }
        finally
        {
            sqlConnection.Close();
        }
    }
    return result;
}

// Retrieves the ID of the first employee that has the provided name.
static int GetEmployeeID(string lastName, string firstName,
    string connectionString)
{
    using (SqlCeConnection connection =
        new SqlCeConnection(connectionString))
    {
        SqlCeCommand command = new SqlCeCommand(
            string.Format(
                "SELECT [Employee ID] FROM Employees " +
                "WHERE [Last Name] = '{0}' AND [First Name] = '{1}'",
                lastName, firstName),
            connection);

        connection.Open();
        try
        {
            return (int)command.ExecuteScalar();
        }
        finally
        {
            connection.Close();
        }
    }
}

```

```

' Adds new employee records to the database.
Private Shared Sub InsertEmployees(ByVal employees() As Employee, ByVal connectionString As String)
    Using connection As New SqlCeConnection(connectionString)
        Try
            ' Create the SQL command.
            Dim command As New SqlCeCommand("INSERT INTO Employees ([Last Name], [First Name])" & "VALUES (@lastName, @firstName)", connection)

            connection.Open()
            For i As Integer = 0 To employees.Length - 1
                ' Set parameters.
                command.Parameters.Clear()
                command.Parameters.Add("@lastName", employees(i).LastName)
                command.Parameters.Add("@firstName", employees(i).FirstName)

                ' Execute the command.
                command.ExecuteNonQuery()
            Next i
        Finally
            connection.Close()
        End Try
    End Using
End Sub

' Retrieves the number of entries in the Employees table in
' the Northwind database.
Private Shared Function GetEmployeeCount(ByVal connectionString As String) As Integer
    Dim result As Integer = 0
    Using sqlConnection As New SqlCeConnection(connectionString)
        Dim sqlCommand As New SqlCeCommand("SELECT COUNT(*) FROM Employees", sqlConnection)

        sqlConnection.Open()
        Try
            result = CInt(Fix(sqlCommand.ExecuteScalar()))
        Finally
            sqlConnection.Close()
        End Try
    End Using
    Return result
End Function

' Retrieves the ID of the first employee that has the provided name.
Private Shared Function GetEmployeeID(ByVal lastName As String, ByVal firstName As String, ByVal
connectionString As String) As Integer
    Using connection As New SqlCeConnection(connectionString)
        Dim command As New SqlCeCommand(String.Format("SELECT [Employee ID] FROM Employees " & "WHERE [Last
Name] = '{0}' AND [First Name] = '{1}'", lastName, firstName), connection)

        connection.Open()
        Try
            Return CInt(Fix(command.ExecuteScalar()))
        Finally
            connection.Close()
        End Try
    End Using
End Function

```

The `InsertEmployees` method adds new employee records to the database. The `GetEmployeeCount` method retrieves the number of entries in the `Employees` table. The `GetEmployeeID` method retrieves the identifier of the first employee that has the provided name. Each of these methods takes a connection string to the Northwind database and uses functionality in the `System.Data.SqlServerCe` namespace to communicate with the database.

Adding Employee Data to the Database Without Using Buffering

Add to the `Program` class the `AddEmployees` and `PostRandomEmployees` methods.

```
// Posts random Employee data to the provided target block.
static void PostRandomEmployees(ITargetBlock<Employee> target, int count)
{
    Console.WriteLine("Adding {0} entries to Employee table...", count);

    for (int i = 0; i < count; i++)
    {
        target.Post(Employee.Random());
    }
}

// Adds random employee data to the database by using dataflow.
static void AddEmployees(string connectionString, int count)
{
    // Create an ActionBlock<Employee> object that adds a single
    // employee entry to the database.
    var insertEmployee = new ActionBlock<Employee>(e =>
        InsertEmployees(new Employee[] { e }, connectionString));

    // Post several random Employee objects to the dataflow block.
    PostRandomEmployees(insertEmployee, count);

    // Set the dataflow block to the completed state and wait for
    // all insert operations to complete.
    insertEmployee.Complete();
    insertEmployee.Completion.Wait();
}
```

```
' Posts random Employee data to the provided target block.
Private Shared Sub PostRandomEmployees(ByVal target As ITargetBlock(Of Employee), ByVal count As Integer)
    Console.WriteLine("Adding {0} entries to Employee table...", count)

    For i As Integer = 0 To count - 1
        target.Post(Employee.Random())
    Next i
End Sub

' Adds random employee data to the database by using dataflow.
Private Shared Sub AddEmployees(ByVal connectionString As String, ByVal count As Integer)
    ' Create an ActionBlock<Employee> object that adds a single
    ' employee entry to the database.
    Dim insertEmployee = New ActionBlock(Of Employee)(Sub(e) InsertEmployees(New Employee() {e},
connectionString))

    ' Post several random Employee objects to the dataflow block.
    PostRandomEmployees(insertEmployee, count)

    ' Set the dataflow block to the completed state and wait for
    ' all insert operations to complete.
    insertEmployee.Complete()
    insertEmployee.Completion.Wait()
End Sub
```

The `AddEmployees` method adds random employee data to the database by using dataflow. It creates an `ActionBlock<TInput>` object that calls the `InsertEmployees` method to add an employee entry to the database. The `AddEmployees` method then calls the `PostRandomEmployees` method to post multiple `Employee` objects to the `ActionBlock<TInput>` object. The `AddEmployees` method then waits for all insert operations to finish.

Using Buffering to Add Employee Data to the Database

Add to the `Program` class the `AddEmployeesBatched` method.

```
// Adds random employee data to the database by using dataflow.  
// This method is similar to AddEmployees except that it uses batching  
// to add multiple employees to the database at a time.  
static void AddEmployeesBatched(string connectionString, int batchSize,  
    int count)  
{  
    // Create a BatchBlock<Employee> that holds several Employee objects and  
    // then propagates them out as an array.  
    var batchEmployees = new BatchBlock<Employee>(batchSize);  
  
    // Create an ActionBlock<Employee[]> object that adds multiple  
    // employee entries to the database.  
    var insertEmployees = new ActionBlock<Employee[]>(a =>  
        InsertEmployees(a, connectionString));  
  
    // Link the batch block to the action block.  
    batchEmployees.LinkTo(insertEmployees);  
  
    // When the batch block completes, set the action block also to complete.  
    batchEmployees.Completion.ContinueWith(delegate { insertEmployees.Complete(); });  
  
    // Post several random Employee objects to the batch block.  
    PostRandomEmployees(batchEmployees, count);  
  
    // Set the batch block to the completed state and wait for  
    // all insert operations to complete.  
    batchEmployees.Complete();  
    insertEmployees.Completion.Wait();  
}
```

```
' Adds random employee data to the database by using dataflow.  
' This method is similar to AddEmployees except that it uses batching  
' to add multiple employees to the database at a time.  
Private Shared Sub AddEmployeesBatched(ByVal connectionString As String, ByVal batchSize As Integer, ByVal  
count As Integer)  
    ' Create a BatchBlock<Employee> that holds several Employee objects and  
    ' then propagates them out as an array.  
    Dim batchEmployees = New BatchBlock(Of Employee)(batchSize)  
  
    ' Create an ActionBlock<Employee[]> object that adds multiple  
    ' employee entries to the database.  
    Dim insertEmployees = New ActionBlock(Of Employee())(Sub(a) Program.InsertEmployees(a,  
connectionString))  
  
    ' Link the batch block to the action block.  
    batchEmployees.LinkTo(insertEmployees)  
  
    ' When the batch block completes, set the action block also to complete.  
    batchEmployees.Completion.ContinueWith(Sub() insertEmployees.Complete())  
  
    ' Post several random Employee objects to the batch block.  
    PostRandomEmployees(batchEmployees, count)  
  
    ' Set the batch block to the completed state and wait for  
    ' all insert operations to complete.  
    batchEmployees.Complete()  
    insertEmployees.Completion.Wait()  
End Sub
```

This method resembles `AddEmployees`, except that it also uses the `BatchBlock<T>` class to buffer multiple `Employee` objects before it sends those objects to the `ActionBlock<TInput>` object. Because the `BatchBlock<T>` class propagates out multiple elements as a collection, the `ActionBlock<TInput>` object is modified to act on an

array of `Employee` objects. As in the `AddEmployees` method, `AddEmployeesBatched` calls the `PostRandomEmployees` method to post multiple `Employee` objects; however, `AddEmployeesBatched` posts these objects to the `BatchBlock<T>` object. The `AddEmployeesBatched` method also waits for all insert operations to finish.

Using Buffered Join to Read Employee Data from the Database

Add to the `Program` class the `GetRandomEmployees` method.

```
// Displays information about several random employees to the console.
static void GetRandomEmployees(string connectionString, int batchSize,
    int count)
{
    // Create a BatchedJoinBlock<Employee, Exception> object that holds
    // both employee and exception data.
    var selectEmployees = new BatchedJoinBlock<Employee, Exception>(batchSize);

    // Holds the total number of exceptions that occurred.
    int totalErrors = 0;

    // Create an action block that prints employee and error information
    // to the console.
    var printEmployees =
        new ActionBlock<Tuple<IList<Employee>, IList<Exception>>>(data =>
    {
        // Print information about the employees in this batch.
        Console.WriteLine("Received a batch...");
        foreach (Employee e in data.Item1)
        {
            Console.WriteLine("Last={0} First={1} ID={2}",
                e.LastName, e.FirstName, e.EmployeeID);
        }

        // Print the error count for this batch.
        Console.WriteLine("There were {0} errors in this batch...",
            data.Item2.Count);

        // Update total error count.
        totalErrors += data.Item2.Count;
    });

    // Link the batched join block to the action block.
    selectEmployees.LinkTo(printEmployees);

    // When the batched join block completes, set the action block also to complete.
    selectEmployees.Completion.ContinueWith(delegate { printEmployees.Complete(); });

    // Try to retrieve the ID for several random employees.
    Console.WriteLine("Selecting random entries from Employees table...");
    for (int i = 0; i < count; i++)
    {
        try
        {
            // Create a random employee.
            Employee e = Employee.Random();

            // Try to retrieve the ID for the employee from the database.
            e.EmployeeID = GetEmployeeID(e.LastName, e.FirstName, connectionString);

            // Post the Employee object to the Employee target of
            // the batched join block.
            selectEmployees.Target1.Post(e);
        }
        catch (NullReferenceException e)
        {
            // GetEmployeeID throws NullReferenceException when there is
```

```

        // no such employee with the given name. When this happens,
        // post the Exception object to the Exception target of
        // the batched join block.
        selectEmployees.Target2.Post(e);
    }

}

// Set the batched join block to the completed state and wait for
// all retrieval operations to complete.
selectEmployees.Complete();
printEmployees.Completion.Wait();

// Print the total error count.
Console.WriteLine("Finished. There were {0} total errors.", totalErrors);
}

```

```

' Displays information about several random employees to the console.
Private Shared Sub GetRandomEmployees(ByVal connectionString As String, ByVal batchSize As Integer, ByVal
count As Integer)
    ' Create a BatchedJoinBlock<Employee, Exception> object that holds
    ' both employee and exception data.
    Dim selectEmployees = New BatchedJoinBlock(Of Employee, Exception)(batchSize)

    ' Holds the total number of exceptions that occurred.
    Dim totalErrors As Integer = 0

    ' Create an action block that prints employee and error information
    ' to the console.
    Dim printEmployees = New ActionBlock(Of Tuple(Of IList(Of Employee), IList(Of Exception)))(Sub(data)
                                                ' Print
information about the employees in this batch.
                                                ' Print
the error count for this batch.
                                                ' Update
total error count.

    Console.WriteLine("Received a batch...")
    For Each
e As Employee In data.Item1

    Console.WriteLine("Last={0} First={1} ID={2}", e.LastName, e.FirstName, e.EmployeeID)
    Next e

    Console.WriteLine("There were {0} errors in this batch...", data.Item2.Count)

    totalErrors += data.Item2.Count
    End Sub)

    ' Link the batched join block to the action block.
    selectEmployees.LinkTo(printEmployees)

    ' When the batched join block completes, set the action block also to complete.
    selectEmployees.Completion.ContinueWith(Sub() printEmployees.Complete())

    ' Try to retrieve the ID for several random employees.
    Console.WriteLine("Selecting random entries from Employees table...")
    For i As Integer = 0 To count - 1
        Try
            ' Create a random employee.
            Dim e As Employee = Employee.Random()

            ' Try to retrieve the ID for the employee from the database.
            e.EmployeeID = GetEmployeeID(e.LastName, e.FirstName, connectionString)

            ' Post the Employee object to the Employee target of
            ' the batched join block.
            selectEmployees.Target1.Post(e)
        Catch ex As Exception
            ' Post the exception to the exception target of the batched join block.
            selectEmployees.Target2.Post(ex)
        End Try
    Next i

```

```

    Catch e As NullReferenceException
        ' GetEmployeeID throws NullReferenceException when there is
        ' no such employee with the given name. When this happens,
        ' post the Exception object to the Exception target of
        ' the batched join block.
        selectEmployees.Target2.Post(e)
    End Try
    Next i

    ' Set the batched join block to the completed state and wait for
    ' all retrieval operations to complete.
    selectEmployees.Complete()
    printEmployees.Completion.Wait()

    ' Print the total error count.
    Console.WriteLine("Finished. There were {0} total errors.", totalErrors)
End Sub

```

This method prints information about random employees to the console. It creates several random `Employee` objects and calls the `GetEmployeeID` method to retrieve the unique identifier for each object. Because the `GetEmployeeID` method throws an exception if there is no matching employee with the given first and last names, the `GetRandomEmployees` method uses the `BatchedJoinBlock<T1,T2>` class to store `Employee` objects for successful calls to `GetEmployeeID` and `System.Exception` objects for calls that fail. The `ActionBlock<TInput>` object in this example acts on a `Tuple<T1,T2>` object that holds a list of `Employee` objects and a list of `Exception` objects. The `BatchedJoinBlock<T1,T2>` object propagates out this data when the sum of the received `Employee` and `Exception` object counts equals the batch size.

The Complete Example

The following example shows the complete code. The `Main` method compares the time that is required to perform batched database insertions versus the time to perform non-batched database insertions. It also demonstrates the use of buffered join to read employee data from the database and also report errors.

```

using System;
using System.Collections.Generic;
using System.Data.SqlServerCe;
using System.Diagnostics;
using System.IO;
using System.Threading.Tasks.Dataflow;

// Demonstrates how to use batched dataflow blocks to improve
// the performance of database operations.
namespace DataflowBatchDatabase
{
    class Program
    {
        // The number of employees to add to the database.
        // TODO: Change this value to experiment with different numbers of
        // employees to insert into the database.
        static readonly int insertCount = 256;

        // The size of a single batch of employees to add to the database.
        // TODO: Change this value to experiment with different batch sizes.
        static readonly int insertBatchSize = 96;

        // The source database file.
        // TODO: Change this value if Northwind.sdf is at a different location
        // on your computer.
        static readonly string sourceDatabase =
            @"C:\Program Files\Microsoft SQL Server Compact Edition\v3.5\Samples\Northwind.sdf";

        // TODO: Change this value if you require a different temporary location.
    }
}

```

```

static readonly string scratchDatabase =
    @"C:\Temp\Northwind.sdf";

// Describes an employee. Each property maps to a
// column in the Employees table in the Northwind database.
// For brevity, the Employee class does not contain
// all columns from the Employees table.
class Employee
{
    public int EmployeeID { get; set; }
    public string LastName { get; set; }
    public string FirstName { get; set; }

    // A random number generator that helps to generate
    // Employee property values.
    static Random rand = new Random(42);

    // Possible random first names.
    static readonly string[] firstNames = { "Tom", "Mike", "Ruth", "Bob", "John" };
    // Possible random last names.
    static readonly string[] lastNames = { "Jones", "Smith", "Johnson", "Walker" };

    // Creates an Employee object that contains random
    // property values.
    public static Employee Random()
    {
        return new Employee
        {
            EmployeeID = -1,
            LastName = lastNames[rand.Next() % lastNames.Length],
            FirstName = firstNames[rand.Next() % firstNames.Length]
        };
    }
}

// Adds new employee records to the database.
static void InsertEmployees(Employee[] employees, string connectionString)
{
    using (SqlCeConnection connection =
        new SqlCeConnection(connectionString))
    {
        try
        {
            // Create the SQL command.
            SqlCeCommand command = new SqlCeCommand(
                "INSERT INTO Employees ([Last Name], [First Name])" +
                "VALUES (@lastName, @firstName)",
                connection);

            connection.Open();
            for (int i = 0; i < employees.Length; i++)
            {
                // Set parameters.
                command.Parameters.Clear();
                command.Parameters.Add("@lastName", employees[i].LastName);
                command.Parameters.Add("@firstName", employees[i].FirstName);

                // Execute the command.
                command.ExecuteNonQuery();
            }
        }
        finally
        {
            connection.Close();
        }
    }
}

// Retrieves the number of entries in the Employees table in

```

```

// the Northwind database.
static int GetEmployeeCount(string connectionString)
{
    int result = 0;
    using (SqlCeConnection sqlConnection =
        new SqlCeConnection(connectionString))
    {
        SqlCeCommand sqlCommand = new SqlCeCommand(
            "SELECT COUNT(*) FROM Employees", sqlConnection);

        sqlConnection.Open();
        try
        {
            result = (int)sqlCommand.ExecuteScalar();
        }
        finally
        {
            sqlConnection.Close();
        }
    }
    return result;
}

// Retrieves the ID of the first employee that has the provided name.
static int GetEmployeeID(string lastName, string firstName,
    string connectionString)
{
    using (SqlCeConnection connection =
        new SqlCeConnection(connectionString))
    {
        SqlCeCommand command = new SqlCeCommand(
            string.Format(
                "SELECT [Employee ID] FROM Employees " +
                "WHERE [Last Name] = '{0}' AND [First Name] = '{1}'",
                lastName, firstName),
            connection);

        connection.Open();
        try
        {
            return (int)command.ExecuteScalar();
        }
        finally
        {
            connection.Close();
        }
    }
}

// Posts random Employee data to the provided target block.
static void PostRandomEmployees(ITargetBlock<Employee> target, int count)
{
    Console.WriteLine("Adding {0} entries to Employee table...", count);

    for (int i = 0; i < count; i++)
    {
        target.Post(Employee.Random());
    }
}

// Adds random employee data to the database by using dataflow.
static void AddEmployees(string connectionString, int count)
{
    // Create an ActionBlock<Employee> object that adds a single
    // employee entry to the database.
    var insertEmployee = new ActionBlock<Employee>(e =>
        InsertEmployees(new Employee[] { e }, connectionString));

    // Post several random Employee objects to the dataflow block.
}

```

```

PostRandomEmployees(insertEmployee, count);

    // Set the dataflow block to the completed state and wait for
    // all insert operations to complete.
    insertEmployee.Complete();
    insertEmployee.Completion.Wait();
}

// Adds random employee data to the database by using dataflow.
// This method is similar to AddEmployees except that it uses batching
// to add multiple employees to the database at a time.
static void AddEmployeesBatched(string connectionString, int batchSize,
    int count)
{
    // Create a BatchBlock<Employee> that holds several Employee objects and
    // then propagates them out as an array.
    var batchEmployees = new BatchBlock<Employee>(batchSize);

    // Create an ActionBlock<Employee[]> object that adds multiple
    // employee entries to the database.
    var insertEmployees = new ActionBlock<Employee[]>(a =>
        InsertEmployees(a, connectionString));

    // Link the batch block to the action block.
    batchEmployees.LinkTo(insertEmployees);

    // When the batch block completes, set the action block also to complete.
    batchEmployees.Completion.ContinueWith(delegate { insertEmployees.Complete(); });

    // Post several random Employee objects to the batch block.
    PostRandomEmployees(batchEmployees, count);

    // Set the batch block to the completed state and wait for
    // all insert operations to complete.
    batchEmployees.Complete();
    insertEmployees.Completion.Wait();
}

// Displays information about several random employees to the console.
static void GetRandomEmployees(string connectionString, int batchSize,
    int count)
{
    // Create a BatchedJoinBlock<Employee, Exception> object that holds
    // both employee and exception data.
    var selectEmployees = new BatchedJoinBlock<Employee, Exception>(batchSize);

    // Holds the total number of exceptions that occurred.
    int totalErrors = 0;

    // Create an action block that prints employee and error information
    // to the console.
    var printEmployees =
        new ActionBlock<Tuple<IList<Employee>, IList<Exception>>>(data =>
    {
        // Print information about the employees in this batch.
        Console.WriteLine("Received a batch...");
        foreach (Employee e in data.Item1)
        {
            Console.WriteLine("Last={0} First={1} ID={2}",
                e.LastName, e.FirstName, e.EmployeeID);
        }

        // Print the error count for this batch.
        Console.WriteLine("There were {0} errors in this batch...",
            data.Item2.Count);

        // Update total error count.
        totalErrors += data.Item2.Count;
    });
}

```

```

// Link the batched join block to the action block.
selectEmployees.LinkTo(printEmployees);

// When the batched join block completes, set the action block also to complete.
selectEmployees.Completion.ContinueWith(delegate { printEmployees.Complete(); });

// Try to retrieve the ID for several random employees.
Console.WriteLine("Selecting random entries from Employees table...");
for (int i = 0; i < count; i++)
{
    try
    {
        // Create a random employee.
        Employee e = Employee.Random();

        // Try to retrieve the ID for the employee from the database.
        e.EmployeeID = GetEmployeeID(e.LastName, e.FirstName, connectionString);

        // Post the Employee object to the Employee target of
        // the batched join block.
        selectEmployees.Target1.Post(e);
    }
    catch (NullReferenceException e)
    {
        // GetEmployeeID throws NullReferenceException when there is
        // no such employee with the given name. When this happens,
        // post the Exception object to the Exception target of
        // the batched join block.
        selectEmployees.Target2.Post(e);
    }
}

// Set the batched join block to the completed state and wait for
// all retrieval operations to complete.
selectEmployees.Complete();
printEmployees.Completion.Wait();

// Print the total error count.
Console.WriteLine("Finished. There were {0} total errors.", totalErrors);
}

static void Main(string[] args)
{
    // Create a connection string for accessing the database.
    // The connection string refers to the temporary database location.
    string connectionString = string.Format(@"Data Source={0}",
        scratchDatabase);

    // Create a Stopwatch object to time database insert operations.
    Stopwatch stopwatch = new Stopwatch();

    // Start with a clean database file by copying the source database to
    // the temporary location.
    File.Copy(sourceDatabase, scratchDatabase, true);

    // Demonstrate multiple insert operations without batching.
    Console.WriteLine("Demonstrating non-batched database insert operations...");
    Console.WriteLine("Original size of Employee table: {0}.",
        GetEmployeeCount(connectionString));
    stopwatch.Start();
    AddEmployees(connectionString, insertCount);
    stopwatch.Stop();
    Console.WriteLine("New size of Employee table: {0}; elapsed insert time: {1} ms.",
        GetEmployeeCount(connectionString), stopwatch.ElapsedMilliseconds);

    Console.WriteLine();

    // Start again with a clean database file.
}

```

```

        File.Copy(sourceDatabase, scratchDatabase, true);

        // Demonstrate multiple insert operations, this time with batching.
        Console.WriteLine("Demonstrating batched database insert operations...");
        Console.WriteLine("Original size of Employee table: {0}.",
            GetEmployeeCount(connectionString));
        stopwatch.Restart();
        AddEmployeesBatched(connectionString, insertBatchSize, insertCount);
        stopwatch.Stop();
        Console.WriteLine("New size of Employee table: {0}; elapsed insert time: {1} ms.",
            GetEmployeeCount(connectionString), stopwatch.ElapsedMilliseconds);

        Console.WriteLine();

        // Start again with a clean database file.
        File.Copy(sourceDatabase, scratchDatabase, true);

        // Demonstrate multiple retrieval operations with error reporting.
        Console.WriteLine("Demonstrating batched join database select operations...");
        // Add a small number of employees to the database.
        AddEmployeesBatched(connectionString, insertBatchSize, 16);
        // Query for random employees.
        GetRandomEmployees(connectionString, insertBatchSize, 10);
    }
}
/* Sample output:
Demonstrating non-batched database insert operations...
Original size of Employee table: 15.
Adding 256 entries to Employee table...
New size of Employee table: 271; elapsed insert time: 11035 ms.

Demonstrating batched database insert operations...
Original size of Employee table: 15.
Adding 256 entries to Employee table...
New size of Employee table: 271; elapsed insert time: 197 ms.

Demonstrating batched join database insert operations...
Adding 16 entries to Employee table...
Selecting items from Employee table...
Received a batch...
Last=Jones First=Tom ID=21
Last=Jones First=John ID=24
Last=Smith First=Tom ID=26
Last=Jones First=Tom ID=21
There were 4 errors in this batch...
Received a batch...
Last=Smith First=Tom ID=26
Last=Jones First=Mike ID=28
There were 0 errors in this batch...
Finished. There were 4 total errors.
*/

```

```

Imports System.Collections.Generic
Imports System.Data.SqlServerCe
Imports System.Diagnostics
Imports System.IO
Imports System.Threading.Tasks.Dataflow

' Demonstrates how to use batched dataflow blocks to improve
' the performance of database operations.
Namespace DataflowBatchDatabase
    Friend Class Program
        ' The number of employees to add to the database.
        ' TODO: Change this value to experiment with different numbers of
        ' employees to insert into the database.

```

```

Private Shared ReadOnly insertCount As Integer = 256

' The size of a single batch of employees to add to the database.
' TODO: Change this value to experiment with different batch sizes.
Private Shared ReadOnly insertBatchSize As Integer = 96

' The source database file.
' TODO: Change this value if Northwind.sdf is at a different location
' on your computer.
Private Shared ReadOnly sourceDatabase As String = "C:\Program Files\Microsoft SQL Server Compact
Edition\v3.5\Samples\Northwind.sdf"

' TODO: Change this value if you require a different temporary location.
Private Shared ReadOnly scratchDatabase As String = "C:\Temp\Northwind.sdf"

' Describes an employee. Each property maps to a
' column in the Employees table in the Northwind database.
' For brevity, the Employee class does not contain
' all columns from the Employees table.
Private Class Employee
    Public Property EmployeeID() As Integer
    Public Property LastName() As String
    Public Property FirstName() As String

    ' A random number generator that helps tp generate
    ' Employee property values.
    Private Shared rand As New Random(42)

    ' Possible random first names.
    Private Shared ReadOnly firstNames() As String = {"Tom", "Mike", "Ruth", "Bob", "John"}
    ' Possible random last names.
    Private Shared ReadOnly lastNames() As String = {"Jones", "Smith", "Johnson", "Walker"}

    ' Creates an Employee object that contains random
    ' property values.
    Public Shared Function Random() As Employee
        Return New Employee With {.EmployeeID = -1, .LastName = lastNames(rand.Next() Mod
lastNames.Length), .FirstName = firstNames(rand.Next() Mod firstNames.Length)}
    End Function
End Class

' Adds new employee records to the database.
Private Shared Sub InsertEmployees(ByVal employees() As Employee, ByVal connectionString As String)
    Using connection As New SqlCeConnection(connectionString)
        Try
            ' Create the SQL command.
            Dim command As New SqlCeCommand("INSERT INTO Employees ([Last Name], [First Name])" &
"VALUES (@lastName, @firstName)", connection)

            connection.Open()
            For i As Integer = 0 To employees.Length - 1
                ' Set parameters.
                command.Parameters.Clear()
                command.Parameters.Add("@lastName", employees(i).LastName)
                command.Parameters.Add("@firstName", employees(i).FirstName)

                ' Execute the command.
                command.ExecuteNonQuery()
            Next i
        Finally
            connection.Close()
        End Try
    End Using
End Sub

' Retrieves the number of entries in the Employees table in
' the Northwind database.
Private Shared Function GetEmployeeCount(ByVal connectionString As String) As Integer
    Dim result As Integer = 0

```

```

        Using sqlConnection As New SqlCeConnection(connectionString)
            Dim sqlCommand As New SqlCeCommand("SELECT COUNT(*) FROM Employees", sqlConnection)

            sqlConnection.Open()
            Try
                result = CInt(Fix(sqlCommand.ExecuteScalar()))
            Finally
                sqlConnection.Close()
            End Try
        End Using
        Return result
    End Function

    ' Retrieves the ID of the first employee that has the provided name.
    Private Shared Function GetEmployeeID(ByVal lastName As String, ByVal firstName As String, ByVal
connectionString As String) As Integer
        Using connection As New SqlCeConnection(connectionString)
            Dim command As New SqlCeCommand(String.Format("SELECT [Employee ID] FROM Employees " &
"WHERE [Last Name] = '{0}' AND [First Name] = '{1}'", lastName, firstName), connection)

            connection.Open()
            Try
                Return CInt(Fix(command.ExecuteScalar()))
            Finally
                connection.Close()
            End Try
        End Using
    End Function

    ' Posts random Employee data to the provided target block.
    Private Shared Sub PostRandomEmployees(ByVal target As ITargetBlock(Of Employee), ByVal count As
Integer)
        Console.WriteLine("Adding {0} entries to Employee table...", count)

        For i As Integer = 0 To count - 1
            target.Post(Employee.Random())
        Next i
    End Sub

    ' Adds random employee data to the database by using dataflow.
    Private Shared Sub AddEmployees(ByVal connectionString As String, ByVal count As Integer)
        ' Create an ActionBlock<Employee> object that adds a single
        ' employee entry to the database.
        Dim insertEmployee = New ActionBlock(Of Employee)(Sub(e) InsertEmployees(New Employee() {e},
connectionString))

        ' Post several random Employee objects to the dataflow block.
        PostRandomEmployees(insertEmployee, count)

        ' Set the dataflow block to the completed state and wait for
        ' all insert operations to complete.
        insertEmployee.Complete()
        insertEmployee.Completion.Wait()
    End Sub

    ' Adds random employee data to the database by using dataflow.
    ' This method is similar to AddEmployees except that it uses batching
    ' to add multiple employees to the database at a time.
    Private Shared Sub AddEmployeesBatched(ByVal connectionString As String, ByVal batchSize As Integer,
ByVal count As Integer)
        ' Create a BatchBlock<Employee> that holds several Employee objects and
        ' then propagates them out as an array.
        Dim batchEmployees = New BatchBlock(Of Employee)(batchSize)

        ' Create an ActionBlock<Employee[]> object that adds multiple
        ' employee entries to the database.
        Dim insertEmployees = New ActionBlock(Of Employee[])(Sub(a) Program.InsertEmployees(a,
connectionString))

```

```

' Link the batch block to the action block.
batchEmployees.LinkTo(insertEmployees)

' When the batch block completes, set the action block also to complete.
batchEmployees.Completion.ContinueWith(Sub() insertEmployees.Complete())

' Post several random Employee objects to the batch block.
PostRandomEmployees(batchEmployees, count)

' Set the batch block to the completed state and wait for
' all insert operations to complete.
batchEmployees.Complete()
insertEmployees.Completion.Wait()

End Sub

' Displays information about several random employees to the console.
Private Shared Sub GetRandomEmployees(ByVal connectionString As String, ByVal batchSize As Integer,
ByVal count As Integer)
    ' Create a BatchedJoinBlock<Employee, Exception> object that holds
    ' both employee and exception data.
    Dim selectEmployees = New BatchedJoinBlock(Of Employee, Exception)(batchSize)

    ' Holds the total number of exceptions that occurred.
    Dim totalErrors As Integer = 0

    ' Create an action block that prints employee and error information
    ' to the console.
    Dim printEmployees = New ActionBlock(Of Tuple(Of IList(Of Employee), IList(Of Exception)))(Sub(data)
        Print information about the employees in this batch.

        Print the error count for this batch.

        Update total error count.

        Console.WriteLine("Received a batch...")

        For Each e As Employee In data.Item1
            Console.WriteLine("Last={0} First={1} ID={2}", e.LastName, e.FirstName, e.EmployeeID)
        Next e

        Console.WriteLine("There were {0} errors in this batch...", data.Item2.Count)
        totalErrors += data.Item2.Count
    End Sub)

    ' Link the batched join block to the action block.
    selectEmployees.LinkTo(printEmployees)

    ' When the batched join block completes, set the action block also to complete.
    selectEmployees.Completion.ContinueWith(Sub() printEmployees.Complete())

    ' Try to retrieve the ID for several random employees.
    Console.WriteLine("Selecting random entries from Employees table...")
    For i As Integer = 0 To count - 1
        Try
            ' Create a random employee.
            Dim e As Employee = Employee.Random()

            ' Try to retrieve the ID for the employee from the database.
            e.EmployeeID = GetEmployeeID(e.LastName, e.FirstName, connectionString)

            ' Post the Employee object to the Employee target of
            ' the batched join block.
        End Try
    Next i
End Sub

```

```

        selectEmployees.Target1.Post(e)
    Catch e As NullReferenceException
        ' GetEmployeeID throws NullReferenceException when there is
        ' no such employee with the given name. When this happens,
        ' post the Exception object to the Exception target of
        ' the batched join block.
        selectEmployees.Target2.Post(e)
    End Try
    Next i

    ' Set the batched join block to the completed state and wait for
    ' all retrieval operations to complete.
    selectEmployees.Complete()
    printEmployees.Completion.Wait()

    ' Print the total error count.
    Console.WriteLine("Finished. There were {0} total errors.", totalErrors)
End Sub

Shared Sub Main(ByVal args() As String)
    ' Create a connection string for accessing the database.
    ' The connection string refers to the temporary database location.
    Dim connectionString As String = String.Format("Data Source={0}", scratchDatabase)

    ' Create a Stopwatch object to time database insert operations.
    Dim stopwatch As New Stopwatch()

    ' Start with a clean database file by copying the source database to
    ' the temporary location.
    File.Copy(sourceDatabase, scratchDatabase, True)

    ' Demonstrate multiple insert operations without batching.
    Console.WriteLine("Demonstrating non-batched database insert operations...")
    Console.WriteLine("Original size of Employee table: {0}.", GetEmployeeCount(connectionString))
    stopwatch.Start()
    AddEmployees(connectionString, insertCount)
    stopwatch.Stop()
    Console.WriteLine("New size of Employee table: {0}; elapsed insert time: {1} ms.",
GetEmployeeCount(connectionString), stopwatch.ElapsedMilliseconds)

    Console.WriteLine()

    ' Start again with a clean database file.
    File.Copy(sourceDatabase, scratchDatabase, True)

    ' Demonstrate multiple insert operations, this time with batching.
    Console.WriteLine("Demonstrating batched database insert operations...")
    Console.WriteLine("Original size of Employee table: {0}.", GetEmployeeCount(connectionString))
    stopwatch.Restart()
    AddEmployeesBatched(connectionString, insertBatchSize, insertCount)
    stopwatch.Stop()
    Console.WriteLine("New size of Employee table: {0}; elapsed insert time: {1} ms.",
GetEmployeeCount(connectionString), stopwatch.ElapsedMilliseconds)

    Console.WriteLine()

    ' Start again with a clean database file.
    File.Copy(sourceDatabase, scratchDatabase, True)

    ' Demonstrate multiple retrieval operations with error reporting.
    Console.WriteLine("Demonstrating batched join database select operations...")
    ' Add a small number of employees to the database.
    AddEmployeesBatched(connectionString, insertBatchSize, 16)
    ' Query for random employees.
    GetRandomEmployees(connectionString, insertBatchSize, 10)
End Sub
End Class
End Namespace
' Sample output:

```

```
'Demonstrating non-batched database insert operations...
'Original size of Employee table: 15.
'Adding 256 entries to Employee table...
'New size of Employee table: 271; elapsed insert time: 11035 ms.
'

'Demonstrating batched database insert operations...
'Original size of Employee table: 15.
'Adding 256 entries to Employee table...
'New size of Employee table: 271; elapsed insert time: 197 ms.
'

'Demonstrating batched join database insert operations...
'Adding 16 entries to Employee table...
'Selecting items from Employee table...
'Received a batch...
'Last=Jones First=Tom ID=21
'Last=Jones First=John ID=24
'Last=Smith First=Tom ID=26
'Last=Jones First=Tom ID=21
'There were 4 errors in this batch...
'Received a batch...
'Last=Smith First=Tom ID=26
'Last=Jones First=Mike ID=28
'There were 0 errors in this batch...
'Finished. There were 4 total errors.
'
```

See also

- [Dataflow](#)

TPL and traditional .NET asynchronous programming

9/20/2022 • 17 minutes to read • [Edit Online](#)

.NET provides the following two standard patterns for performing I/O-bound and compute-bound asynchronous operations:

- Asynchronous Programming Model (APM), in which asynchronous operations are represented by a pair of begin/end methods. For example: `FileStream.BeginRead` and `Stream.EndRead`.
- Event-based asynchronous pattern (EAP), in which asynchronous operations are represented by a method/event pair that are named `<OperationName>Async` and `<OperationName>Completed`. For example: `WebClient.DownloadStringAsync` and `WebClient.DownloadStringCompleted`.

The Task Parallel Library (TPL) can be used in various ways in conjunction with either of the asynchronous patterns. You can expose both APM and EAP operations as `Task` objects to library consumers, or you can expose the APM patterns but use `Task` objects to implement them internally. In both scenarios, by using `Task` objects, you can simplify the code and take advantage of the following useful functionality:

- Register callbacks, in the form of task continuations, at any time after the task has started.
- Coordinate multiple operations that execute in response to a `Begin_` method by using the `ContinueWhenAll` and `ContinueWhenAny` methods, or the `WaitAll` and `WaitAny` methods.
- Encapsulate asynchronous I/O-bound and compute-bound operations in the same `Task` object.
- Monitor the status of the `Task` object.
- Marshal the status of an operation to a `Task` object by using `TaskCompletionSource<TResult>`.

Wrap APM operations in a Task

Both the `System.Threading.Tasks.TaskFactory` and `System.Threading.Tasks.TaskFactory<TResult>` classes provide several overloads of the `TaskFactory.FromAsync` and `TaskFactory<TResult>.FromAsync` methods that let you encapsulate an APM begin/end method pair in one `Task` or `Task<TResult>` instance. The various overloads accommodate any begin/end method pair that have from zero to three input parameters.

For pairs that have `End` methods that return a value (a `Function` in Visual Basic), use the methods in `TaskFactory<TResult>` that create a `Task<TResult>`. For `End` methods that return void (a `Sub` in Visual Basic), use the methods in `TaskFactory` that create a `Task`.

For those few cases in which the `Begin` method has more than three parameters or contains `ref` or `out` parameters, additional `FromAsync` overloads that encapsulate only the `End` method are provided.

The following example shows the signature for the `FromAsync` overload that matches the `FileStream.BeginRead` and `FileStream.EndRead` methods.

```

public Task<TResult> FromAsync<TArg1, TArg2, TArg3>(
    Func<TArg1, TArg2, TArg3, AsyncCallback, object, IAsyncResult> beginMethod, //BeginRead
    Func<IAsyncResult, TResult> endMethod, //EndRead
    TArg1 arg1, // the byte[] buffer
    TArg2 arg2, // the offset in arg1 at which to start writing data
    TArg3 arg3, // the maximum number of bytes to read
    object state // optional state information
)

```

```

Public Function FromAsync(Of TArg1, TArg2, TArg3)(
    ByVal beginMethod As Func(Of TArg1, TArg2, TArg3, AsyncCallback, Object, IAsyncResult),
    ByVal endMethod As Func(Of IAsyncResult, TResult),
    ByVal dataBuffer As TArg1,
    ByVal byteOffsetToStartAt As TArg2,
    ByVal maxBytesToRead As TArg3,
    ByVal stateInfo As Object)

```

This overload takes three input parameters, as follows. The first parameter is a `Func<T1,T2,T3,T4,T5,TResult>` delegate that matches the signature of the `FileStream.BeginRead` method. The second parameter is a `Func<T,TResult>` delegate that takes an `IAsyncResult` and returns a `TResult`. Because `EndRead` returns an integer, the compiler infers the type of `TResult` as `Int32` and the type of the task as `Task`. The last four parameters are identical to those in the `FileStream.BeginRead` method:

- The buffer in which to store the file data.
- The offset in the buffer at which to begin writing data.
- The maximum amount of data to read from the file.
- An optional object that stores user-defined state data to pass to the callback.

Use ContinueWith for the callback functionality

If you require access to the data in the file, as opposed to just the number of bytes, the `FromAsync` method is not sufficient. Instead, use `Task`, whose `Result` property contains the file data. You can do this by adding a continuation to the original task. The continuation performs the work that would typically be performed by the `AsyncCallback` delegate. It is invoked when the antecedent completes, and the data buffer has been filled. (The `FileStream` object should be closed before returning.)

The following example shows how to return a `Task` that encapsulates the `BeginRead / EndRead` pair of the `FileStream` class.

```
const int MAX_FILE_SIZE = 14000000;
public static Task<string> GetFileStringAsync(string path)
{
    FileInfo fi = new FileInfo(path);
    byte[] data = null;
    data = new byte[fi.Length];

    FileStream fs = new FileStream(path, FileMode.Open, FileAccess.Read, FileShare.Read, data.Length, true);

    //Task<int> returns the number of bytes read
    Task<int> task = Task<int>.Factory.FromAsync(
        fs.BeginRead, fs.EndRead, data, 0, data.Length, null);

    // It is possible to do other work here while waiting
    // for the antecedent task to complete.
    // ...

    // Add the continuation, which returns a Task<string>.
    return task.ContinueWith((antecedent) =>
    {
        fs.Close();

        // Result = "number of bytes read" (if we need it.)
        if (antecedent.Result < 100)
        {
            return "Data is too small to bother with.";
        }
        else
        {
            // If we did not receive the entire file, the end of the
            // data buffer will contain garbage.
            if (antecedent.Result < data.Length)
                Array.Resize(ref data, antecedent.Result);

            // Will be returned in the Result property of the Task<string>
            // at some future point after the asynchronous file I/O operation completes.
            return new UTF8Encoding().GetString(data);
        }
    });
});
```

```

Const MAX_FILE_SIZE As Integer = 14000000
Shared Function GetFileStringAsync(ByVal path As String) As Task(Of String)
    Dim fi As New FileInfo(path)
    Dim data(fi.Length - 1) As Byte

    Dim fs As FileStream = New FileStream(path, FileMode.Open, FileAccess.Read, FileShare.Read, data.Length,
True)

    ' Task(Of Integer) returns the number of bytes read
    Dim myTask As Task(Of Integer) = Task(Of Integer).Factory.FromAsync(
        AddressOf fs.BeginRead, AddressOf fs.EndRead, data, 0, data.Length, Nothing)

    ' It is possible to do other work here while waiting
    ' for the antecedent task to complete.
    ' ...

    ' Add the continuation, which returns a Task<string>.
    Return myTask.ContinueWith(Function(antecedent)
        fs.Close()
        If (antecedent.Result < 100) Then
            Return "Data is too small to bother with."
        End If
        ' If we did not receive the entire file, the end of the
        ' data buffer will contain garbage.
        If (antecedent.Result < data.Length) Then
            Array.Resize(data, antecedent.Result)
        End If

        ' Will be returned in the Result property of the Task<string>
        ' at some future point after the asynchronous file I/O operation
completes.
        Return New UTF8Encoding().GetString(data)
    End Function
End Function

```

The method can then be called, as follows.

```

Task<string> t = GetFileStringAsync(path);

// Do some other work:
// ...

try
{
    Console.WriteLine(t.Result.Substring(0, 500));
}
catch (AggregateException ae)
{
    Console.WriteLine(ae.InnerException.Message);
}

```

```

Dim myTask As Task(Of String) = GetFileStringAsync(path)

' Do some other work
' ...

Try
    Console.WriteLine(myTask.Result.Substring(0, 500))
Catch ex As AggregateException
    Console.WriteLine(ex.InnerException.Message)
End Try

```

Provide custom state data

In typical [IAsyncResult](#) operations, if your [AsyncCallback](#) delegate requires some custom state data, you have to pass it in through the last parameter in the `Begin` method, so that the data can be packaged into the [IAsyncResult](#) object that is eventually passed to the callback method. This is typically not required when the `FromAsync` methods are used. If the custom data is known to the continuation, then it can be captured directly in the continuation delegate. The following example resembles the previous example, but instead of examining the `Result` property of the antecedent, the continuation examines the custom state data that is directly accessible to the user delegate of the continuation.

```
public Task<string> GetFileStringAsync2(string path)
{
    FileInfo fi = new FileInfo(path);
    byte[] data = new byte[fi.Length];
    MyCustomState state = GetCustomState();
    FileStream fs = new FileStream(path, FileMode.Open, FileAccess.Read, FileShare.Read, data.Length, true);
    // We still pass null for the last parameter because
    // the state variable is visible to the continuation delegate.
    Task<int> task = Task<int>.Factory.FromAsync(
        fs.BeginRead, fs.EndRead, data, 0, data.Length, null);

    return task.ContinueWith((antecedent) =>
    {
        // It is safe to close the filestream now.
        fs.Close();

        // Capture custom state data directly in the user delegate.
        // No need to pass it through the FromAsync method.
        if (state.StateData.Contains("New York, New York"))
        {
            return "Start spreading the news!";
        }
        else
        {
            // If we did not receive the entire file, the end of the
            // data buffer will contain garbage.
            if (antecedent.Result < data.Length)
                Array.Resize(ref data, antecedent.Result);

            // Will be returned in the Result property of the Task<string>
            // at some future point after the asynchronous file I/O operation completes.
            return new UTF8Encoding().GetString(data);
        }
    });
}
```

```

Public Function GetFileStringAsync2(ByVal path As String) As Task(Of String)
    Dim fi = New FileInfo(path)
    Dim data(fi.Length - 1) As Byte
    Dim state As New MyCustomState()

    Dim fs As New FileStream(path, FileMode.Open, FileAccess.Read, FileShare.Read, data.Length, True)
    ' We still pass null for the last parameter because
    ' the state variable is visible to the continuation delegate.
    Dim myTask As Task(Of Integer) = Task(Of Integer).Factory.FromAsync(
        AddressOf fs.BeginRead, AddressOf fs.EndRead, data, 0, data.Length, Nothing)

    Return myTask.ContinueWith(Function(antecedent)
        fs.Close()
        ' Capture custom state data directly in the user delegate.
        ' No need to pass it through the FromAsync method.
        If (state.StateData.Contains("New York, New York")) Then
            Return "Start spreading the news!"
        End If

        ' If we did not receive the entire file, the end of the
        ' data buffer will contain garbage.
        If (antecedent.Result < data.Length) Then
            Array.Resize(data, antecedent.Result)
        End If
        '/ Will be returned in the Result property of the Task<string>
        '/ at some future point after the asynchronous file I/O operation
    completes.
        Return New UTF8Encoding().GetString(data)
    End Function)

End Function

```

Synchronize multiple FromAsync tasks

The static [ContinueWhenAll](#) and [ContinueWhenAny](#) methods provide added flexibility when used in conjunction with the `FromAsync` methods. The following example shows how to initiate multiple asynchronous I/O operations, and then wait for all of them to complete before you execute the continuation.

```

public Task<string> GetMultiFileData(string[] filesToRead)
{
    FileStream fs;
    Task<string>[] tasks = new Task<string>[filesToRead.Length];
    byte[] fileData = null;
    for (int i = 0; i < filesToRead.Length; i++)
    {
        fileData = new byte[0x1000];
        fs = new FileStream(filesToRead[i], FileMode.Open, FileAccess.Read, FileShare.Read, fileData.Length,
true);

        // By adding the continuation here, the
        // Result of each task will be a string.
        tasks[i] = Task<int>.Factory.FromAsync(
            fs.BeginRead, fs.EndRead, fileData, 0, fileData.Length, null)
            .ContinueWith((antecedent) =>
        {
            fs.Close();

            // If we did not receive the entire file, the end of the
            // data buffer will contain garbage.
            if (antecedent.Result < fileData.Length)
                Array.Resize(ref fileData, antecedent.Result);

            // Will be returned in the Result property of the Task<string>
            // at some future point after the asynchronous file I/O operation completes.
            return new UTF8Encoding().GetString(fileData);
        });
    }

    // Wait for all tasks to complete.
    return Task<string>.Factory.ContinueWhenAll(tasks, (data) =>
{
    // Propagate all exceptions and mark all faulted tasks as observed.
    Task.WaitAll(data);

    // Combine the results from all tasks.
    StringBuilder sb = new StringBuilder();
    foreach (var t in data)
    {
        sb.Append(t.Result);
    }
    // Final result to be returned eventually on the calling thread.
    return sb.ToString();
});
}

```

```

Public Function GetMultiFileData(ByVal filesToRead As String()) As Task(Of String)
    Dim fs As FileStream
    Dim tasks(filesToRead.Length - 1) As Task(Of String)
    Dim fileData() As Byte = Nothing
    For i As Integer = 0 To filesToRead.Length
        fileData(&H1000) = New Byte()
        fs = New FileStream(filesToRead(i), FileMode.Open, FileAccess.Read, FileShare.Read, fileData.Length,
True)

        ' By adding the continuation here, the
        ' Result of each task will be a string.
        tasks(i) = Task(Of Integer).Factory.FromAsync(AddressOf fs.BeginRead,
                                                AddressOf fs.EndRead,
                                                fileData,
                                                0,
                                                fileData.Length,
                                                Nothing).
        ContinueWith(Function(antecedent)
                    fs.Close()
                    'If we did not receive the entire file,
the end of the
                    ' data buffer will contain garbage.
                    If (antecedent.Result < fileData.Length)
Then
                    ReDim Preserve
fileData(antecedent.Result)
                    End If

                    'Will be returned in the Result property
of the Task<string>
                    ' at some future point after the
asynchronous file I/O operation completes.
                    Return New
UTF8Encoding().GetString(fileData)
                    End Function)
Next

Return Task(Of String).Factory.ContinueWhenAll(tasks, Function(data)
                                                ' Propagate all exceptions and mark all
faulted tasks as observed.
                                                Task.WaitAll(data)

                                                ' Combine the results from all tasks.
                                                Dim sb As New StringBuilder()
                                                For Each t As Task(Of String) In data
                                                    sb.Append(t.Result)
                                                Next
                                                ' Final result to be returned eventually on
the calling thread.
                                                Return sb.ToString()
End Function)
End Function

```

FromAsync tasks for only the End method

For those few cases in which the `Begin` method requires more than three input parameters or has `ref` or `out` parameters, you can use the `FromAsync` overloads, for example, `TaskFactory<TResult>.FromAsync(IAsyncResult, Func<IAsyncResult,TResult>)`, that represent only the `End` method. These methods can also be used in any scenario in which you're passed an `IAsyncResult` and want to encapsulate it in a Task.

```

static Task<String> ReturnTaskFromAsyncResult()
{
    IAsyncResult ar = DoSomethingAsynchronously();
    Task<String> t = Task<string>.Factory.FromAsync(ar, _ =>
    {
        return (string)ar.AsyncState;
    });

    return t;
}

```

```

Shared Function ReturnTaskFromAsyncResult() As Task(Of String)
    Dim ar As IAsyncResult = DoSomethingAsynchronously()
    Dim t As Task(Of String) = Task(Of String).Factory.FromAsync(ar, Function(res) CStr(res.AsyncState))
    Return t
End Function

```

Start and cancel FromAsync tasks

The task returned by a `FromAsync` method has a status of `WaitingForActivation` and will be started by the system at some point after the task is created. If you attempt to call `Start` on such a task, an exception will be raised.

You cannot cancel a `FromAsync` task, because the underlying .NET APIs currently do not support in-progress cancellation of file or network I/O. You can add cancellation functionality to a method that encapsulates a `FromAsync` call, but you can only respond to the cancellation before `FromAsync` is called or after it completed (for example, in a continuation task).

Some classes that support EAP, for example, [WebClient](#), do support cancellation, and you can integrate that native cancellation functionality by using cancellation tokens.

Expose complex EAP operations As tasks

The TPL does not provide any methods that are specifically designed to encapsulate an event-based asynchronous operation in the same way that the `FromAsync` family of methods wrap the `IAsyncResult` pattern. However, the TPL does provide the [System.Threading.Tasks.TaskCompletionSource<TResult>](#) class, which can be used to represent any arbitrary set of operations as a `Task<TResult>`. The operations may be synchronous or asynchronous, and may be I/O bound or compute-bound, or both.

The following example shows how to use a `TaskCompletionSource<TResult>` to expose a set of asynchronous [WebClient](#) operations to client code as a basic `Task<TResult>`. The method lets you enter an array of Web URLs, and a term or name to search for, and then returns the number of times the search term occurs on each site.

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Threading;
using System.Threading.Tasks;

public class SimpleWebExample
{
    public Task<string[]> GetWordCountsSimplified(string[] urls, string name,
                                                    CancellationToken token)
    {
        TaskCompletionSource<string[]> tcs = new TaskCompletionSource<string[]>();
        WebClient[] webClients = new WebClient[urls.Length];
        object m_lock = new object();
        int count = 0;

```

```

List<string> results = new List<string>();

// If the user cancels the CancellationToken, then we can use the
// WebClient's ability to cancel its own async operations.
token.Register(() =>
{
    foreach (var wc in webClients)
    {
        if (wc != null)
            wc.CancelAsync();
    }
});

for (int i = 0; i < urls.Length; i++)
{
    webClients[i] = new WebClient();

    #region callback
    // Specify the callback for the DownloadStringCompleted
    // event that will be raised by this WebClient instance.
    webClients[i].DownloadStringCompleted += (obj, args) =>
    {

        // Argument validation and exception handling omitted for brevity.

        // Split the string into an array of words,
        // then count the number of elements that match
        // the search term.
        string[] words = args.Result.Split(' ');
        string NAME = name.ToUpper();
        int nameCount = (from word in words.AsParallel()
                         where word.ToUpper().Contains(NAME)
                         select word)
                     .Count();

        // Associate the results with the url, and add new string to the array that
        // the underlying Task object will return in its Result property.
        lock (m_lock)
        {
            results.Add(String.Format("{0} has {1} instances of {2}", args.UserState, nameCount,
name));
        }

        // If this is the last async operation to complete,
        // then set the Result property on the underlying Task.
        count++;
        if (count == urls.Length)
        {
            tcs.TrySetResult(results.ToArray());
        }
    }
};

#endregion

// Call DownloadStringAsync for each URL.
Uri address = null;
address = new Uri(urls[i]);
webClients[i].DownloadStringAsync(address, address);
} // end for

// Return the underlying Task. The client code
// waits on the Result property, and handles exceptions
// in the try-catch block there.
return tcs.Task;
}
}

```

Imports System.Collections.Generic
Imports System.Net

```

Imports System
Imports System.Threading
Imports System.Threading.Tasks

Public Class SimpleWebExample
    Dim tcs As New TaskCompletionSource(Of String())
    Dim token As CancellationToken
    Dim results As New List(Of String)
    Dim m_lock As New Object()
    Dim count As Integer
    Dim addresses() As String
    Dim nameToSearch As String

    Public Function GetWordCountsSimplified(ByVal urls() As String, ByVal str As String,
                                             ByVal token As CancellationToken) As Task(Of String())
        addresses = urls
        nameToSearch = str

        Dim webClients(urls.Length - 1) As WebClient

        ' If the user cancels the CancellationToken, then we can use the
        ' WebClient's ability to cancel its own async operations.
        token.Register(Sub()
            For Each wc As WebClient In webClients
                If wc IsNot Nothing Then
                    wc.CancelAsync()
                End If
            Next
        End Sub)

        For i As Integer = 0 To urls.Length - 1
            webClients(i) = New WebClient()

            ' Specify the callback for the DownloadStringCompleted
            ' event that will be raised by this WebClient instance.
            AddHandler webClients(i).DownloadStringCompleted, AddressOf WebEventHandler

            Dim address As New Uri(urls(i))
            ' Pass the address, and also use it for the userToken
            ' to identify the page when the delegate is invoked.
            webClients(i).DownloadStringAsync(address, address)
        Next

        ' Return the underlying Task. The client code
        ' waits on the Result property, and handles exceptions
        ' in the try-catch block there.
        Return tcs.Task
    End Function

    Public Sub WebEventHandler(ByVal sender As Object, ByVal args As DownloadStringCompletedEventArgs)

        If args.Cancelled = True Then
            tcs.TrySetCanceled()
            Return
        ElseIf args.Error IsNot Nothing Then
            tcs.TrySetException(args.Error)
            Return
        Else
            ' Split the string into an array of words,
            ' then count the number of elements that match
            ' the search term.
            Dim words() As String = args.Result.Split(" "c)

            Dim name As String = nameToSearch.ToUpper()
            Dim nameCount = (From word In words.AsParallel()
                            Where word.ToUpper().Contains(name)
                            Select word).Count()

            ' Associate the results with the url, and add new string to the array that

```

```
' the underlying task object will return in its Result property.  
SyncLock (m_lock)  
    results.Add(String.Format("{0} has {1} instances of {2}", args.UserState, nameCount,  
nameToSearch))  
    count = count + 1  
    If (count = addresses.Length) Then  
        tcs.TrySetResult(results.ToArray())  
    End If  
End SyncLock  
End If  
End Sub  
End Class
```

For a more complete example, which includes additional exception handling and shows how to call the method from client code, see [How to: Wrap EAP Patterns in a Task](#).

Remember that any task that's created by a `TaskCompletionSource<TResult>` will be started by that `TaskCompletionSource` and, therefore, user code should not call the `Start` method on that task.

Implement the APM pattern by using tasks

In some scenarios, it may be desirable to directly expose the `IAsyncResult` pattern by using begin/end method pairs in an API. For example, you may want to maintain consistency with existing APIs, or you may have automated tools that require this pattern. In such cases, you can use `Task` objects to simplify how the APM pattern is implemented internally.

The following example shows how to use tasks to implement an APM begin/end method pair for a long-running compute-bound method.

```

class Calculator
{
    public IAsyncResult BeginCalculate(int decimalPlaces, AsyncCallback ac, object state)
    {
        Console.WriteLine("Calling BeginCalculate on thread {0}", Thread.CurrentThread.ManagedThreadId);
        Task<string> f = Task<string>.Factory.StartNew(_ => Compute(decimalPlaces), state);
        if (ac != null) f.ContinueWith((res) => ac(f));
        return f;
    }

    public string Compute(int numPlaces)
    {
        Console.WriteLine("Calling compute on thread {0}", Thread.CurrentThread.ManagedThreadId);

        // Simulating some heavy work.
        Thread.SpinWait(500000000);

        // Actual implementation left as exercise for the reader.
        // Several examples are available on the Web.
        return "3.14159265358979323846264338327950288";
    }

    public string EndCalculate(IAsyncResult ar)
    {
        Console.WriteLine("Calling EndCalculate on thread {0}", Thread.CurrentThread.ManagedThreadId);
        return ((Task<string>)ar).Result;
    }
}

public class CalculatorClient
{
    static int decimalPlaces = 12;
    public static void Main()
    {
        Calculator calc = new Calculator();
        int places = 35;

        AsyncCallback callBack = new AsyncCallback(PrintResult);
        IAsyncResult ar = calc.BeginCalculate(places, callBack, calc);

        // Do some work on this thread while the calculator is busy.
        Console.WriteLine("Working...");
        Thread.SpinWait(500000);
        Console.ReadLine();
    }

    public static void PrintResult(IAsyncResult result)
    {
        Calculator c = (Calculator)result.AsyncState;
        string piString = c.EndCalculate(result);
        Console.WriteLine("Calling PrintResult on thread {0}; result = {1}",
            Thread.CurrentThread.ManagedThreadId, piString);
    }
}

```

```

Class Calculator
    Public Function BeginCalculate(ByVal decimalPlaces As Integer, ByVal ac As AsyncCallback, ByVal state As Object) As IAsyncResult
        Console.WriteLine("Calling BeginCalculate on thread {0}", Thread.CurrentThread.ManagedThreadId)
        Dim myTask = Task(Of String).Factory.StartNew(Function(obj) Compute(decimalPlaces), state)
        myTask.ContinueWith(Sub(antedecent) ac(myTask))

    End Function
    Private Function Compute(ByVal decimalPlaces As Integer)
        Console.WriteLine("Calling compute on thread {0}", Thread.CurrentThread.ManagedThreadId)

        ' Simulating some heavy work.
        Thread.SpinWait(500000000)

        ' Actual implemenation left as exercise for the reader.
        ' Several examples are available on the Web.
        Return "3.14159265358979323846264338327950288"
    End Function

    Public Function EndCalculate(ByVal ar As IAsyncResult) As String
        Console.WriteLine("Calling EndCalculate on thread {0}", Thread.CurrentThread.ManagedThreadId)
        Return CType(ar, Task(Of String)).Result
    End Function
End Class

Class CalculatorClient
    Shared decimalPlaces As Integer
    Shared Sub Main()
        Dim calc As New Calculator
        Dim places As Integer = 35
        Dim callback As New AsyncCallback(AddressOf PrintResult)
        Dim ar As IAsyncResult = calc.BeginCalculate(places, callback, calc)

        ' Do some work on this thread while the calulator is busy.
        Console.WriteLine("Working...")
        Thread.SpinWait(50000)
        Console.ReadLine()
    End Sub

    Public Shared Sub PrintResult(ByVal result As IAsyncResult)
        Dim c As Calculator = CType(result.AsyncState, Calculator)
        Dim piString As String = c.EndCalculate(result)
        Console.WriteLine("Calling PrintResult on thread {0}; result = {1}",
            Thread.CurrentThread.ManagedThreadId, piString)
    End Sub
End Class

```

Use the StreamExtensions sample code

The *StreamExtensions.cs* file, in the [.NET Standard parallel extensions extras](#) repository, contains several reference implementations that use `Task` objects for asynchronous file and network I/O.

See also

- [Task Parallel Library \(TPL\)](#)

How to: Wrap EAP Patterns in a Task

9/20/2022 • 5 minutes to read • [Edit Online](#)

The following example shows how to expose an arbitrary sequence of Event-Based Asynchronous Pattern (EAP) operations as one task by using a `TaskCompletionSource<TResult>`. The example also shows how to use a `CancellationToken` to invoke the built-in cancellation methods on the `WebClient` objects.

Example

```
class WebDataDownloader
{
    static void Main()
    {
        WebDataDownloader downloader = new WebDataDownloader();
        string[] addresses = { "http://www.msnbc.com", "http://www.yahoo.com",
                               "http://www.nytimes.com", "http://www.washingtonpost.com",
                               "http://www.latimes.com", "http://www.newsday.com" };
        CancellationTokenSource cts = new CancellationTokenSource();

        // Create a UI thread from which to cancel the entire operation
        Task.Factory.StartNew(() =>
        {
            Console.WriteLine("Press c to cancel");
            if (Console.ReadKey().KeyChar == 'c')
                cts.Cancel();
        });

        // Using a neutral search term that is sure to get some hits.
        Task<string[]> webTask = downloader.GetWordCounts(addresses, "the", cts.Token);

        // Do some other work here unless the method has already completed.
        if (!webTask.IsCompleted)
        {
            // Simulate some work.
            Thread.Sleep(5000000);
        }

        string[] results = null;
        try
        {
            results = webTask.Result;
        }
        catch (AggregateException e)
        {
            foreach (var ex in e.InnerExceptions)
            {
                OperationCanceledException oce = ex as OperationCanceledException;
                if (oce != null)
                {
                    if (oce.CancellationToken == cts.Token)
                    {
                        Console.WriteLine("Operation canceled by user.");
                    }
                }
                else
                {
                    Console.WriteLine(ex.Message);
                }
            }
        }
    }
}
```

```

        }

        finally
        {
            cts.Dispose();
        }
    if (results != null)
    {
        foreach (var item in results)
            Console.WriteLine(item);
    }
    Console.ReadKey();
}

Task<string[]> GetWordCounts(string[] urls, string name, CancellationToken token)
{
    TaskCompletionSource<string[]> tcs = new TaskCompletionSource<string[]>();
    WebClient[] webClients = new WebClient[urls.Length];

    // If the user cancels the CancellationToken, then we can use the
    // WebClient's ability to cancel its own async operations.
    token.Register(() =>
    {
        foreach (var wc in webClients)
        {
            if (wc != null)
                wc.CancelAsync();
        }
    });

    object m_lock = new object();
    int count = 0;
    List<string> results = new List<string>();
    for (int i = 0; i < urls.Length; i++)
    {
        webClients[i] = new WebClient();

        #region callback
        // Specify the callback for the DownloadStringCompleted
        // event that will be raised by this WebClient instance.
        webClients[i].DownloadStringCompleted += (obj, args) =>
        {
            if (args.Cancelled == true)
            {
                tcs.TrySetCanceled();
                return;
            }
            else if (args.Error != null)
            {
                // Pass through to the underlying Task
                // any exceptions thrown by the WebClient
                // during the asynchronous operation.
                tcs.TrySetException(args.Error);
                return;
            }
            else
            {
                // Split the string into an array of words,
                // then count the number of elements that match
                // the search term.
                string[] words = null;
                words = args.Result.Split(' ');
                string NAME = name.ToUpper();
                int nameCount = (from word in words.AsParallel()
                                where word.ToUpper().Contains(NAME)
                                select word)
                            .Count();

                // Associate the results with the url, and add new string to the array that
                // the underlying Task object will return in its Result property.
                results.Add(new { url = urls[i], name = NAME, count = nameCount });
            }
        };
    }
}

```

```

        results.Add(String.Format("{0} has {1} instances of {2}", args.userState, nameCount,
name));
    }

    // If this is the last async operation to complete,
    // then set the Result property on the underlying Task.
    lock (m_lock)
    {
        count++;
        if (count == urls.Length)
        {
            tcs.TrySetResult(results.ToArray());
        }
    }
};

#endregion

// Call DownloadStringAsync for each URL.
Uri address = null;
try
{
    address = new Uri(urls[i]);
    // Pass the address, and also use it for the userToken
    // to identify the page when the delegate is invoked.
    webClients[i].DownloadStringAsync(address, address);
}

catch (UriFormatException ex)
{
    // Abandon the entire operation if one url is malformed.
    // Other actions are possible here.
    tcs.TrySetException(ex);
    return tcs.Task;
}
}

// Return the underlying Task. The client code
// waits on the Result property, and handles exceptions
// in the try-catch block there.
return tcs.Task;
}

```

```

Class WebDataDownLoader

Dim tcs As New TaskCompletionSource(Of String())
Dim nameToSearch As String
Dim token As CancellationToken
Dim results As New List(Of String)
Dim m_lock As Object
Dim count As Integer
Dim addresses() As String

Shared Sub Main()

    Dim downloader As New WebDataDownLoader()
    downloader.addresses = {"http://www.msnbc.com", "http://www.yahoo.com", _
                           "http://www.nytimes.com", "http://www.washingtonpost.com", _
                           "http://www.latimes.com", "http://www.newsday.com"}
    Dim cts As New CancellationTokenSource()

    ' Create a UI thread from which to cancel the entire operation
    Task.Factory.StartNew(Sub()
        Console.WriteLine("Press c to cancel")
        If Console.ReadKey().KeyChar = "c" Then
            cts.Cancel()
        End If
    End Sub)

```

```

' Using a neutral search term that is sure to get some hits on English web sites.
' Please substitute your favorite search term.
downloader.nameToSearch = "the"
Dim webTask As Task(Of String()) = downloader.GetWordCounts(downloader.addresses,
downloader.nameToSearch, cts.Token)

' Do some other work here unless the method has already completed.
If (webTask.IsCompleted = False) Then
    ' Simulate some work
    Thread.Sleep(5000000)
End If

Dim results As String() = Nothing
Try
    results = webTask.Result
Catch ae As AggregateException
    For Each ex As Exception In ae.InnerExceptions
        If (TypeOf (ex) Is OperationCanceledException) Then
            Dim oce As OperationCanceledException = CType(ex, OperationCanceledException)
            If oce.CancellationToken = cts.Token Then
                Console.WriteLine("Operation canceled by user.")
            End If
        Else
            Console.WriteLine(ex.Message)
        End If
    Next
Finally
    cts.Dispose()
End Try

If (Not results Is Nothing) Then
    For Each item As String In results
        Console.WriteLine(item)
    Next
End If

Console.WriteLine("Press any key to exit")
Console.ReadKey()
End Sub

Public Function GetWordCounts(ByVal urls() As String, ByVal str As String, ByVal token As
CancellationToken) As Task(Of String())

    Dim webClients() As WebClient
    ReDim webClients(urls.Length)
    m_lock = New Object()

    ' If the user cancels the CancellationToken, then we can use the
    ' WebClient's ability to cancel its own async operations.
    token.Register(Sub()
        For Each wc As WebClient In webClients
            If Not wc Is Nothing Then
                wc.CancelAsync()
            End If
        Next
    End Sub)

    For i As Integer = 0 To urls.Length - 1
        webClients(i) = New WebClient()

        ' Specify the callback for the DownloadStringCompleted
        ' event that will be raised by this WebClient instance.
        AddHandler webClients(i).DownloadStringCompleted, AddressOf WebEventHandler

        Dim address As Uri = Nothing
        Try

```

```

        address = New Uri(uris(i))
        ' Pass the address, and also use it for the userToken
        ' to identify the page when the delegate is invoked.
        webClients(i).DownloadStringAsync(address, address)
    Catch ex As UriFormatException
        tcs.TrySetException(ex)
        Return tcs.Task
    End Try

    Next

    ' Return the underlying Task. The client code
    ' waits on the Result property, and handles exceptions
    ' in the try-catch block there.
    Return tcs.Task
End Function

Public Sub WebEventHandler(ByVal sender As Object, ByVal args As DownloadStringCompletedEventArgs)

    If args.Cancelled = True Then
        tcs.TrySetCanceled()
        Return
    ElseIf Not args.Error Is Nothing Then
        tcs.TrySetException(args.Error)
        Return
    Else
        ' Split the string into an array of words,
        ' then count the number of elements that match
        ' the search term.
        Dim words() As String = args.Result.Split(" "c)
        Dim NAME As String = nameToSearch.ToUpper()
        Dim nameCount = (From word In words.AsParallel()
                         Where word.ToUpper().Contains(NAME)
                         Select word).Count()

        ' Associate the results with the url, and add new string to the array that
        ' the underlying Task object will return in its Result property.
        results.Add(String.Format("{0} has {1} instances of {2}", args.UserState, nameCount,
nameToSearch))
    End If

    SyncLock (m_lock)
        count = count + 1
        If (count = addresses.Length) Then
            tcs.TrySetResult(results.ToArray())
        End If
    End SyncLock
End Sub

```

See also

- [TPL and Traditional .NET Asynchronous Programming](#)

Potential Pitfalls in Data and Task Parallelism

9/20/2022 • 7 minutes to read • [Edit Online](#)

In many cases, [Parallel.For](#) and [Parallel.ForEach](#) can provide significant performance improvements over ordinary sequential loops. However, the work of parallelizing the loop introduces complexity that can lead to problems that, in sequential code, are not as common or are not encountered at all. This topic lists some practices to avoid when you write parallel loops.

Do Not Assume That Parallel Is Always Faster

In certain cases a parallel loop might run slower than its sequential equivalent. The basic rule of thumb is that parallel loops that have few iterations and fast user delegates are unlikely to speedup much. However, because many factors are involved in performance, we recommend that you always measure actual results.

Avoid Writing to Shared Memory Locations

In sequential code, it is not uncommon to read from or write to static variables or class fields. However, whenever multiple threads are accessing such variables concurrently, there is a big potential for race conditions. Even though you can use locks to synchronize access to the variable, the cost of synchronization can hurt performance. Therefore, we recommend that you avoid, or at least limit, access to shared state in a parallel loop as much as possible. The best way to do this is to use the overloads of [Parallel.For](#) and [Parallel.ForEach](#) that use a [System.Threading.ThreadLocal<T>](#) variable to store thread-local state during loop execution. For more information, see [How to: Write a Parallel.For Loop with Thread-Local Variables](#) and [How to: Write a Parallel.ForEach Loop with Partition-Local Variables](#).

Avoid Over-Parallelization

By using parallel loops, you incur the overhead costs of partitioning the source collection and synchronizing the worker threads. The benefits of parallelization are further limited by the number of processors on the computer. There is no speedup to be gained by running multiple compute-bound threads on just one processor. Therefore, you must be careful not to over-parallelize a loop.

The most common scenario in which over-parallelization can occur is in nested loops. In most cases, it is best to parallelize only the outer loop unless one or more of the following conditions apply:

- The inner loop is known to be very long.
- You are performing an expensive computation on each order. (The operation shown in the example is not expensive.)
- The target system is known to have enough processors to handle the number of threads that will be produced by parallelizing the query on `cust.Orders`.

In all cases, the best way to determine the optimum query shape is to test and measure.

Avoid Calls to Non-Thread-Safe Methods

Writing to non-thread-safe instance methods from a parallel loop can lead to data corruption which may or may not go undetected in your program. It can also lead to exceptions. In the following example, multiple threads would be attempting to call the [FileStream.WriteByte](#) method simultaneously, which is not supported by the class.

```
FileStream fs = File.OpenWrite(path);
byte[] bytes = new Byte[10000000];
// ...
Parallel.For(0, bytes.Length, (i) => fs.WriteByte(bytes[i]));
```

```
Dim fs As FileStream = File.OpenWrite(filepath)
Dim bytes() As Byte
ReDim bytes(1000000)
' ...init byte array
Parallel.For(0, bytes.Length, Sub(n) fs.WriteByte(bytes(n)))
```

Limit Calls to Thread-Safe Methods

Most static methods in .NET are thread-safe and can be called from multiple threads concurrently. However, even in these cases, the synchronization involved can lead to significant slowdown in the query.

NOTE

You can test for this yourself by inserting some calls to [WriteLine](#) in your queries. Although this method is used in the documentation examples for demonstration purposes, do not use it in parallel loops unless necessary.

Be Aware of Thread Affinity Issues

Some technologies, for example, COM interoperability for Single-Threaded Apartment (STA) components, Windows Forms, and Windows Presentation Foundation (WPF), impose thread affinity restrictions that require code to run on a specific thread. For example, in both Windows Forms and WPF, a control can only be accessed on the thread on which it was created. This means, for example, that you cannot update a list control from a parallel loop unless you configure the thread scheduler to schedule work only on the UI thread. For more information, see [Specifying a synchronization context](#).

Use Caution When Waiting in Delegates That Are Called by Parallel.Invoke

In certain circumstances, the Task Parallel Library will inline a task, which means it runs on the task on the currently executing thread. (For more information, see [Task Schedulers](#).) This performance optimization can lead to deadlock in certain cases. For example, two tasks might run the same delegate code, which signals when an event occurs, and then waits for the other task to signal. If the second task is inlined on the same thread as the first, and the first goes into a Wait state, the second task will never be able to signal its event. To avoid such an occurrence, you can specify a timeout on the Wait operation, or use explicit thread constructors to help ensure that one task cannot block the other.

Do Not Assume that Iterations of ForEach, For and ForAll Always Execute in Parallel

It is important to keep in mind that individual iterations in a [For](#), [ForEach](#) or [ForAll](#) loop may but do not have to execute in parallel. Therefore, you should avoid writing any code that depends for correctness on parallel execution of iterations or on the execution of iterations in any particular order. For example, this code is likely to deadlock:

```

ManualResetEventSlim mre = new ManualResetEventSlim();
Enumerable.Range(0, Environment.ProcessorCount * 100)
    .AsParallel()
    .ForAll((j) =>
{
    if (j == Environment.ProcessorCount)
    {
        Console.WriteLine("Set on {0} with value of {1}",
            Thread.CurrentThread.ManagedThreadId, j);
        mre.Set();
    }
    else
    {
        Console.WriteLine("Waiting on {0} with value of {1}",
            Thread.CurrentThread.ManagedThreadId, j);
        mre.Wait();
    }
}); //deadlocks

```

```

Dim mres = New ManualResetEventSlim()
Enumerable.Range(0, Environment.ProcessorCount * 100) _
    .AsParallel() _
    .ForAll(Sub(j)

        If j = Environment.ProcessorCount Then
            Console.WriteLine("Set on {0} with value of {1}",
                Thread.CurrentThread.ManagedThreadId, j)
            mres.Set()
        Else
            Console.WriteLine("Waiting on {0} with value of {1}",
                Thread.CurrentThread.ManagedThreadId, j)
            mres.Wait()
        End If
    End Sub) ' deadlocks

```

In this example, one iteration sets an event, and all other iterations wait on the event. None of the waiting iterations can complete until the event-setting iteration has completed. However, it is possible that the waiting iterations block all threads that are used to execute the parallel loop, before the event-setting iteration has had a chance to execute. This results in a deadlock – the event-setting iteration will never execute, and the waiting iterations will never wake up.

In particular, one iteration of a parallel loop should never wait on another iteration of the loop to make progress. If the parallel loop decides to schedule the iterations sequentially but in the opposite order, a deadlock will occur.

Avoid Executing Parallel Loops on the UI Thread

It is important to keep your application's user interface (UI) responsive. If an operation contains enough work to warrant parallelization, then it likely should not be run on the UI thread. Instead, it should offload that operation to be run on a background thread. For example, if you want to use a parallel loop to compute some data that should then be rendered into a UI control, you should consider executing the loop within a task instance rather than directly in a UI event handler. Only when the core computation has completed should you then marshal the UI update back to the UI thread.

If you do run parallel loops on the UI thread, be careful to avoid updating UI controls from within the loop. Attempting to update UI controls from within a parallel loop that is executing on the UI thread can lead to state corruption, exceptions, delayed updates, and even deadlocks, depending on how the UI update is invoked. In the following example, the parallel loop blocks the UI thread on which it's executing until all iterations are complete. However, if an iteration of the loop is running on a background thread (as [For](#) may do), the call to [Invoke](#) causes a message to be submitted to the UI thread and blocks waiting for that message to be processed. Since the UI

thread is blocked running the [For](#), the message can never be processed, and the UI thread deadlocks.

```
private void button1_Click(object sender, EventArgs e)
{
    Parallel.For(0, N, i =>
    {
        // do work for i
        button1.Invoke((Action)delegate { DisplayProgress(i); });
    });
}
```

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click

    Dim iterations As Integer = 20
    Parallel.For(0, iterations, Sub(x)
        Button1.Invoke(Sub()
            DisplayProgress(x)
        End Sub)
    End Sub)
End Sub
```

The following example shows how to avoid the deadlock, by running the loop inside a task instance. The UI thread is not blocked by the loop, and the message can be processed.

```
private void button1_Click(object sender, EventArgs e)
{
    Task.Factory.StartNew(() =>
        Parallel.For(0, N, i =>
        {
            // do work for i
            button1.Invoke((Action)delegate { DisplayProgress(i); });
        }));
}
```

```
Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click

    Dim iterations As Integer = 20
    Task.Factory.StartNew(Sub() Parallel.For(0, iterations, Sub(x)
        Button1.Invoke(Sub()
            DisplayProgress(x)
        End Sub))
    End Sub))
```

See also

- [Parallel Programming](#)
- [Potential Pitfalls with PLINQ](#)
- [Patterns for Parallel Programming: Understanding and Applying Parallel Patterns with the .NET Framework 4](#)

Introduction to PLINQ

9/20/2022 • 11 minutes to read • [Edit Online](#)

Parallel LINQ (PLINQ) is a parallel implementation of the [Language-Integrated Query \(LINQ\)](#) pattern. PLINQ implements the full set of LINQ standard query operators as extension methods for the [System.Linq](#) namespace and has additional operators for parallel operations. PLINQ combines the simplicity and readability of LINQ syntax with the power of parallel programming.

TIP

If you're not familiar with LINQ, it features a unified model for querying any enumerable data source in a type-safe manner. LINQ to Objects is the name for LINQ queries that are run against in-memory collections such as [List<T>](#) and arrays. This article assumes that you have a basic understanding of LINQ. For more information, see [Language-Integrated Query \(LINQ\)](#).

What is a Parallel query?

A PLINQ query in many ways resembles a non-parallel LINQ to Objects query. PLINQ queries, just like sequential LINQ queries, operate on any in-memory [IEnumerable](#) or [IEnumerable<T>](#) data source, and have deferred execution, which means they do not begin executing until the query is enumerated. The primary difference is that PLINQ attempts to make full use of all the processors on the system. It does this by partitioning the data source into segments, and then executing the query on each segment on separate worker threads in parallel on multiple processors. In many cases, parallel execution means that the query runs significantly faster.

Through parallel execution, PLINQ can achieve significant performance improvements over legacy code for certain kinds of queries, often just by adding the [AsParallel](#) query operation to the data source. However, parallelism can introduce its own complexities, and not all query operations run faster in PLINQ. In fact, parallelization actually slows down certain queries. Therefore, you should understand how issues such as ordering affect parallel queries. For more information, see [Understanding Speedup in PLINQ](#).

NOTE

This documentation uses lambda expressions to define delegates in PLINQ. If you are not familiar with lambda expressions in C# or Visual Basic, see [Lambda Expressions in PLINQ and TPL](#).

The remainder of this article gives an overview of the main PLINQ classes and discusses how to create PLINQ queries. Each section contains links to more detailed information and code examples.

The ParallelEnumerable Class

The [System.Linq.ParallelEnumerable](#) class exposes almost all of PLINQ's functionality. It and the rest of the [System.Linq](#) namespace types are compiled into the [System.Core.dll](#) assembly. The default C# and Visual Basic projects in Visual Studio both reference the assembly and import the namespace.

[ParallelEnumerable](#) includes implementations of all the standard query operators that LINQ to Objects supports, although it does not attempt to parallelize each one. If you are not familiar with LINQ, see [Introduction to LINQ \(C#\)](#) and [Introduction to LINQ \(Visual Basic\)](#).

In addition to the standard query operators, the [ParallelEnumerable](#) class contains a set of methods that enable

behaviors specific to parallel execution. These PLINQ-specific methods are listed in the following table.

PARALLELENUMERABLE OPERATOR	DESCRIPTION
AsParallel	The entry point for PLINQ. Specifies that the rest of the query should be parallelized, if it is possible.
AsSequential	Specifies that the rest of the query should be run sequentially, as a non-parallel LINQ query.
AsOrdered	Specifies that PLINQ should preserve the ordering of the source sequence for the rest of the query, or until the ordering is changed, for example by the use of an orderby (Order By in Visual Basic) clause.
AsUnordered	Specifies that PLINQ for the rest of the query is not required to preserve the ordering of the source sequence.
WithCancellation	Specifies that PLINQ should periodically monitor the state of the provided cancellation token and cancel execution if it is requested.
WithDegreeOfParallelism	Specifies the maximum number of processors that PLINQ should use to parallelize the query.
WithMergeOptions	Provides a hint about how PLINQ should, if it is possible, merge parallel results back into just one sequence on the consuming thread.
WithExecutionMode	Specifies whether PLINQ should parallelize the query even when the default behavior would be to run it sequentially.
ForAll	A multithreaded enumeration method that, unlike iterating over the results of the query, enables results to be processed in parallel without first merging back to the consumer thread.
Aggregate overload	An overload that is unique to PLINQ and enables intermediate aggregation over thread-local partitions, plus a final aggregation function to combine the results of all partitions.

The Opt-in Model

When you write a query, opt in to PLINQ by invoking the [ParallelEnumerable.AsParallel](#) extension method on the data source, as shown in the following example.

```
var source = Enumerable.Range(1, 10000);

// Opt in to PLINQ with AsParallel.
var evenNums = from num in source.AsParallel()
               where num % 2 == 0
               select num;
Console.WriteLine("{0} even numbers out of {1} total",
                 evenNums.Count(), source.Count());
// The example displays the following output:
//      5000 even numbers out of 10000 total
```

```

Dim source = Enumerable.Range(1, 10000)

' Opt in to PLINQ with AsParallel
Dim evenNums = From num In source.AsParallel()
    Where num Mod 2 = 0
    Select num
Console.WriteLine("{0} even numbers out of {1} total",
                  evenNums.Count(), source.Count())
' The example displays the following output:
'      5000 even numbers out of 10000 total

```

The [AsParallel](#) extension method binds the subsequent query operators, in this case, `where` and `select`, to the [System.Linq.ParallelEnumerable](#) implementations.

Execution Modes

By default, PLINQ is conservative. At run time, the PLINQ infrastructure analyzes the overall structure of the query. If the query is likely to yield speedups by parallelization, PLINQ partitions the source sequence into tasks that can be run concurrently. If it is not safe to parallelize a query, PLINQ just runs the query sequentially. If PLINQ has a choice between a potentially expensive parallel algorithm or an inexpensive sequential algorithm, it chooses the sequential algorithm by default. You can use the [WithExecutionMode](#) method and the [System.Linq.ParallelExecutionMode](#) enumeration to instruct PLINQ to select the parallel algorithm. This is useful when you know by testing and measurement that a particular query executes faster in parallel. For more information, see [How to: Specify the Execution Mode in PLINQ](#).

Degree of Parallelism

By default, PLINQ uses all of the processors on the host computer. You can instruct PLINQ to use no more than a specified number of processors by using the [WithDegreeOfParallelism](#) method. This is useful when you want to make sure that other processes running on the computer receive a certain amount of CPU time. The following snippet limits the query to utilizing a maximum of two processors.

```

var query = from item in source.AsParallel().WithDegreeOfParallelism(2)
            where Compute(item) > 42
            select item;

```

```

Dim query = From item In source.AsParallel().WithDegreeOfParallelism(2)
            Where Compute(item) > 42
            Select item

```

In cases where a query is performing a significant amount of non-compute-bound work such as File I/O, it might be beneficial to specify a degree of parallelism greater than the number of cores on the machine.

Ordered Versus Unordered Parallel Queries

In some queries, a query operator must produce results that preserve the ordering of the source sequence. PLINQ provides the [AsOrdered](#) operator for this purpose. [AsOrdered](#) is distinct from [AsSequential](#). An [AsOrdered](#) sequence is still processed in parallel, but its results are buffered and sorted. Because order preservation typically involves extra work, an [AsOrdered](#) sequence might be processed more slowly than the default [AsUnordered](#) sequence. Whether a particular ordered parallel operation is faster than a sequential version of the operation depends on many factors.

The following code example shows how to opt in to order preservation.

```
var evenNums =
    from num in numbers.AsParallel().AsOrdered()
    where num % 2 == 0
    select num;
```

```
Dim evenNums = From num In numbers.AsParallel().AsOrdered()
    Where num Mod 2 = 0
    Select num
```

For more information, see [Order Preservation in PLINQ](#).

Parallel vs. Sequential Queries

Some operations require that the source data be delivered in a sequential manner. The [ParallelEnumerable](#) query operators revert to sequential mode automatically when it is required. For user-defined query operators and user delegates that require sequential execution, PLINQ provides the [AsSequential](#) method. When you use [AsSequential](#), all subsequent operators in the query are executed sequentially until [AsParallel](#) is called again. For more information, see [How to: Combine Parallel and Sequential LINQ Queries](#).

Options for Merging Query Results

When a PLINQ query executes in parallel, its results from each worker thread must be merged back onto the main thread for consumption by a `foreach` loop (`For Each` in Visual Basic), or insertion into a list or array. In some cases, it might be beneficial to specify a particular kind of merge operation, for example, to begin producing results more quickly. For this purpose, PLINQ supports the [WithMergeOptions](#) method, and the [ParallelMergeOptions](#) enumeration. For more information, see [Merge Options in PLINQ](#).

The ForAll Operator

In sequential LINQ queries, execution is deferred until the query is enumerated either in a `foreach` (`For Each` in Visual Basic) loop or by invoking a method such as [ToList](#), [ToArray](#), or [ToDictionary](#). In PLINQ, you can also use `foreach` to execute the query and iterate through the results. However, `foreach` itself does not run in parallel, and therefore, it requires that the output from all parallel tasks be merged back into the thread on which the loop is running. In PLINQ, you can use `foreach` when you must preserve the final ordering of the query results, and also whenever you are processing the results in a serial manner, for example when you are calling `Console.WriteLine` for each element. For faster query execution when order preservation is not required and when the processing of the results can itself be parallelized, use the [ForAll](#) method to execute a PLINQ query. [ForAll](#) does not perform this final merge step. The following code example shows how to use the [ForAll](#) method. [System.Collections.Concurrent.ConcurrentBag<T>](#) is used here because it is optimized for multiple threads adding concurrently without attempting to remove any items.

```
var nums = Enumerable.Range(10, 10000);
var query =
    from num in nums.AsParallel()
    where num % 10 == 0
    select num;

// Process the results as each thread completes
// and add them to a System.Collections.Concurrent.ConcurrentBag(Of Int)
// which can safely accept concurrent add operations
query.ForAll(e => concurrentBag.Add(Compute(e)));
```

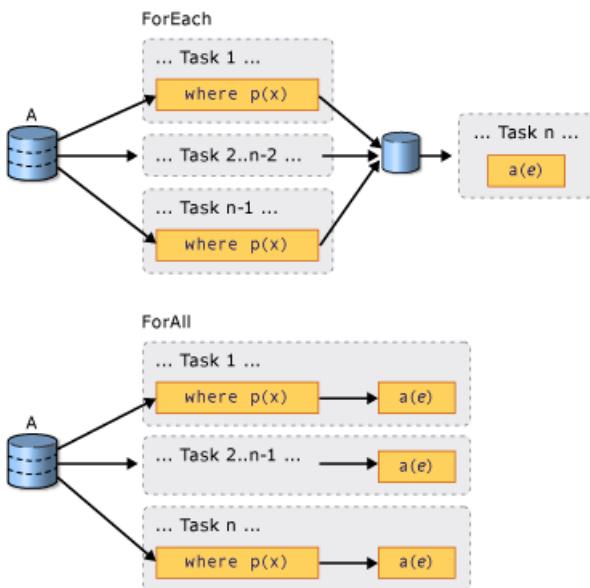
```

Dim nums = Enumerable.Range(10, 10000)
Dim query = From num In nums.AsParallel()
            Where num Mod 10 = 0
            Select num

' Process the results as each thread completes
' and add them to a System.Collections.Concurrent.ConcurrentBag(Of Int)
' which can safely accept concurrent add operations
query.ForAll(Sub(e) concurrentBag.Add(Compute(e)))

```

The following illustration shows the difference between `foreach` and `ForAll` with regard to query execution.



Cancellation

PLINQ is integrated with the cancellation types in .NET. (For more information, see [Cancellation in Managed Threads](#).) Therefore, unlike sequential LINQ to Objects queries, PLINQ queries can be canceled. To create a cancelable PLINQ query, use the `WithCancellation` operator on the query and provide a `CancellationToken` instance as the argument. When the `IsCancellationRequested` property on the token is set to true, PLINQ will notice it, stop processing on all threads, and throw an `OperationCanceledException`.

It is possible that a PLINQ query might continue to process some elements after the cancellation token is set.

For greater responsiveness, you can also respond to cancellation requests in long-running user delegates. For more information, see [How to: Cancel a PLINQ Query](#).

Exceptions

When a PLINQ query executes, multiple exceptions might be thrown from different threads simultaneously. Also, the code to handle the exception might be on a different thread than the code that threw the exception. PLINQ uses the `AggregateException` type to encapsulate all the exceptions that were thrown by a query, and marshal those exceptions back to the calling thread. On the calling thread, only one try-catch block is required. However, you can iterate through all of the exceptions that are encapsulated in the `AggregateException` and catch any that you can safely recover from. In rare cases, some exceptions may be thrown that are not wrapped in an `AggregateException`, and `ThreadAbortException`s are also not wrapped.

When exceptions are allowed to bubble up back to the joining thread, then it is possible that a query may continue to process some items after the exception is raised.

For more information, see [How to: Handle Exceptions in a PLINQ Query](#).

Custom Partitioners

In some cases, you can improve query performance by writing a custom partitioner that takes advantage of some characteristic of the source data. In the query, the custom partitioner itself is the enumerable object that is queried.

```
int[] arr = new int[9999];
Partitioner<int> partitioner = new MyArrayPartitioner<int>(arr);
var query = partitioner.AsParallel().Select(SomeFunction);
```

```
Dim arr(10000) As Integer
Dim partitioner As Partitioner(Of Integer) = New MyArrayPartitioner(Of Integer)(arr)
Dim query = partitioner.AsParallel().Select(Function(x) SomeFunction(x))
```

PLINQ supports a fixed number of partitions (although data may be dynamically reassigned to those partitions during run time for load balancing.). [For](#) and [ForEach](#) support only dynamic partitioning, which means that the number of partitions changes at run time. For more information, see [Custom Partitioners for PLINQ and TPL](#).

Measuring PLINQ Performance

In many cases, a query can be parallelized, but the overhead of setting up the parallel query outweighs the performance benefit gained. If a query does not perform much computation or if the data source is small, a PLINQ query may be slower than a sequential LINQ to Objects query. You can use the Parallel Performance Analyzer in Visual Studio Team Server to compare the performance of various queries, to locate processing bottlenecks, and to determine whether your query is running in parallel or sequentially. For more information, see [Concurrency Visualizer](#) and [How to: Measure PLINQ Query Performance](#).

See also

- [Parallel LINQ \(PLINQ\)](#)
- [Understanding Speedup in PLINQ](#)

Understanding Speedup in PLINQ

9/20/2022 • 6 minutes to read • [Edit Online](#)

The primary purpose of PLINQ is to speed up the execution of LINQ to Objects queries by executing the query delegates in parallel on multi-core computers. PLINQ performs best when the processing of each element in a source collection is independent, with no shared state involved among the individual delegates. Such operations are common in LINQ to Objects and PLINQ, and are often called "*delightfully parallel*" because they lend themselves easily to scheduling on multiple threads. However, not all queries consist entirely of delightfully parallel operations; in most cases, a query involves some operators that either cannot be parallelized, or that slow down parallel execution. And even with queries that are entirely delightfully parallel, PLINQ must still partition the data source and schedule the work on the threads, and usually merge the results when the query completes. All these operations add to the computational cost of parallelization; these costs of adding parallelization are called *overhead*. To achieve optimum performance in a PLINQ query, the goal is to maximize the parts that are delightfully parallel and minimize the parts that require overhead. This article provides information that will help you write PLINQ queries that are as efficient as possible while still yielding correct results.

Factors that Impact PLINQ Query Performance

The following sections lists some of the most important factors that impact parallel query performance. These are general statements that by themselves are not sufficient to predict query performance in all cases. As always, it is important to measure actual performance of specific queries on computers with a range of representative configurations and loads.

1. Computational cost of the overall work.

To achieve speedup, a PLINQ query must have enough delightfully parallel work to offset the overhead. The work can be expressed as the computational cost of each delegate multiplied by the number of elements in the source collection. Assuming that an operation can be parallelized, the more computationally expensive it is, the greater the opportunity for speedup. For example, if a function takes one millisecond to execute, a sequential query over 1000 elements will take one second to perform that operation, whereas a parallel query on a computer with four cores might take only 250 milliseconds. This yields a speedup of 750 milliseconds. If the function required one second to execute for each element, then the speedup would be 750 seconds. If the delegate is very expensive, then PLINQ might offer significant speedup with only a few items in the source collection. Conversely, small source collections with trivial delegates are generally not good candidates for PLINQ.

In the following example, queryA is probably a good candidate for PLINQ, assuming that its Select function involves a lot of work. queryB is probably not a good candidate because there is not enough work in the Select statement, and the overhead of parallelization will offset most or all of the speedup.

```
Dim queryA = From num In numberList.AsParallel()
              Select ExpensiveFunction(num); 'good for PLINQ

Dim queryB = From num In numberList.AsParallel()
              Where num Mod 2 > 0
              Select num; 'not as good for PLINQ
```

```

var queryA = from num in numberList.AsParallel()
             select ExpensiveFunction(num); //good for PLINQ

var queryB = from num in numberList.AsParallel()
             where num % 2 > 0
             select num; //not as good for PLINQ

```

2. The number of logical cores on the system (degree of parallelism).

This point is an obvious corollary to the previous section, queries that are delightfully parallel run faster on machines with more cores because the work can be divided among more concurrent threads. The overall amount of speedup depends on what percentage of the overall work of the query is parallelizable. However, do not assume that all queries will run twice as fast on an eight core computer as a four core computer. When tuning queries for optimal performance, it is important to measure actual results on computers with various numbers of cores. This point is related to point #1: larger datasets are required to take advantage of greater computing resources.

3. The number and kind of operations.

PLINQ provides the `AsOrdered` operator for situations in which it is necessary to maintain the order of elements in the source sequence. There is a cost associated with ordering, but this cost is usually modest. `GroupBy` and `Join` operations likewise incur overhead. PLINQ performs best when it is allowed to process elements in the source collection in any order, and pass them to the next operator as soon as they are ready. For more information, see [Order Preservation in PLINQ](#).

4. The form of query execution.

If you are storing the results of a query by calling `ToArray` or `ToList`, then the results from all parallel threads must be merged into the single data structure. This involves an unavoidable computational cost. Likewise, if you iterate the results by using a `foreach` (`For Each` in Visual Basic) loop, the results from the worker threads need to be serialized onto the enumerator thread. But if you just want to perform some action based on the result from each thread, you can use the `ForAll` method to perform this work on multiple threads.

5. The type of merge options.

PLINQ can be configured to either buffer its output, and produce it in chunks or all at once after the entire result set is produced, or else to stream individual results as they are produced. The former results in decreased overall execution time and the latter results in decreased latency between yielded elements. While the merge options do not always have a major impact on overall query performance, they can impact perceived performance because they control how long a user must wait to see results. For more information, see [Merge Options in PLINQ](#).

6. The kind of partitioning.

In some cases, a PLINQ query over an indexable source collection may result in an unbalanced work load. When this occurs, you might be able to increase the query performance by creating a custom partitioner. For more information, see [Custom Partitioners for PLINQ and TPL](#).

When PLINQ Chooses Sequential Mode

PLINQ will always attempt to execute a query at least as fast as the query would run sequentially. Although PLINQ does not look at how computationally expensive the user delegates are, or how big the input source is, it does look for certain query "shapes." Specifically, it looks for query operators or combinations of operators that typically cause a query to execute more slowly in parallel mode. When it finds such shapes, PLINQ by default falls back to sequential mode.

However, after measuring a specific query's performance, you may determine that it actually runs faster in parallel mode. In such cases you can use the [ParallelExecutionMode.ForceParallelism](#) flag via the [WithExecutionMode](#) method to instruct PLINQ to parallelize the query. For more information, see [How to: Specify the Execution Mode in PLINQ](#).

The following list describes the query shapes that PLINQ by default will execute in sequential mode:

- Queries that contain a Select, indexed Where, indexed SelectMany, or ElementAt clause after an ordering or filtering operator that has removed or rearranged original indices.
- Queries that contain a Take, TakeWhile, Skip, SkipWhile operator and where indices in the source sequence are not in the original order.
- Queries that contain Zip or SequenceEquals, unless one of the data sources has an originally ordered index and the other data source is indexable (i.e. an array or `IList(T)`).
- Queries that contain Concat, unless it is applied to indexable data sources.
- Queries that contain Reverse, unless applied to an indexable data source.

See also

- [Parallel LINQ \(PLINQ\)](#)

Order Preservation in PLINQ

9/20/2022 • 5 minutes to read • [Edit Online](#)

In PLINQ, the goal is to maximize performance while maintaining correctness. A query should run as fast as possible but still produce the correct results. In some cases, correctness requires the order of the source sequence to be preserved; however, ordering can be computationally expensive. Therefore, by default, PLINQ does not preserve the order of the source sequence. In this regard, PLINQ resembles LINQ to SQL, but is unlike LINQ to Objects, which does preserve ordering.

To override the default behavior, you can turn on order-preservation by using the [AsOrdered](#) operator on the source sequence. You can then turn off order preservation later in the query by using the [AsUnordered](#) method. With both methods, the query is processed based on the heuristics that determine whether to execute the query as parallel or as sequential. For more information, see [Understanding Speedup in PLINQ](#).

The following example shows an unordered parallel query that filters for all the elements that match a condition, without trying to order the results in any way.

```
var cityQuery =
    (from city in cities.AsParallel()
     where city.Population > 10000
     select city).Take(1000);
```

```
Dim cityQuery = From city In cities.AsParallel()
                  Where city.Population > 10000
                  Take (1000)
```

This query does not necessarily produce the first 1000 cities in the source sequence that meet the condition, but rather some set of 1000 cities that meet the condition. PLINQ query operators partition the source sequence into multiple subsequences that are processed as concurrent tasks. If order preservation is not specified, the results from each partition are handed off to the next stage of the query in an arbitrary order. Also, a partition may yield a subset of its results before it continues to process the remaining elements. The resulting order may be different every time. Your application cannot control this because it depends on how the operating system schedules the threads.

The following example overrides the default behavior by using the [AsOrdered](#) operator on the source sequence. This ensures that the [Take](#) method returns the first 1000 cities in the source sequence that meet the condition.

```
var orderedCities =
    (from city in cities.AsParallel().AsOrdered()
     where city.Population > 10000
     select city).Take(1000);
```

```
Dim orderedCities = From city In cities.AsParallel().AsOrdered()
                      Where city.Population > 10000
                      Take (1000)
```

However, this query probably does not run as fast as the unordered version because it must keep track of the original ordering throughout the partitions and at merge time ensure that the ordering is consistent. Therefore, we recommend that you use [AsOrdered](#) only when it is required, and only for those parts of the query that

require it. When order preservation is no longer required, use [AsUnordered](#) to turn it off. The following example achieves this by composing two queries.

```
var orderedCities2 =
    (from city in cities.AsParallel().AsOrdered()
     where city.Population > 10000
     select city).Take(1000);

var finalResult =
    from city in orderedCities2.AsUnordered()
    join p in people.AsParallel()
    on city.Name equals p.CityName into details
    from c in details
    select new
    {
        city.Name,
        Pop = city.Population,
        c.Mayor
    };

foreach (var city in finalResult) { /*...*/ }
```

```
Dim orderedCities2 = From city In cities.AsParallel().AsOrdered()
                     Where city.Population > 10000
                     Select city
                     Take (1000)

Dim finalResult = From city In orderedCities2.AsUnordered()
                  Join p In people.AsParallel() On city.Name Equals p.CityName
                  Select New With {.Name = city.Name, .Pop = city.Population, .Mayor = city.Mayor}

For Each city In finalResult
    Console.WriteLine(city.Name & ":" & city.Pop & ":" & city.Mayor)
Next
```

Note that PLINQ preserves the ordering of a sequence produced by order-imposing operators for the rest of the query. In other words, operators such as [OrderBy](#) and [ThenBy](#) are treated as if they were followed by a call to [AsOrdered](#).

Query Operators and Ordering

The following query operators introduce order preservation into all subsequent operations in a query, or until [AsUnordered](#) is called:

- [OrderBy](#)
- [OrderByDescending](#)
- [ThenBy](#)
- [ThenByDescending](#)

The following PLINQ query operators may in some cases require ordered source sequences to produce correct results:

- [Reverse](#)
- [SequenceEqual](#)
- [TakeWhile](#)

- [SkipWhile](#)
- [Zip](#)

Some PLINQ query operators behave differently, depending on whether their source sequence is ordered or unordered. The following table lists these operators.

OPERATOR	RESULT WHEN THE SOURCE SEQUENCE IS ORDERED	RESULT WHEN THE SOURCE SEQUENCE IS UNORDERED
Aggregate	Nondeterministic output for nonassociative or noncommutative operations	Nondeterministic output for nonassociative or noncommutative operations
All	Not applicable	Not applicable
Any	Not applicable	Not applicable
AsEnumerable	Not applicable	Not applicable
Average	Nondeterministic output for nonassociative or noncommutative operations	Nondeterministic output for nonassociative or noncommutative operations
Cast	Ordered results	Unordered results
Concat	Ordered results	Unordered results
Count	Not applicable	Not applicable
DefaultIfEmpty	Not applicable	Not applicable
Distinct	Ordered results	Unordered results
ElementAt	Return specified element	Arbitrary element
ElementAtOrDefault	Return specified element	Arbitrary element
Except	Unordered results	Unordered results
First	Return specified element	Arbitrary element
FirstOrDefault	Return specified element	Arbitrary element
ForAll	Executes nondeterministically in parallel	Executes nondeterministically in parallel
GroupBy	Ordered results	Unordered results
GroupJoin	Ordered results	Unordered results
Intersect	Ordered results	Unordered results
Join	Ordered results	Unordered results

OPERATOR	RESULT WHEN THE SOURCE SEQUENCE IS ORDERED	RESULT WHEN THE SOURCE SEQUENCE IS UNORDERED
Last	Return specified element	Arbitrary element
LastOrDefault	Return specified element	Arbitrary element
LongCount	Not applicable	Not applicable
Min	Not applicable	Not applicable
OrderBy	Reorders the sequence	Starts new ordered section
OrderByDescending	Reorders the sequence	Starts new ordered section
Range	Not applicable (same default as AsParallel)	Not applicable
Repeat	Not applicable (same default as AsParallel)	Not applicable
Reverse	Reverses	Does nothing
Select	Ordered results	Unordered results
Select (indexed)	Ordered results	Unordered results.
SelectMany	Ordered results.	Unordered results
SelectMany (indexed)	Ordered results.	Unordered results.
SequenceEqual	Ordered comparison	Unordered comparison
Single	Not applicable	Not applicable
SingleOrDefault	Not applicable	Not applicable
Skip	Skips first n elements	Skips any n elements
SkipWhile	Ordered results.	Nondeterministic. Performs SkipWhile on the current arbitrary order
Sum	Nondeterministic output for nonassociative or noncommutative operations	Nondeterministic output for nonassociative or noncommutative operations
Take	Takes first n elements	Takes any n elements
TakeWhile	Ordered results	Nondeterministic. Performs TakeWhile on the current arbitrary order
ThenBy	Supplements OrderBy	Supplements OrderBy

OPERATOR	RESULT WHEN THE SOURCE SEQUENCE IS ORDERED	RESULT WHEN THE SOURCE SEQUENCE IS UNORDERED
ThenByDescending	Supplements <code>OrderBy</code>	Supplements <code>OrderBy</code>
ToArray	Ordered results	Unordered results
ToDictionary	Not applicable	Not applicable
ToList	Ordered results	Unordered results
ToLookup	Ordered results	Unordered results
Union	Ordered results	Unordered results
Where	Ordered results	Unordered results
Where (indexed)	Ordered results	Unordered results
Zip	Ordered results	Unordered results

Unordered results are not actively shuffled; they simply do not have any special ordering logic applied to them. In some cases, an unordered query may retain the ordering of the source sequence. For queries that use the indexed Select operator, PLINQ guarantees that the output elements will come out in the order of increasing indices, but makes no guarantees about which indices will be assigned to which elements.

See also

- [Parallel LINQ \(PLINQ\)](#)
- [Parallel Programming](#)

Merge Options in PLINQ

9/20/2022 • 3 minutes to read • [Edit Online](#)

When a query is executing as parallel, PLINQ partitions the source sequence so that multiple threads can work on different parts concurrently, typically on separate threads. If the results are to be consumed on one thread, for example, in a `foreach` (`For Each` in Visual Basic) loop, then the results from every thread must be merged back into one sequence. The kind of merge that PLINQ performs depends on the operators that are present in the query. For example, operators that impose a new order on the results must buffer all elements from all threads. From the perspective of the consuming thread (which is also that of the application user) a fully buffered query might run for a noticeable period of time before it produces its first result. Other operators, by default, are partially buffered; they yield their results in batches. One operator, `ForAll` is not buffered by default. It yields all elements from all threads immediately.

By using the `WithMergeOptions` method, as shown in the following example, you can provide a hint to PLINQ that indicates what kind of merging to perform.

```
var scanLines = from n in nums.AsParallel()
                .WithMergeOptions(ParallelMergeOptions.NotBuffered)
                where n % 2 == 0
                select ExpensiveFunc(n);
```

```
Dim scanlines = From n In nums.AsParallel().WithMergeOptions(ParallelMergeOptions.NotBuffered)
                 Where n Mod 2 = 0
                 Select ExpensiveFunc(n)
```

For the complete example, see [How to: Specify Merge Options in PLINQ](#).

If the particular query cannot support the requested option, then the option will just be ignored. In most cases, you do not have to specify a merge option for a PLINQ query. However, in some cases you may find by testing and measurement that a query executes best in a non-default mode. A common use of this option is to force a chunk-merging operator to stream its results in order to provide a more responsive user interface.

ParallelMergeOptions

The `ParallelMergeOptions` enumeration includes the following options that specify, for supported query shapes, how the final output of the query is yielded when the results are consumed on one thread:

- `Not Buffered`

The `NotBuffered` option causes each processed element to be returned from each thread as soon as it is produced. This behavior is analogous to "streaming" the output. If the `AsOrdered` operator is present in the query, `NotBuffered` preserves the order of the source elements. Although `NotBuffered` starts yielding results as soon as they're available, the total time to produce all the results might still be longer than using one of the other merge options.

- `Auto Buffered`

The `AutoBuffered` option causes the query to collect elements into a buffer and then periodically yield the buffer contents all at once to the consuming thread. This is analogous to yielding the source data in "chunks" instead of using the "streaming" behavior of `NotBuffered`. `AutoBuffered` may take longer than

`NotBuffered` to make the first element available on the consuming thread. The size of the buffer and the exact yielding behavior are not configurable and may vary, depending on various factors that relate to the query.

- `FullyBuffered`

The `FullyBuffered` option causes the output of the whole query to be buffered before any of the elements are yielded. When you use this option, it can take longer before the first element is available on the consuming thread, but the complete results might still be produced faster than by using the other options.

Query Operators that Support Merge Options

The following table lists the operators that support all merge option modes, subject to the specified restrictions.

OPERATOR	RESTRICTIONS
<code>AsEnumerable</code>	None
<code>Cast</code>	None
<code>Concat</code>	Non-ordered queries that have an Array or List source only.
<code>DefaultIfEmpty</code>	None
<code>OfType</code>	None
<code>Reverse</code>	Non-ordered queries that have an Array or List source only.
<code>Select</code>	None
<code>SelectMany</code>	None
<code>Skip</code>	None
<code>Take</code>	None
<code>Where</code>	None

All other PLINQ query operators might ignore user-provided merge options. Some query operators, for example, `Reverse` and `OrderBy`, cannot yield any elements until all have been produced and reordered. Therefore, when `ParallelMergeOptions` is used in a query that also contains an operator such as `Reverse`, the merge behavior will not be applied in the query until after that operator has produced its results.

The ability of some operators to handle merge options depends on the type of the source sequence, and whether the `AsOrdered` operator was used earlier in the query. `ForAll` is always `NotBuffered`; it yields its elements immediately. `OrderBy` is always `FullyBuffered`; it must sort the whole list before it yields.

See also

- [Parallel LINQ \(PLINQ\)](#)
- [How to: Specify Merge Options in PLINQ](#)

Potential pitfalls with PLINQ

9/20/2022 • 5 minutes to read • [Edit Online](#)

In many cases, PLINQ can provide significant performance improvements over sequential LINQ to Objects queries. However, the work of parallelizing the query execution introduces complexity that can lead to problems that, in sequential code, are not as common or are not encountered at all. This topic lists some practices to avoid when you write PLINQ queries.

Don't assume that parallel is always faster

Parallelization sometimes causes a PLINQ query to run slower than its LINQ to Objects equivalent. The basic rule of thumb is that queries that have few source elements and fast user delegates are unlikely to speedup much. However, because many factors are involved in performance, we recommend that you measure actual results before you decide whether to use PLINQ. For more information, see [Understanding Speedup in PLINQ](#).

Avoid writing to shared memory locations

In sequential code, it is not uncommon to read from or write to static variables or class fields. However, whenever multiple threads are accessing such variables concurrently, there is a big potential for race conditions. Even though you can use locks to synchronize access to the variable, the cost of synchronization can hurt performance. Therefore, we recommend that you avoid, or at least limit, access to shared state in a PLINQ query as much as possible.

Avoid over-parallelization

By using the `AsParallel` method, you incur the overhead costs of partitioning the source collection and synchronizing the worker threads. The benefits of parallelization are further limited by the number of processors on the computer. There is no speedup to be gained by running multiple compute-bound threads on just one processor. Therefore, you must be careful not to over-parallelize a query.

The most common scenario in which over-parallelization can occur is in nested queries, as shown in the following snippet.

```
var q = from cust in customers.AsParallel()
        from order in cust.Orders.AsParallel()
        where order.OrderDate > date
        select new { cust, order };
```

```
Dim q = From cust In customers.AsParallel()
        From order In cust.Orders.AsParallel()
        Where order.OrderDate > aDate
        Select New With {cust, order}
```

In this case, it is best to parallelize only the outer data source (`customers`) unless one or more of the following conditions apply:

- The inner data source (`cust.Orders`) is known to be very long.
- You are performing an expensive computation on each order. (The operation shown in the example is not expensive.)

- The target system is known to have enough processors to handle the number of threads that will be produced by parallelizing the query on `cust.Orders`.

In all cases, the best way to determine the optimum query shape is to test and measure. For more information, see [How to: Measure PLINQ Query Performance](#).

Avoid calls to non-thread-safe methods

Writing to non-thread-safe instance methods from a PLINQ query can lead to data corruption which may or may not go undetected in your program. It can also lead to exceptions. In the following example, multiple threads would be attempting to call the `Filestream.Write` method simultaneously, which is not supported by the class.

```
Dim fs As FileStream = File.OpenWrite(...)  
a.AsParallel().Where(...).OrderBy(...).Select(...).ForAll(Sub(x) fs.Write(x))
```

```
FileStream fs = File.OpenWrite(...);  
a.AsParallel().Where(...).OrderBy(...).Select(...).ForAll(x => fs.Write(x));
```

Limit calls to thread-safe methods

Most static methods in .NET are thread-safe and can be called from multiple threads concurrently. However, even in these cases, the synchronization involved can lead to significant slowdown in the query.

NOTE

You can test for this yourself by inserting some calls to `WriteLine` in your queries. Although this method is used in the documentation examples for demonstration purposes, do not use it in PLINQ queries.

Avoid unnecessary ordering operations

When PLINQ executes a query in parallel, it divides the source sequence into partitions that can be operated on concurrently on multiple threads. By default, the order in which the partitions are processed and the results are delivered is not predictable (except for operators such as `OrderBy`). You can instruct PLINQ to preserve the ordering of any source sequence, but this has a negative impact on performance. The best practice, whenever possible, is to structure queries so that they do not rely on order preservation. For more information, see [Order Preservation in PLINQ](#).

Prefer `ForAll` to `ForEach` when it is possible

Although PLINQ executes a query on multiple threads, if you consume the results in a `foreach` loop (`For Each` in Visual Basic), then the query results must be merged back into one thread and accessed serially by the enumerator. In some cases, this is unavoidable; however, whenever possible, use the `ForAll` method to enable each thread to output its own results, for example, by writing to a thread-safe collection such as [System.Collections.Concurrent.ConcurrentBag<T>](#).

The same issue applies to `Parallel.ForEach`. In other words, `source.AsParallel().Where().ForAll(...)` should be strongly preferred to `Parallel.ForEach(source.AsParallel().Where(), ...)`.

Be aware of thread affinity issues

Some technologies, for example, COM interoperability for Single-Threaded Apartment (STA) components,

Windows Forms, and Windows Presentation Foundation (WPF), impose thread affinity restrictions that require code to run on a specific thread. For example, in both Windows Forms and WPF, a control can only be accessed on the thread on which it was created. If you try to access the shared state of a Windows Forms control in a PLINQ query, an exception is raised if you are running in the debugger. (This setting can be turned off.) However, if your query is consumed on the UI thread, then you can access the control from the `foreach` loop that enumerates the query results because that code executes on just one thread.

Don't assume that iterations of `ForEach`, `For`, and `ForAll` always execute in parallel

It is important to keep in mind that individual iterations in a `Parallel.For`, `Parallel.ForEach`, or `ForAll` loop may but do not have to execute in parallel. Therefore, you should avoid writing any code that depends for correctness on parallel execution of iterations or on the execution of iterations in any particular order.

For example, this code is likely to deadlock:

```
Dim mre = New ManualResetEventSlim()
Enumerable.Range(0, Environment.ProcessorCount * 100).AsParallel().ForAll(Sub(j)
    If j = Environment.ProcessorCount Then
        Console.WriteLine("Set on {0} with value of {1}", Thread.CurrentThread.ManagedThreadId, j)
        mre.Set()
    Else
        Console.WriteLine("Waiting on {0} with value of {1}", Thread.CurrentThread.ManagedThreadId, j)
        mre.Wait()
    End If
End Sub) ' deadlocks
```

```
ManualResetEventSlim mre = new ManualResetEventSlim();
Enumerable.Range(0, Environment.ProcessorCount * 100).AsParallel().ForAll((j) =>
{
    if (j == Environment.ProcessorCount)
    {
        Console.WriteLine("Set on {0} with value of {1}", Thread.CurrentThread.ManagedThreadId, j);
        mre.Set();
    }
    else
    {
        Console.WriteLine("Waiting on {0} with value of {1}", Thread.CurrentThread.ManagedThreadId, j);
        mre.Wait();
    }
}); //deadlocks
```

In this example, one iteration sets an event, and all other iterations wait on the event. None of the waiting iterations can complete until the event-setting iteration has completed. However, it is possible that the waiting iterations block all threads that are used to execute the parallel loop, before the event-setting iteration has had a chance to execute. This results in a deadlock – the event-setting iteration will never execute, and the waiting iterations will never wake up.

In particular, one iteration of a parallel loop should never wait on another iteration of the loop to make progress. If the parallel loop decides to schedule the iterations sequentially but in the opposite order, a deadlock will occur.

See also

- [Parallel LINQ \(PLINQ\)](#)

How to: Create and Execute a Simple PLINQ Query

9/20/2022 • 2 minutes to read • [Edit Online](#)

The example in this article shows how to create a simple Parallel Language Integrated Query (LINQ) query by using the `ParallelEnumerable.AsParallel` extension method on the source sequence and executing the query by using the `ParallelEnumerable.ForAll` method.

NOTE

This documentation uses lambda expressions to define delegates in PLINQ. If you are not familiar with lambda expressions in C# or Visual Basic, see [Lambda Expressions in PLINQ and TPL](#).

Example

```
using System;
using System.Linq;

class ExampleForAll
{
    public static void Main()
    {
        var source = Enumerable.Range(100, 20000);

        // Result sequence might be out of order.
        var parallelQuery =
            from num in source.AsParallel()
            where num % 10 == 0
            select num;

        // Process result sequence in parallel
        parallelQuery.ForAll((e) => DoSomething(e));

        // Or use foreach to merge results first.
        foreach (var n in parallelQuery)
        {
            Console.WriteLine(n);
        }

        // You can also use ToArray, ToList, etc as with LINQ to Objects.
        var parallelQuery2 =
            (from num in source.AsParallel()
            where num % 10 == 0
            select num).ToArray();

        // Method syntax is also supported
        var parallelQuery3 =
            source.AsParallel()
                .Where(n => n % 10 == 0)
                .Select(n => n);

        Console.WriteLine("\nPress any key to exit...");
        Console.ReadLine();
    }

    static void DoSomething(int _) { }
}
```

```

Public Class Program
    Public Shared Sub Main()
        Dim source = Enumerable.Range(100, 20000)

        ' Result sequence might be out of order.
        Dim parallelQuery = From num In source.AsParallel()
                            Where num Mod 10 = 0
                            Select num

        ' Process result sequence in parallel
        parallelQuery.ForAll(Sub(e)
                            DoSomething(e)
                            End Sub)

        ' Or use For Each to merge results first
        ' as in this example, Where results must
        ' be serialized sequentially through static Console method.
        For Each n In parallelQuery
            Console.WriteLine("{0} ", n)
        Next

        ' You can also use ToArray,ToList, etc, as with LINQ to Objects.
        Dim parallelQuery2 = (From num In source.AsParallel()
                            Where num Mod 10 = 0
                            Select num).ToArray()

        ' Method syntax is also supported
        Dim parallelQuery3 =
            source.AsParallel().Where(Function(n)
                Return (n Mod 10) = 0
            End Function).Select(Function(n)
                Return n
            End Function)

        For Each i As Integer In parallelQuery3
            Console.WriteLine($"{i} ")
        Next

        Console.WriteLine()
        Console.WriteLine("Press any key to exit...")
        Console.ReadLine()
    End Sub

    ' A toy function to demonstrate syntax. Typically you need a more
    ' computationally expensive method to see speedup over sequential queries.
    Shared Sub DoSomething(ByVal i As Integer)
        Console.WriteLine($"{Math.Sqrt(i):###.##} ")
    End Sub
End Class

```

This example demonstrates the basic pattern for creating and executing any Parallel LINQ query when the ordering of the result sequence is not important. Unordered queries are generally faster than ordered queries. The query partitions the source into tasks that are executed asynchronously on multiple threads. The order in which each task completes depends not only on the amount of work involved to process the elements in the partition, but also on external factors such as how the operating system schedules each thread. This example is intended to demonstrate usage, and might not run faster than the equivalent sequential LINQ to Objects query. For more information about speedup, see [Understanding Speedup in PLINQ](#). For more information about how to preserve the ordering of elements in a query, see [How to: Control Ordering in a PLINQ Query](#).

See also

- Parallel LINQ (PLINQ)

How to: Control Ordering in a PLINQ Query

9/20/2022 • 3 minutes to read • [Edit Online](#)

These examples show how to control the ordering in a PLINQ query by using the `AsOrdered` extension method.

WARNING

These examples are primarily intended to demonstrate usage, and may or may not run faster than the equivalent sequential LINQ to Objects queries.

Example 1

The following example preserves the ordering of the source sequence. This is sometimes necessary; for example some query operators require an ordered source sequence to produce correct results.

```
var source = Enumerable.Range(9, 10000);

// Source is ordered; let's preserve it.
var parallelQuery =
    from num in source.AsParallel().AsOrdered()
    where num % 3 == 0
    select num;

// Use foreach to preserve order at execution time.
foreach (var item in parallelQuery)
{
    Console.WriteLine($"{item} ");
}

// Some operators expect an ordered source sequence.
var lowValues = parallelQuery.Take(10);
```

```
Sub OrderedQuery()

    Dim source = Enumerable.Range(9, 10000)

    ' Source is ordered let's preserve it.
    Dim parallelQuery = From num In source.AsParallel().AsOrdered()
        Where num Mod 3 = 0
        Select num

    ' Use For Each to preserve order at execution time.
    For Each item In parallelQuery
        Console.WriteLine("{0} ", item)
    Next

    ' Some operators expect an ordered source sequence.
    Dim lowValues = parallelQuery.Take(10)

End Sub
```

Example 2

The following example shows some query operators whose source sequence is probably expected to be

ordered. These operators will work on unordered sequences, but they might produce unexpected results.

```
// Paste into PLINQDataSample class.
static void SimpleOrdering()
{
    var customers = GetCustomers();

    // Take the first 20, preserving the original order
    var firstTwentyCustomers = customers
        .AsParallel()
        .AsOrdered()
        .Take(20);

    foreach (var c in firstTwentyCustomers)
        Console.WriteLine("{0} ", c.CustomerID);

    // All elements in reverse order.
    var reverseOrder = customers
        .AsParallel()
        .AsOrdered()
        .Reverse();

    foreach (var v in reverseOrder)
        Console.WriteLine("{0} ", v.CustomerID);

    // Get the element at a specified index.
    var cust = customers.AsParallel()
        .AsOrdered()
        .ElementAt(48);

    Console.WriteLine("Element #48 is: {0}", cust.CustomerID);
}
```

```

' Paste into PLINQDataSample class
Shared Sub SimpleOrdering()
    Dim customers As List(Of Customer) = GetCustomers().ToList()

    ' Take the first 20, preserving the original order

    Dim firstTwentyCustomers = customers _
        .AsParallel() _
        .AsOrdered() _
        .Take(20)

    Console.WriteLine("Take the first 20 in original order")
    For Each c As Customer In firstTwentyCustomers
        Console.Write(c.CustomerID & " ")
    Next

    ' All elements in reverse order.
    Dim reverseOrder = customers _
        .AsParallel() _
        .AsOrdered() _
        .Reverse()

    Console.WriteLine(vbCrLf & "Take all elements in reverse order")
    For Each c As Customer In reverseOrder
        Console.Write("{0} ", c.CustomerID)
    Next

    ' Get the element at a specified index.
    Dim cust = customers.AsParallel() _
        .AsOrdered() _
        .ElementAt(48)

    Console.WriteLine("Element #48 is: " & cust.CustomerID)

End Sub

```

To run this method, paste it into the `PLINQDataSample` class in the [PLINQ Data Sample](#) project and press F5.

Example 3

The following example shows how to preserve ordering for the first part of a query, then remove the ordering to increase the performance of a join clause, and then reapply ordering to the final result sequence.

```

// Paste into PLINQDataSample class.
static void OrderedThenUnordered()
{
    var orders = GetOrders();
    var orderDetails = GetOrderDetails();

    var q2 = orders.AsParallel()
        .Where(o => o.OrderDate < DateTime.Parse("07/04/1997"))
        .Select(o => o)
        .OrderBy(o => o.CustomerID) // Preserve original ordering for Take operation.
        .Take(20)
        .AsUnordered() // Remove ordering constraint to make join faster.
        .Join(
            orderDetails.AsParallel(),
            ord => ord.OrderID,
            od => od.OrderID,
            (ord, od) =>
            new
            {
                ID = ord.OrderID,
                Customer = ord.CustomerID,
                Product = od.ProductID
            }
        )
        .OrderBy(i => i.Product); // Apply new ordering to final result sequence.

    foreach (var v in q2)
        Console.WriteLine("{0} {1} {2}", v.ID, v.Customer, v.Product);
}

```

```

' Paste into PLINQDataSample class
Sub OrderedThenUnordered()
    Dim Orders As IEnumerable(Of Order) = GetOrders()
    Dim orderDetails As IEnumerable(Of OrderDetail) = GetOrderDetails()

    ' Sometimes it's easier to create a query
    ' by composing two subqueries
    Dim query1 = From ord In Orders.AsParallel()
        Where ord.OrderDate < DateTime.Parse("07/04/1997")
        Select ord
        Order By ord.CustomerID
        Take 20

    Dim query2 = From ord In query1.AsUnordered()
        Join od In orderDetails.AsParallel() On ord.OrderID Equals od.OrderID
        Order By od.ProductID
        Select New With {ord.OrderID, ord.CustomerID, od.ProductID}

    For Each item In query2
        Console.WriteLine("{0} {1} {2}", item.OrderID, item.CustomerID, item.ProductID)
    Next
End Sub

```

To run this method, paste it into the [PLINQ Data Sample](#) project and press F5.

See also

- [ParallelEnumerable](#)
- [Parallel LINQ \(PLINQ\)](#)

How to: Combine Parallel and Sequential LINQ Queries

9/20/2022 • 2 minutes to read • [Edit Online](#)

This example shows how to use the [AsSequential](#) method to instruct PLINQ to process all subsequent operators in the query sequentially. Although sequential processing is often slower than parallel, sometimes it's necessary to produce correct results.

NOTE

This example is intended to demonstrate usage and might not run faster than the equivalent sequential LINQ to Objects query. For more information about speedup, see [Understanding Speedup in PLINQ](#).

Example

The following example shows one scenario in which [AsSequential](#) is required, namely to preserve the ordering that was established in a previous clause of the query.

```
// Paste into PLINQDataSample class.
static void SequentialDemo()
{
    var orders = GetOrders();
    var query = (from order in orders.AsParallel()
        orderby order.OrderID
        select new
        {
            order.OrderID,
            OrderedOn = order.OrderDate,
            ShippedOn = order.ShippedDate
        })
        .AsSequential().Take(5);
}
```

```
' Paste into PLINQDataSample class
Shared Sub SequentialDemo()

    Dim orders = GetOrders()
    Dim query = From ord In orders.AsParallel()
        Order By ord.OrderID
        Select New With
        {
            ord.OrderID,
            ord.OrderDate,
            ord.ShippedDate
        }

    Dim query2 = query.AsSequential().Take(5)

    For Each item In query2
        Console.WriteLine("{0}, {1}, {2}", item.OrderDate, item.OrderID, item.ShippedDate)
    Next
End Sub
```

Compiling the Code

To compile and run this code, paste it into the [PLINQ Data Sample](#) project, add a line to call the method from `Main`, and press F5.

See also

- [Parallel LINQ \(PLINQ\)](#)

How to: Handle Exceptions in a PLINQ Query

9/20/2022 • 4 minutes to read • [Edit Online](#)

The first example in this topic shows how to handle the [System.AggregateException](#) that can be thrown from a PLINQ query when it executes. The second example shows how to put try-catch blocks within delegates, as close as possible to where the exception will be thrown. In this way, you can catch them as soon as they occur and possibly continue query execution. When exceptions are allowed to bubble up back to the joining thread, then it is possible that a query may continue to process some items after the exception is raised.

In some cases when PLINQ falls back to sequential execution, and an exception occurs, the exception may be propagated directly, and not wrapped in an [AggregateException](#). Also, [ThreadAbortExceptions](#) are always propagated directly.

NOTE

When "Just My Code" is enabled, Visual Studio will break on the line that throws the exception and display an error message that says "exception not handled by user code." This error is benign. You can press F5 to continue from it, and see the exception-handling behavior that is demonstrated in the examples below. To prevent Visual Studio from breaking on the first error, just uncheck the "Just My Code" checkbox under **Tools**, **Options**, **Debugging**, **General**.

This example is intended to demonstrate usage, and might not run faster than the equivalent sequential LINQ to Objects query. For more information about speedup, see [Understanding Speedup in PLINQ](#).

Example 1

This example shows how to put the try-catch blocks around the code that executes the query to catch any [System.AggregateExceptions](#) that are thrown.

```

// Paste into PLINQDataSample class.
static void PLINQExceptions_1()
{
    // Using the raw string array here. See PLINQ Data Sample.
    string[] customers = GetCustomersAsStrings().ToArray();

    // First, we must simulate some corrupt input.
    customers[54] = "###";

    var parallelQuery = from cust in customers.AsParallel()
        let fields = cust.Split(',')
        where fields[3].StartsWith("C") //throw indexoutofrange
        select new { city = fields[3], thread = Thread.CurrentThread.ManagedThreadId };

    try
    {
        // We use ForAll although it doesn't really improve performance
        // since all output is serialized through the Console.
        parallelQuery.ForAll(e => Console.WriteLine("City: {0}, Thread:{1}", e.city, e.thread));
    }

    // In this design, we stop query processing when the exception occurs.
    catch (AggregateException e)
    {
        foreach (var ex in e.InnerExceptions)
        {
            Console.WriteLine(ex.Message);
            if (ex is IndexOutOfRangeException)
                Console.WriteLine("The data source is corrupt. Query stopped.");
        }
    }
}

```

```

' Paste into PLINQDataSample class
Shared Sub PLINQExceptions_1()

    ' Using the raw string array here. See PLINQ Data Sample.
    Dim customers As String() = GetCustomersAsStrings().ToArray()

    ' First, we must simulate some corrupt input.
    customers(20) = "###"

    'throws indexoutofrange
    Dim query = From cust In customers.AsParallel()
        Let fields = cust.Split(",c")
        Where fields(3).StartsWith("C")
        Select fields

    Try
        ' We use ForAll although it doesn't really improve performance
        ' since all output is serialized through the Console.
        query.ForAll(Sub(e)
            Console.WriteLine("City: {0}, Thread:{1}")
        End Sub)
    Catch e As AggregateException

        ' In this design, we stop query processing when the exception occurs.
        For Each ex In e.InnerExceptions
            Console.WriteLine(ex.Message)
            If TypeOf ex Is IndexOutOfRangeException Then
                Console.WriteLine("The data source is corrupt. Query stopped.")
            End If
        Next
    End Try
End Sub

```

In this example, the query cannot continue after the exception is thrown. By the time your application code

catches the exception, PLINQ has already stopped the query on all threads.

Example 2

The following example shows how to put a try-catch block in a delegate to make it possible to catch an exception and continue with the query execution.

```
// Paste into PLINQDataSample class.
static void PLINQExceptions_2()
{
    var customers = GetCustomersAsStrings().ToArray();
    // Using the raw string array here.
    // First, we must simulate some corrupt input
    customers[54] = "###";

    // Assume that in this app, we expect malformed data
    // occasionally and by design we just report it and continue.
    static bool IsTrue(string[] f, string c)
    {
        try
        {
            string s = f[3];
            return s.StartsWith(c);
        }
        catch (IndexOutOfRangeException)
        {
            Console.WriteLine($"Malformed cust: {f}");
            return false;
        }
    };

    // Using the raw string array here
    var parallelQuery =
        from cust in customers.AsParallel()
        let fields = cust.Split(',')
        where IsTrue(fields, "C") //use a named delegate with a try-catch
        select new { City = fields[3] };

    try
    {
        // We use ForAll although it doesn't really improve performance
        // since all output must be serialized through the Console.
        parallelQuery.ForAll(e => Console.WriteLine(e.City));
    }

    // IndexOutOfRangeException will not bubble up
    // because we handle it where it is thrown.
    catch (AggregateException e)
    {
        foreach (var ex in e.InnerExceptions)
        {
            Console.WriteLine(ex.Message);
        }
    }
}
```

```

' Paste into PLINQDataSample class
Shared Sub PLINQExceptions_2()

    Dim customers() = GetCustomersAsStrings().ToArray()
    ' Using the raw string array here.
    ' First, we must simulate some corrupt input
    customers(20) = "###"

    ' Create a delegate with a lambda expression.
    ' Assume that in this app, we expect malformed data
    ' occasionally and by design we just report it and continue.
    Dim isTrue As Func(Of String(), String, Boolean) = Function(f, c)

        Try

            Dim s As String = f(3)
            Return s.StartsWith(c)

        Catch e As IndexOutOfRangeException

            Console.WriteLine("Malformed cust: {0}", f)
            Return False
        End Try
    End Function

    ' Using the raw string array here
    Dim query = From cust In customers.AsParallel()
        Let fields = cust.Split(",","c")
        Where isTrue(fields, "C")
        Select New With {.City = fields(3)}
    Try
        ' We use ForAll although it doesn't really improve performance
        ' since all output must be serialized through the Console.
        query.ForAll(Sub(e) Console.WriteLine(e.City))

        ' IndexOutOfRangeException will not bubble up
        ' because we handle it where it is thrown.
    Catch e As AggregateException
        For Each ex In e.InnerExceptions
            Console.WriteLine(ex.Message)
        Next
    End Try
End Sub

```

Compiling the Code

- To compile and run these examples, copy them into the PLINQ Data Sample example and call the method from Main.

Robust Programming

Do not catch an exception unless you know how to handle it so that you do not corrupt the state of your program.

See also

- [ParallelEnumerable](#)
- [Parallel LINQ \(PLINQ\)](#)

How to: Cancel a PLINQ Query

9/20/2022 • 6 minutes to read • [Edit Online](#)

The following examples show two ways to cancel a PLINQ query. The first example shows how to cancel a query that consists mostly of data traversal. The second example shows how to cancel a query that contains a user function that is computationally expensive.

NOTE

When "Just My Code" is enabled, Visual Studio will break on the line that throws the exception and display an error message that says "exception not handled by user code." This error is benign. You can press F5 to continue from it, and see the exception-handling behavior that is demonstrated in the examples below. To prevent Visual Studio from breaking on the first error, just uncheck the "Just My Code" checkbox under **Tools**, **Options**, **Debugging**, **General**.

This example is intended to demonstrate usage, and might not run faster than the equivalent sequential LINQ to Objects query. For more information about speedup, see [Understanding Speedup in PLINQ](#).

Example 1

```
namespace PLINQCancellation_1
{
    using System;
    using System.Linq;
    using System.Threading;
    using System.Threading.Tasks;
    using static System.Console;

    class Program
    {
        static void Main()
        {
            int[] source = Enumerable.Range(1, 10000000).ToArray();
            using CancellationTokenSource cts = new();

            // Start a new asynchronous task that will cancel the
            // operation from another thread. Typically you would call
            // Cancel() in response to a button click or some other
            // user interface event.
            Task.Factory.StartNew(() =>
            {
                UserClicksTheCancelButton(cts);
            });

            int[] results = null;
            try
            {
                results =
                    (from num in source.AsParallel().WithCancellation(cts.Token)
                     where num % 3 == 0
                     orderby num descending
                     select num).ToArray();
            }
            catch (OperationCanceledException e)
            {
                WriteLine(e.Message);
            }
            catch (AggregateException ae)
            {
```

```
        if (ae.InnerException != null)
        {
            foreach (Exception e in ae.InnerException)
            {
                WriteLine(e.Message);
            }
        }

        foreach (var item in results ?? Array.Empty<int>())
        {
            WriteLine(item);
        }
        WriteLine();
        ReadKey();
    }

    static void UserClicksTheCancelButton(CancellationTokenSource cts)
    {
        // Wait between 150 and 500 ms, then cancel.
        // Adjust these values if necessary to make
        // cancellation fire while query is still executing.
        Random rand = new();
        Thread.Sleep(rand.Next(150, 500));
        cts.Cancel();
    }
}
```

```

Class Program
    Private Shared Sub Main(ByVal args As String())
        Dim source As Integer() = Enumerable.Range(1, 10000000).ToArray()
        Dim cs As New CancellationTokenSource()

        ' Start a new asynchronous task that will cancel the
        ' operation from another thread. Typically you would call
        ' Cancel() in response to a button click or some other
        ' user interface event.
        Task.Factory.StartNew(Sub()
            UserClicksTheCancelButton(cs)
        End Sub)

        Dim results As Integer() = Nothing
        Try

            results = (From num In source.AsParallel().WithCancellation(cs.Token) _
                Where num Mod 3 = 0 _
                Order By num Descending _
                Select num).ToArray()
        Catch e As OperationCanceledException

            Console.WriteLine(e.Message)
        Catch ae As AggregateException

            If ae.InnerException IsNot Nothing Then
                For Each e As Exception In ae.InnerException
                    Console.WriteLine(e.Message)
                Next
            End If
        Finally
            cs.Dispose()
        End Try

        If results IsNot Nothing Then
            For Each item In results
                Console.WriteLine(item)
            Next
        End If
        Console.WriteLine()

        Console.ReadKey()
    End Sub

    Private Shared Sub UserClicksTheCancelButton(ByVal cs As CancellationTokenSource)
        ' Wait between 150 and 500 ms, then cancel.
        ' Adjust these values if necessary to make
        ' cancellation fire while query is still executing.
        Dim rand As New Random()
        Thread.Sleep(rand.Next(150, 350))
        cs.Cancel()
    End Sub
End Class

```

The PLINQ framework does not roll a single [OperationCanceledException](#) into an [System.AggregateException](#); the [OperationCanceledException](#) must be handled in a separate catch block. If one or more user delegates throws an [OperationCanceledException\(externalCT\)](#) (by using an external [System.Threading.CancellationToken](#)) but no other exception, and the query was defined as `AsParallel().WithCancellation(externalCT)`, then PLINQ will issue a single [OperationCanceledException](#) (externalCT) rather than an [System.AggregateException](#). However, if one user delegate throws an [OperationCanceledException](#), and another delegate throws another exception type, then both exceptions will be rolled into an [AggregateException](#).

The general guidance on cancellation is as follows:

1. If you perform user-delegate cancellation, you should inform PLINQ about the external [CancellationToken](#) and throw an [OperationCanceledException](#)(externalCT).
2. If cancellation occurs and no other exceptions are thrown, then handle an [OperationCanceledException](#) rather than an [AggregateException](#).

Example 2

The following example shows how to handle cancellation when you have a computationally expensive function in user code.

```
namespace PLINQCancellation_2
{
    using System;
    using System.Linq;
    using System.Threading;
    using System.Threading.Tasks;
    using static System.Console;

    class Program
    {
        static void Main(string[] args)
        {
            int[] source = Enumerable.Range(1, 10000000).ToArray();
            using CancellationTokenSource cts = new();

            // Start a new asynchronous task that will cancel the
            // operation from another thread. Typically you would call
            // Cancel() in response to a button click or some other
            // user interface event.
            Task.Factory.StartNew(() =>
            {
                UserClicksTheCancelButton(cts);
            });

            double[] results = null;
            try
            {
                results =
                    (from num in source.AsParallel().WithCancellation(cts.Token)
                     where num % 3 == 0
                     select Function(num, cts.Token)).ToArray();
            }
            catch (OperationCanceledException e)
            {
                WriteLine(e.Message);
            }
            catch (AggregateException ae)
            {
                if (ae.InnerException != null)
                {
                    foreach (Exception e in ae.InnerException)
                        WriteLine(e.Message);
                }
            }

            foreach (var item in results ?? Array.Empty<double>())
            {
                WriteLine(item);
            }
            WriteLine();
            ReadKey();
        }

        // A toy method to simulate work.
        static double Function(int n, CancellationToken ct)
```

```

    {
        // If work is expected to take longer than 1 ms
        // then try to check cancellation status more
        // often within that work.
        for (int i = 0; i < 5; i++)
        {
            // Work hard for approx 1 millisecond.
            Thread.Sleep(5000);

            // Check for cancellation request.
            ct.ThrowIfCancellationRequested();
        }
        // Anything will do for our purposes.
        return Math.Sqrt(n);
    }

    static void UserClicksTheCancelButton(CancellationTokenSource cts)
    {
        // Wait between 150 and 500 ms, then cancel.
        // Adjust these values if necessary to make
        // cancellation fire while query is still executing.
        Random rand = new();
        Thread.Sleep(rand.Next(150, 500));
        WriteLine("Press 'c' to cancel");
        if (ReadKey().KeyChar == 'c')
        {
            cts.Cancel();
        }
    }
}
}

```

```

Class Program2
Private Shared Sub Main(ByVal args As String())

    Dim source As Integer() = Enumerable.Range(1, 10000000).ToArray()
    Dim cs As New CancellationTokenSource()

    ' Start a new asynchronous task that will cancel the
    ' operation from another thread. Typically you would call
    ' Cancel() in response to a button click or some other
    ' user interface event.
    Task.Factory.StartNew(Sub()

        UserClicksTheCancelButton(cs)
    End Sub)

    Dim results As Double() = Nothing
    Try

        results = (From num In source.AsParallel().WithCancellation(cs.Token) _
                   Where num Mod 3 = 0 _
                   Select [Function](num, cs.Token)).ToArray()
    Catch e As OperationCanceledException

        Console.WriteLine(e.Message)
    Catch ae As AggregateException
        If ae.InnerExceptions IsNot Nothing Then
            For Each e As Exception In ae.InnerExceptions
                Console.WriteLine(e.Message)
            Next
        End If
    Finally
        cs.Dispose()
    End Try

```

```

If results IsNot Nothing Then
    For Each item In results
        Console.WriteLine(item)
    Next
End If
Console.WriteLine()

Console.ReadKey()
End Sub

' A toy method to simulate work.
Private Shared Function [Function](ByVal n As Integer, ByVal ct As CancellationToken) As Double
    ' If work is expected to take longer than 1 ms
    ' then try to check cancellation status more
    ' often within that work.
    For i As Integer = 0 To 4
        ' Work hard for approx 1 millisecond.
        Thread.SpinWait(50000)

        ' Check for cancellation request.
        If ct.IsCancellationRequested Then
            Throw New OperationCanceledException(ct)
        End If
    Next
    ' Anything will do for our purposes.
    Return Math.Sqrt(n)
End Function

Private Shared Sub UserClicksTheCancelButton(ByVal cs As CancellationTokenSource)
    ' Wait between 150 and 500 ms, then cancel.
    ' Adjust these values if necessary to make
    ' cancellation fire while query is still executing.
    Dim rand As New Random()
    Thread.Sleep(rand.[Next](150, 350))
    Console.WriteLine("Press 'c' to cancel")
    If Console.ReadKey().KeyChar = "c"c Then
        cs.Cancel()

    End If
End Sub
End Class

```

When you handle the cancellation in user code, you do not have to use [WithCancellation](#) in the query definition. However, we recommend that you do use [WithCancellation](#), because [WithCancellation](#) has no effect on query performance and it enables the cancellation to be handled by query operators and your user code.

In order to ensure system responsiveness, we recommend that you check for cancellation around once per millisecond; however, any period up to 10 milliseconds is considered acceptable. This frequency should not have a negative impact on your code's performance.

When an enumerator is disposed, for example when code breaks out of a foreach (For Each in Visual Basic) loop that is iterating over query results, then the query is canceled, but no exception is thrown.

See also

- [ParallelEnumerable](#)
- [Parallel LINQ \(PLINQ\)](#)
- [Cancellation in Managed Threads](#)

How to: Write a Custom PLINQ Aggregate Function

9/20/2022 • 2 minutes to read • [Edit Online](#)

This example shows how to use the [Aggregate](#) method to apply a custom aggregation function to a source sequence.

WARNING

This example is intended to demonstrate usage, and might not run faster than the equivalent sequential LINQ to Objects query. For more information about speedup, see [Understanding Speedup in PLINQ](#).

Example

The following example calculates the standard deviation of a sequence of integers.

```
namespace PLINQAggregation
{
    using System;
    using System.Linq;

    class aggregation
    {
        static void Main(string[] args)
        {

            // Create a data source for demonstration purposes.
            int[] source = new int[100000];
            Random rand = new Random();
            for (int x = 0; x < source.Length; x++)
            {
                // Should result in a mean of approximately 15.0.
                source[x] = rand.Next(10, 20);
            }

            // Standard deviation calculation requires that we first
            // calculate the mean average. Average is a predefined
            // aggregation operator, along with Max, Min and Count.
            double mean = source.AsParallel().Average();

            // We use the overload that is unique to ParallelEnumerable. The
            // third Func parameter combines the results from each thread.
            double standardDev = source.AsParallel().Aggregate(
                // initialize subtotal. Use decimal point to tell
                // the compiler this is a type double. Can also use: 0d.
                0.0,
                // do this on each thread
                (subtotal, item) => subtotal + Math.Pow((item - mean), 2),

                // aggregate results after all threads are done.
                (total, thisThread) => total + thisThread,

                // perform standard deviation calc on the aggregated result.
                (finalSum) => Math.Sqrt((finalSum / (source.Length - 1)))
            );
            Console.WriteLine("Mean value is = {0}", mean);
            Console.WriteLine("Standard deviation is {0}", standardDev);
            Console.ReadLine();
        }
    }
}
```

```

Class aggregation
    Private Shared Sub Main(ByVal args As String())

        ' Create a data source for demonstration purposes.
        Dim source As Integer() = New Integer(99999) {}
        Dim rand As New Random()
        For x As Integer = 0 To source.Length - 1
            ' Should result in a mean of approximately 15.0.
            source(x) = rand.Next(10, 20)
        Next

        ' Standard deviation calculation requires that we first
        ' calculate the mean average. Average is a predefined
        ' aggregation operator, along with Max, Min and Count.
        Dim mean As Double = source.AsParallel().Average()

        ' We use the overload that is unique to ParallelEnumerable. The
        ' third Func parameter combines the results from each thread.
        ' initialize subtotal. Use decimal point to tell
        ' the compiler this is a type double. Can also use: 0d.

        ' do this on each thread

        ' aggregate results after all threads are done.

        ' perform standard deviation calc on the aggregated result.
        Dim standardDev As Double = source.AsParallel().Aggregate(0.0R, Function(subtotal, item) subtotal +
            Math.Pow((item - mean), 2), Function(total, thisThread) total + thisThread, Function(finalSum)
            Math.Sqrt((finalSum / (source.Length - 1))))
        Console.WriteLine("Mean value is = {0}", mean)
        Console.WriteLine("Standard deviation is {0}", standardDev)

        Console.ReadLine()
    End Sub
End Class

```

This example uses an overload of the Aggregate standard query operator that is unique to PLINQ. This overload takes an extra [System.Func<T1,T2,TResult>](#) as the third input parameter. This delegate combines the results from all threads before it performs the final calculation on the aggregated results. In this example we add together the sums from all the threads.

Note that when a lambda expression body consists of a single expression, the return value of the [System.Func<T,TResult>](#) delegate is the value of the expression.

See also

- [ParallelEnumerable](#)
- [Parallel LINQ \(PLINQ\)](#)

How to: Specify the Execution Mode in PLINQ

9/20/2022 • 2 minutes to read • [Edit Online](#)

This example shows how to force PLINQ to bypass its default heuristics and parallelize a query regardless of the query's shape.

NOTE

This example is intended to demonstrate usage and might not run faster than the equivalent sequential LINQ to Objects query. For more information about speedup, see [Understanding Speedup in PLINQ](#).

Example

```
// Paste into PLINQDataSample class.  
static void ForceParallel()  
{  
    var customers = GetCustomers();  
    var parallelQuery = (from cust in customers.AsParallel()  
        .WithExecutionMode(ParallelExecutionMode.ForceParallelism)  
        where cust.City == "Berlin"  
        select cust.CustomerName)  
        .ToList();  
}
```

```
Private Shared Sub ForceParallel()  
    Dim customers = GetCustomers()  
    Dim parallelQuery = (From cust In  
customers.AsParallel().WithExecutionMode(ParallelExecutionMode.ForceParallelism)  
        Where cust.City = "Berlin"  
        Select cust.CustomerName).ToList()  
End Sub
```

PLINQ is designed to exploit opportunities for parallelization. However, not all queries benefit from parallel execution. For example, when a query contains a single user delegate that does little work, the query will usually run faster sequentially. Sequential execution is faster because the overhead involved in enabling parallelizing execution is more expensive than the speedup that's obtained. Therefore, PLINQ does not automatically parallelize every query. It first examines the shape of the query and the various operators that comprise it. Based on this analysis, PLINQ in the default execution mode may decide to execute some or all of the query sequentially. However, in some cases you may know more about your query than PLINQ is able to determine from its analysis. For example, you may know that a delegate is expensive and that the query will definitely benefit from parallelization. In such cases, you can use the [WithExecutionMode](#) method and specify the [ForceParallelism](#) value to instruct PLINQ to always run the query as parallel.

Compiling the Code

Cut and paste this code into the [PLINQ Data Sample](#) and call the method from `Main`.

See also

- [AsSequential](#)

- Parallel LINQ (PLINQ)

How to: Specify Merge Options in PLINQ

9/20/2022 • 2 minutes to read • [Edit Online](#)

This example shows how to specify the merge options that will apply to all subsequent operators in a PLINQ query. You do not have to set merge options explicitly, but doing so may improve performance. For more information about merge options, see [Merge Options in PLINQ](#).

WARNING

This example is intended to demonstrate usage, and might not run faster than the equivalent sequential LINQ to Objects query. For more information about speedup, see [Understanding Speedup in PLINQ](#).

Example

The following example demonstrates the behavior of merge options in a basic scenario that has an unordered source and applies an expensive function to every element.

```
namespace MergeOptions
{
    using System;
    using System.Diagnostics;
    using System.Linq;
    using System.Threading;

    class Program
    {
        static void Main(string[] args)
        {

            var nums = Enumerable.Range(1, 10000);

            // Replace NotBuffered with AutoBuffered
            // or FullyBuffered to compare behavior.
            var scanLines = from n in nums.AsParallel()
                            .WithMergeOptions(ParallelMergeOptions.NotBuffered)
                            where n % 2 == 0
                            select ExpensiveFunc(n);

            Stopwatch sw = Stopwatch.StartNew();
            foreach (var line in scanLines)
            {
                Console.WriteLine(line);
            }

            Console.WriteLine("Elapsed time: {0} ms. Press any key to exit.",
                sw.ElapsedMilliseconds);
            Console.ReadKey();
        }

        // A function that demonstrates what a fly
        // sees when it watches television :-
        static string ExpensiveFunc(int i)
        {
            Thread.Sleep(2000000);
            return string.Format("{0} *****", i);
        }
    }
}
```

```

Class MergeOptions2
    Sub DoMergeOptions()

        Dim nums = Enumerable.Range(1, 10000)

        ' Replace NotBuffered with AutoBuffered
        ' or FullyBuffered to compare behavior.
        Dim scanLines = From n In nums.AsParallel().WithMergeOptions(ParallelMergeOptions.NotBuffered)
                        Where n Mod 2 = 0
                        Select ExpensiveFunc(n)

        Dim sw = Stopwatch.StartNew()
        For Each line In scanLines
            Console.WriteLine(line)
        Next

        Console.WriteLine("Elapsed time: {0} ms. Press any key to exit.")
        Console.ReadKey()

    End Sub
    ' A function that demonstrates what a fly
    ' sees when it watches television :-)
    Function ExpensiveFunc(ByVal i As Integer) As String
        Threading.Thread.Sleep(2000000)
        Return String.Format("{0} *****", i)
    End Function
End Class

```

In cases where the [AutoBuffered](#) option incurs an undesirable latency before the first element is yielded, try the [NotBuffered](#) option to yield result elements faster and more smoothly.

See also

- [ParallelMergeOptions](#)
- [Parallel LINQ \(PLINQ\)](#)

How to: Iterate File Directories with PLINQ

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article shows two ways to parallelize operations on file directories. The first query uses the [GetFiles](#) method to populate an array of file names in a directory and all subdirectories. This method can introduce latency at the beginning of the operation, because it doesn't return until the entire array is populated. However, after the array is populated, PLINQ can process it in parallel quickly.

The second query uses the static [EnumerateDirectories](#) and [EnumerateFiles](#) methods, which begin returning results immediately. This approach can be faster when you're iterating over large directory trees, but the processing time compared to the first example depends on many factors.

NOTE

These examples are intended to demonstrate usage and might not run faster than the equivalent sequential LINQ to Objects query. For more information about speedup, see [Understanding Speedup in PLINQ](#).

GetFiles example

This example shows how to iterate over file directories in simple scenarios when you have access to all directories in the tree, the file sizes aren't large, and the access times are not significant. This approach involves a period of latency at the beginning while the array of file names is being constructed.

```

// Use Directory.GetFiles to get the source sequence of file names.
public static void FileIterationOne(string path)
{
    var sw = Stopwatch.StartNew();
    int count = 0;
    string[] files = null;
    try
    {
        files = Directory.GetFiles(path, "*.*", SearchOption.AllDirectories);
    }
    catch (UnauthorizedAccessException)
    {
        Console.WriteLine("You do not have permission to access one or more folders in this directory tree.");
        return;
    }
    catch (FileNotFoundException)
    {
        Console.WriteLine($"The specified directory {path} was not found.");
    }

    var fileContents =
        from FileName in files.AsParallel()
        let extension = Path.GetExtension(FileName)
        where extension == ".txt" || extension == ".htm"
        let Text = File.ReadAllText(FileName)
        select new
    {
        Text,
        FileName
    };

    try
    {
        foreach (var item in fileContents)
        {
            Console.WriteLine($"{Path.GetFileName(item.FileName)}:{item.Text.Length}");
            count++;
        }
    }
    catch (AggregateException ae)
    {
        ae.Handle(ex =>
    {
        if (ex is UnauthorizedAccessException uae)
        {
            Console.WriteLine(uae.Message);
            return true;
        }
        return false;
    });
    }

    Console.WriteLine($"FileIterationOne processed {count} files in {sw.ElapsedMilliseconds} milliseconds");
}

```

EnumerateFiles example

This example shows how to iterate over file directories in simple scenarios when you have access to all directories in the tree, the file sizes aren't large, and the access times are not significant. This approach begins producing results faster than the previous example.

```

public static void FileIterationTwo(string path) //225512 ms
{
    var count = 0;
    var sw = Stopwatch.StartNew();
    var fileNames =
        from dir in Directory.EnumerateFiles(path, "*.*", SearchOption.AllDirectories)
        select dir;

    var fileContents =
        from FileName in fileNames.AsParallel()
        let extension = Path.GetExtension(FileName)
        where extension == ".txt" || extension == ".htm"
        let Text = File.ReadAllText(FileName)
        select new
        {
            Text,
            FileName
        };
    try
    {
        foreach (var item in fileContents)
        {
            Console.WriteLine($"{Path.GetFileName(item.FileName)}:{item.Text.Length}");
            count++;
        }
    }
    catch (AggregateException ae)
    {
        ae.Handle(ex =>
        {
            if (ex is UnauthorizedAccessException uae)
            {
                Console.WriteLine(uae.Message);
                return true;
            }
            return false;
        });
    }
}

Console.WriteLine($"FileIterationTwo processed {count} files in {sw.ElapsedMilliseconds} milliseconds");
}

```

When using [GetFiles](#), be sure that you have sufficient permissions on all directories in the tree. Otherwise, an exception will be thrown and no results will be returned. When using the [EnumerateDirectories](#) in a PLINQ query, it is problematic to handle I/O exceptions in a graceful way that enables you to continue iterating. If your code must handle I/O or unauthorized access exceptions, then you should consider the approach described in [How to: Iterate File Directories with the Parallel Class](#).

If I/O latency is an issue, for example with file I/O over a network, consider using one of the asynchronous I/O techniques described in [TPL and Traditional .NET Asynchronous Programming](#) and in this [blog post](#).

See also

- [Parallel LINQ \(PLINQ\)](#)

How to: Measure PLINQ Query Performance

9/20/2022 • 2 minutes to read • [Edit Online](#)

This example shows how to use the [Stopwatch](#) class to measure the time it takes for a PLINQ query to execute.

Example

This example uses an empty `foreach` loop (`For Each` in Visual Basic) to measure the time it takes for the query to execute. In real-world code, the loop typically contains additional processing steps that add to the total query execution time. Notice that the stopwatch is not started until just before the loop, because that's when the query execution begins. If you require more fine-grained measurement, you can use the `ElapsedTicks` property instead of `ElapsedMilliseconds`.

```
using System;
using System.Diagnostics;
using System.Linq;

class ExampleMeasure
{
    static void Main()
    {
        var source = Enumerable.Range(0, 3000000);

        var queryToMeasure =
            from num in source.AsParallel()
            where num % 3 == 0
            select Math.Sqrt(num);

        Console.WriteLine("Measuring...");

        // The query does not run until it is enumerated.
        // Therefore, start the timer here.
        var sw = Stopwatch.StartNew();

        // For pure query cost, enumerate and do nothing else.
        foreach (var n in queryToMeasure) { }

        sw.Stop();
        long elapsed = sw.ElapsedMilliseconds; // or sw.ElapsedTicks
        Console.WriteLine("Total query time: {0} ms", elapsed);

        Console.WriteLine("Press any key to exit.");
        Console.ReadKey();
    }
}
```

```

Module ExampleMeasure
Sub Main()
    Dim source = Enumerable.Range(0, 3000000)
    ' Define parallel and non-parallel queries.
    Dim queryToMeasure = From num In source.AsParallel()
        Where num Mod 3 = 0
        Select Math.Sqrt(num)

    Console.WriteLine("Measuring...")

    ' The query does not run until it is enumerated.
    ' Therefore, start the timer here.
    Dim sw = Stopwatch.StartNew()

    ' For pure query cost, enumerate and do nothing else.
    For Each n As Double In queryToMeasure
    Next

    sw.Stop()
    Dim elapsed As Long = sw.ElapsedMilliseconds ' or sw.ElapsedTicks
    Console.WriteLine($"Total query time: {elapsed} ms.")

    Console.WriteLine("Press any key to exit.")
    Console.ReadKey()
End Sub
End Module

```

The total execution time is a useful metric when you are experimenting with query implementations, but it doesn't always tell the whole story. To get a deeper and richer view of the interaction of the query threads with one another and with other running processes, use the [Concurrency Visualizer](#).

See also

- [Parallel LINQ \(PLINQ\)](#)

PLINQ Data Sample

9/20/2022 • 19 minutes to read • [Edit Online](#)

This sample contains example data in .csv format, together with methods that transform it into in-memory collections of Customers, Products, Orders, and Order Details. To further experiment with PLINQ, you can paste code examples from certain other topics into the code in this topic and invoke it from the `Main` method. You can also use this data with your own PLINQ queries.

The data represents a subset of the Northwind database. Fifty (50) customer records are included, but not all fields. A subset of the rows from the Orders and corresponding Order_Detail data for every Customer is included. All Products are included.

NOTE

The data set is not large enough to demonstrate that PLINQ is faster than LINQ to Objects for queries that contain just basic `where` and `select` clauses. To observe speed increases for small data sets such as this, use queries that contain computationally expensive operations on every element in the data set.

To set up this sample

1. Create a Visual Basic or Visual C# console application project.
2. Replace the contents of Module1.vb or Program.cs by using the code that follows these steps.
3. On the **Project** menu, click **Add New Item**. Select **Text File** and then click **OK**. Copy the data in this topic and then paste it in the new text file. On the **File** menu, click **Save**, name the file Plinqdata.csv, and then save it in the folder that contains your source code files.
4. Press F5 to verify that the project builds and runs correctly. The following output should be displayed in the console window.

```
Customer count: 50
Product count: 77
Order count: 190
Order Details count: 483
Press any key to exit.
```

```
// This class contains a subset of data from the Northwind database
// in the form of string arrays. The methods such as GetCustomers, GetOrders, and so on
// transform the strings into object arrays that you can query in an object-oriented way.
// Many of the code examples in the PLINQ How-to topics are designed to be pasted into
// the PLINQDataSample class and invoked from the Main method.
partial class PLINQDataSample
{

    public static void Main()
    {
        //Call methods here.
        TestDataSource();

        Console.WriteLine("Press any key to exit.");
        Console.ReadKey();
    }

    static void TestDataSource()
```

```

    {
        Console.WriteLine("Customer count: {0}", GetCustomers().Count());
        Console.WriteLine("Product count: {0}", GetProducts().Count());
        Console.WriteLine("Order count: {0}", GetOrders().Count());
        Console.WriteLine("Order Details count: {0}", GetOrderDetails().Count());
    }

#region DataClasses
public class Order
{
    private Lazy<OrderDetail[]> _orderDetails;
    public Order()
    {
        _orderDetails = new Lazy<OrderDetail[]>(() => GetOrderDetailsForOrder(OrderID));
    }
    public int OrderID { get; set; }
    public string CustomerID { get; set; }
    public DateTime OrderDate { get; set; }
    public DateTime ShippedDate { get; set; }
    public OrderDetail[] OrderDetails { get { return _orderDetails.Value; } }
}

public class Customer
{
    private Lazy<Order[]> _orders;
    public Customer()
    {
        _orders = new Lazy<Order[]>(() => GetOrdersForCustomer(CustomerID));
    }
    public string CustomerID { get; set; }
    public string CustomerName { get; set; }
    public string Address { get; set; }
    public string City { get; set; }
    public string PostalCode { get; set; }
    public Order[] Orders
    {
        get
        {
            return _orders.Value;
        }
    }
}

public class Product
{
    public string ProductName { get; set; }
    public int ProductID { get; set; }
    public double UnitPrice { get; set; }
}

public class OrderDetail
{
    public int OrderID { get; set; }
    public int ProductID { get; set; }
    public double UnitPrice { get; set; }
    public double Quantity { get; set; }
    public double Discount { get; set; }
}
#endregion

public static IEnumerable<string> GetCustomersAsStrings()
{
    return System.IO.File.ReadAllLines(@"..\..\plinqdata.csv")
        .SkipWhile((line) => line.StartsWith("CUSTOMERS") == false)
        .Skip(1)
        .TakeWhile((line) => line.StartsWith("END CUSTOMERS") == false);
}

public static IEnumerable<Customer> GetCustomers()

```

```

{
    var customers = System.IO.File.ReadAllLines(@"..\..\plinqdata.csv")
        .SkipWhile((line) => line.StartsWith("CUSTOMERS") == false)
        .Skip(1)
        .TakeWhile((line) => line.StartsWith("END CUSTOMERS") == false);
    return (from line in customers
        let fields = line.Split(',')
        let custID = fields[0].Trim()
        select new Customer()
    {
        CustomerID = custID,
        CustomerName = fields[1].Trim(),
        Address = fields[2].Trim(),
        City = fields[3].Trim(),
        PostalCode = fields[4].Trim()
    });
}

public static Order[] GetOrdersForCustomer(string id)
{
    // Assumes we copied the file correctly!
    var orders = System.IO.File.ReadAllLines(@"..\..\plinqdata.csv")
        .SkipWhile((line) => line.StartsWith("ORDERS") == false)
        .Skip(1)
        .TakeWhile((line) => line.StartsWith("END ORDERS") == false);
    var orderStrings = from line in orders
        let fields = line.Split(',')
        where fields[1].CompareTo(id) == 0
        select new Order()
    {
        OrderID = Convert.ToInt32(fields[0]),
        CustomerID = fields[1].Trim(),
        OrderDate = DateTime.Parse(fields[2]),
        ShippedDate = DateTime.Parse(fields[3])
    };
    return orderStrings.ToArray();
}

// "10248, VINET, 7/4/1996 12:00:00 AM, 7/16/1996 12:00:00 AM
public static IEnumerable<Order> GetOrders()
{
    // Assumes we copied the file correctly!
    var orders = System.IO.File.ReadAllLines(@"..\..\plinqdata.csv")
        .SkipWhile((line) => line.StartsWith("ORDERS") == false)
        .Skip(1)
        .TakeWhile((line) => line.StartsWith("END ORDERS") == false);
    return from line in orders
        let fields = line.Split(',')
        select new Order()
    {
        OrderID = Convert.ToInt32(fields[0]),
        CustomerID = fields[1].Trim(),
        OrderDate = DateTime.Parse(fields[2]),
        ShippedDate = DateTime.Parse(fields[3])
    };
}

public static IEnumerable<Product> GetProducts()
{
    // Assumes we copied the file correctly!
    var products = System.IO.File.ReadAllLines(@"..\..\plinqdata.csv")
        .SkipWhile((line) => line.StartsWith("PRODUCTS") == false)
        .Skip(1)
        .TakeWhile((line) => line.StartsWith("END PRODUCTS") == false);
    return from line in products
        let fields = line.Split(',')
        select new Product()
    {

```

```

        ProductID = Convert.ToInt32(fields[0]),
        ProductName = fields[1].Trim(),
        UnitPrice = Convert.ToDouble(fields[2])
    );
}

public static IEnumerable<OrderDetail> GetOrderDetails()
{
    // Assumes we copied the file correctly!
    var orderDetails = System.IO.File.ReadAllLines(@"..\..\plinqdata.csv")
        .SkipWhile((line) => line.StartsWith("ORDER DETAILS") == false)
        .Skip(1)
        .TakeWhile((line) => line.StartsWith("END ORDER DETAILS") == false);

    return from line in orderDetails
        let fields = line.Split(',')
        select new OrderDetail()
    {
        OrderID = Convert.ToInt32(fields[0]),
        ProductID = Convert.ToInt32(fields[1]),
        UnitPrice = Convert.ToDouble(fields[2]),
        Quantity = Convert.ToDouble(fields[3]),
        Discount = Convert.ToDouble(fields[4])
    };
}

public static OrderDetail[] GetOrderDetailsForOrder(int id)
{
    // Assumes we copied the file correctly!
    var orderDetails = System.IO.File.ReadAllLines(@"..\..\plinqdata.csv")
        .SkipWhile((line) => line.StartsWith("ORDER DETAILS") == false)
        .Skip(1)
        .TakeWhile((line) => line.StartsWith("END ORDER DETAILS") == false);

    var orderDetailStrings = from line in orderDetails
        let fields = line.Split(',')
        let ordID = Convert.ToInt32(fields[0])
        where ordID == id
        select new OrderDetail()
    {
        OrderID = ordID,
        ProductID = Convert.ToInt32(fields[1]),
        UnitPrice = Convert.ToDouble(fields[2]),
        Quantity = Convert.ToDouble(fields[3]),
        Discount = Convert.ToDouble(fields[4])
    };
}

return orderDetailStrings.ToArray();
}
}

```

```

' This class contains a subset of data from the Northwind database
' in the form of string arrays. The methods such as GetCustomers, GetOrders, and so on
' transform the strings into object arrays that you can query in an object-oriented way.
' Many of the code examples in the PLINQ How-to topics are designed to be pasted into
' the PLINQDataSample class and invoked from the Main method.
' IMPORTANT: This data set is not large enough for meaningful comparisons of PLINQ vs. LINQ performance.
Class PLINQDataSample
    Shared Sub Main(ByVal args As String())

        'Call methods here.
        TestDataSource()

        Console.WriteLine("Press any key to exit.")
        Console.ReadKey()
    End Sub

```

```

Shared Sub TestDataSource()
    Console.WriteLine("Customer count: {0}", GetCustomers().Count())
    Console.WriteLine("Product count: {0}", GetProducts().Count())
    Console.WriteLine("Order count: {0}", GetOrders().Count())
    Console.WriteLine("Order Details count: {0}", GetOrderDetails().Count())
End Sub

#Region "DataClasses"
Class Order
    Public Sub New()
        _OrderDetails = New Lazy(Of OrderDetail())(Function() GetOrderDetailsForOrder(OrderID))
    End Sub
    Private _OrderID As Integer
    Public Property OrderID() As Integer
        Get
            Return _OrderID
        End Get
        Set(ByVal value As Integer)
            _OrderID = value
        End Set
    End Property
    Private _CustomerID As String
    Public Property CustomerID() As String
        Get
            Return _CustomerID
        End Get
        Set(ByVal value As String)
            _CustomerID = value
        End Set
    End Property
    Private _OrderDate As DateTime
    Public Property OrderDate() As DateTime
        Get
            Return _OrderDate
        End Get
        Set(ByVal value As DateTime)
            _OrderDate = value
        End Set
    End Property
    Private _ShippedDate As DateTime
    Public Property ShippedDate() As DateTime
        Get
            Return _ShippedDate
        End Get
        Set(ByVal value As DateTime)
            _ShippedDate = value
        End Set
    End Property
    Private _OrderDetails As Lazy(Of OrderDetail())
    Public ReadOnly Property OrderDetails As OrderDetail()
        Get
            Return _OrderDetails.Value
        End Get
    End Property
End Class

Class Customer

    Private _Orders As Lazy(Of Order())
    Public Sub New()
        _Orders = New Lazy(Of Order())(Function() GetOrdersForCustomer(_CustomerID))
    End Sub

    Private _CustomerID As String
    Public Property CustomerID() As String
        Get
            Return _CustomerID
        End Get
        Set(ByVal value As String)

```

```

        _CustomerID = value
    End Set
End Property
Private _CustomerName As String
Public Property CustomerName() As String
    Get
        Return _CustomerName
    End Get
    Set(ByVal value As String)
        _CustomerName = value
    End Set
End Property
Private _Address As String
Public Property Address() As String
    Get
        Return _Address
    End Get
    Set(ByVal value As String)
        _Address = value
    End Set
End Property
Private _City As String
Public Property City() As String
    Get
        Return _City
    End Get
    Set(ByVal value As String)
        _City = value
    End Set
End Property
Private _PostalCode As String
Public Property PostalCode() As String
    Get
        Return _PostalCode
    End Get
    Set(ByVal value As String)
        _PostalCode = value
    End Set
End Property
Public ReadOnly Property Orders() As Order()
    Get
        Return _Orders.Value
    End Get
End Property
End Class

Class Product
    Private _ProductName As String
    Public Property ProductName() As String
        Get
            Return _ProductName
        End Get
        Set(ByVal value As String)
            _ProductName = value
        End Set
    End Property
    Private _ProductID As Integer
    Public Property ProductID() As Integer
        Get
            Return _ProductID
        End Get
        Set(ByVal value As Integer)
            _ProductID = value
        End Set
    End Property
    Private _UnitPrice As Double
    Public Property UnitPrice() As Double
        Get
            Return _UnitPrice
        End Get
    End Property
End Class

```

```

        End Get
        Set(ByVal value As Double)
            _UnitPrice = value
        End Set
    End Property
End Class

Class OrderDetail
    Private _OrderID As Integer
    Public Property OrderID() As Integer
        Get
            Return _OrderID
        End Get
        Set(ByVal value As Integer)
            _OrderID = value
        End Set
    End Property
    Private _ProductID As Integer
    Public Property ProductID() As Integer
        Get
            Return _ProductID
        End Get
        Set(ByVal value As Integer)
            _ProductID = value
        End Set
    End Property
    Private _UnitPrice As Double
    Public Property UnitPrice() As Double
        Get
            Return _UnitPrice
        End Get
        Set(ByVal value As Double)
            _UnitPrice = value
        End Set
    End Property
    Private _Quantity As Double
    Public Property Quantity() As Double
        Get
            Return _Quantity
        End Get
        Set(ByVal value As Double)
            _Quantity = value
        End Set
    End Property
    Private _Discount As Double
    Public Property Discount() As Double
        Get
            Return _Discount
        End Get
        Set(ByVal value As Double)
            _Discount = value
        End Set
    End Property
End Class
#End Region

Shared Function GetCustomersAsStrings() As IEnumerable(Of String)

    Return IO.File.ReadAllLines("../..\plinqdata.csv").
        SkipWhile(Function(line) line.StartsWith("CUSTOMERS") = False).
        Skip(1).
        TakeWhile(Function(line) line.StartsWith("END CUSTOMERS") = False)
End Function

Shared Function GetCustomers() As IEnumerable(Of Customer)

    Dim customers = IO.File.ReadAllLines("../..\plinqdata.csv").
        SkipWhile(Function(line) line.StartsWith("CUSTOMERS") = False).

```

```

Skip(1).
TakeWhile(Function(line) line.StartsWith("END CUSTOMERS") = False)

Return From line In customers
Let fields = line.Split(",","c")
Select New Customer With {
    .CustomerID = fields(0).Trim(),
    .CustomerName = fields(1).Trim(),
    .Address = fields(2).Trim(),
    .City = fields(3).Trim(),
    .PostalCode = fields(4).Trim()
}
End Function

Shared Function GetOrders() As IEnumerable(Of Order)

Dim orders = IO.File.ReadAllLines("../..\plinqdata.csv").
    SkipWhile(Function(line) line.StartsWith("ORDERS") = False).
    Skip(1).
    TakeWhile(Function(line) line.StartsWith("END ORDERS") = False)

Return From line In orders
Let fields = line.Split(",","c")
Select New Order With {
    .OrderID = CType(fields(0).Trim(), Integer),
    .CustomerID = fields(1).Trim(),
    .OrderDate = DateTime.Parse(fields(2)),
    .ShippedDate = DateTime.Parse(fields(3))}
}
End Function

Shared Function GetOrdersForCustomer(ByVal id As String) As Order()

Dim orders = IO.File.ReadAllLines("../..\plinqdata.csv").
    SkipWhile(Function(line) line.StartsWith("ORDERS") = False).
    Skip(1).
    TakeWhile(Function(line) line.StartsWith("END ORDERS") = False)

Dim orderStrings = From line In orders
Let fields = line.Split(",","c")
Let custID = fields(1).Trim()
Where custID = id
Select New Order With {
    .OrderID = CType(fields(0).Trim(), Integer),
    .CustomerID = custID,
    .OrderDate = DateTime.Parse(fields(2)),
    .ShippedDate = DateTime.Parse(fields(3))}

Return orderStrings.ToArray()
End Function

Shared Function GetProducts() As IEnumerable(Of Product)

Dim products = IO.File.ReadAllLines("../..\plinqdata.csv").
    SkipWhile(Function(line) line.StartsWith("PRODUCTS") = False).
    Skip(1).
    TakeWhile(Function(line) line.StartsWith("END PRODUCTS") = False)

Return From line In products
Let fields = line.Split(",","c")
Select New Product With {
    .ProductID = CType(fields(0), Integer),
    .ProductName = fields(1).Trim(),
    .UnitPrice = CType(fields(2), Double)}
}
End Function

Shared Function GetOrderDetailsForOrder(ByVal orderID As Integer) As OrderDetail()
Dim orderDetails = IO.File.ReadAllLines("../..\plinqdata.csv").
    SkipWhile(Function(line) line.StartsWith("ORDER DETAILS") = False).
    Skip(1)

```

```

    TakeWhile(Function(line) line.StartsWith("END ORDER DETAILS") = False)

    Dim ordDetailStrings = From line In orderDetails
        Let fields = line.Split(",c")
        Let ordID = Convert.ToInt32(fields(0))
        Where ordID = orderID
        Select New OrderDetail With {
            .OrderID = ordID,
            .ProductID = CType(fields(1), Integer),
            .UnitPrice = CType(fields(2), Double),
            .Quantity = CType(fields(3), Double),
            .Discount = CType(fields(4), Double)}
    Return ordDetailStrings.ToArray()

End Function

Shared Function GetOrderDetails() As IEnumerable(Of OrderDetail)
    Dim orderDetails = IO.File.ReadAllLines("../..\\plinqdata.csv").
        SkipWhile(Function(line) line.StartsWith("ORDER DETAILS") = False).
        Skip(1).
        TakeWhile(Function(line) line.StartsWith("END ORDER DETAILS") = False)

    Return From line In orderDetails
        Let fields = line.Split(",c")
        Select New OrderDetail With {
            .OrderID = CType(fields(0), Integer),
            .ProductID = CType(fields(1), Integer),
            .UnitPrice = CType(fields(2), Double),
            .Quantity = CType(fields(3), Double),
            .Discount = CType(fields(4), Double)}
End Function

End Class

```

Data

CUSTOMERS
ALFKI,Alfreds Futterkiste,Obere Str. 57,Berlin,12209
ANATR,Aña Trujillo Emparedados y helados,Avda. de la Constitución 2222,México D.F.,05021
ANTON,Antonio Moreno Taquería,Mataderos 2312,México D.F.,05023
AROUT,Around the Horn,120 Hanover Sq.,London,WA1 1DP
BERGS,Berglunds snabbköp,Berguvsvägen 8,Luleå,S-958 22
BLAUS,Blauer See Delikatessen,Forsterstr. 57,Mannheim,68306
BLONP,Blondesddsl père et fils,24, place Kléber,Strasbourg,67000
BOLID,Bólido Comidas preparadas,C/ Araquil, 67,Madrid,28023
BONAP,Bon app',12, rue des Bouchers,Marseille,13008
BOTTM,Bottom-Dollar Markets,23 Tsawassen Blvd.,Tsawassen,T2F 8M4
BSBEV,B's Beverages,Fauntleroy Circus,London,EC2 5NT
CACTU,Cactus Comidas para llevar,Cerrito 333,Buenos Aires,1010
CENTC,Centro comercial Moctezuma,Sierras de Granada 9993,México D.F.,05022
CHOPS,Chop-suey Chinese,Hauptstr. 29,Bern,3012
COMMI,Comércio Mineiro,Av. dos Lusíadas, 23,Sao Paulo,05432-043
CONSH,Consolidated Holdings,Berkely Gardens 12 Brewery,London,WX1 6LT
DRACD,Drachenblut Delikatessen,Walserweg 21,Aachen,52066
DUMON,Du monde entier,67, rue des Cinquante Otages,Nantes,44000
EASTC,Eastern Connection,35 King George,London,WX3 6FW
ERNSH,Ernst Handel,Kirchgasse 6,Graz,8010
FAMIA,Familia Arquibaldo,Rua Orós, 92,Sao Paulo,05442-030
FISSA,FISSA Fabrica Inter. Salchichas S.A.,C/ Moralzarzal, 86, Madrid,28034
FOLIG,Folies gourmandes,184, chaussée de Tournai,Lille,59000
FOLKO,Folk och fä HB,Åkergratan 24,Bräcke,S-844 67
FRANK,Frankenversand,Berliner Platz 43,München,80805
FRANR,France restauration,54, rue Royale,Nantes,44000
FRANS,Franchi S.p.A.,Via Monte Bianco 34,Torino,10100
FIIRTR,Furia Bacalhau e Frutos do Mar,Jardim das rosas n. 32,Lisboa 1675

GALED,Galería del gastrónomo,Rambla de Cataluña, 23,Barcelona,08022
GODOS,Godos Cocina Típica,C/ Romero, 33,Sevilla,41101
GOURL,Gourmet Lanchonetes,Av. Brasil, 442,Campinas,04876-786
GREAL,Great Lakes Food Market,2732 Baker Blvd.,Eugene,97403
GROSER,GROSELLA-Restaurante,5^a Ave. Los Palos Grandes,Caracas,1081
HANAR,Hanari Carnes,Rua do Paço, 67,Rio de Janeiro,05454-876
HILAA,HILARION-Abastos,Carrera 22 con Ave. Carlos Soublette #8-35,San Cristóbal,5022
HUNG,C,Hungry Coyote Import Store,City Center Plaza 516 Main St.,Elgin,97827
HUNGO,Hungry Owl All-Night Grocers,8 Johnstown Road,Cork,
ISLAT,Island Trading,Garden House Crowther Way,Cowes,PO31 7PJ
KOENE,Königlich Essen,Maibelstr. 90,Brandenburg,14776
LACOR,La corne d'abondance,67, avenue de l'Europe,Versailles,78000
LAMAI,La maison d'Asie,1 rue Alsace-Lorraine,Toulouse,31000
LAUGB,Laughing Bacchus Wine Cellars,1900 Oak St.,Vancouver,V3F 2K1
LAZYK,Lazy K Kountry Store,12 Orchestra Terrace,Walla Walla,99362
LEHMS,Lehmans Marktstand,Magazinweg 7,Frankfurt a.M.,60528
LETSS,Let's Stop N Shop,87 Polk St. Suite 5,San Francisco,94117
LILAS,LILA-Supermercado,Carrera 52 con Ave. Bolívar #65-98 Llano Largo,Barquisimeto,3508
LINOD,LINO-Delicateses,Ave. 5 de Mayo Porlamar,I. de Margarita,4980
LONEP,Lonesome Pine Restaurant,89 Chiaroscuro Rd.,Portland,97219
MAGAA,Magazzini Alimentari Riuniti,Via Ludovico il Moro 22,Bergamo,24100
MAISD,Maison Dewey,Rue Joseph-Bens 532,Bruxelles,B-1180
END CUSTOMERS

ORDERS

10250,HANAR,7/8/1996 12:00:00 AM,7/12/1996 12:00:00 AM
10253,HANAR,7/10/1996 12:00:00 AM,7/16/1996 12:00:00 AM
10254,CHOPS,7/11/1996 12:00:00 AM,7/23/1996 12:00:00 AM
10257,HILAA,7/16/1996 12:00:00 AM,7/22/1996 12:00:00 AM
10258,ERNSH,7/17/1996 12:00:00 AM,7/23/1996 12:00:00 AM
10263,ERNSH,7/23/1996 12:00:00 AM,7/31/1996 12:00:00 AM
10264,FOLKO,7/24/1996 12:00:00 AM,8/23/1996 12:00:00 AM
10265,BLONP,7/25/1996 12:00:00 AM,8/12/1996 12:00:00 AM
10267,FRANK,7/29/1996 12:00:00 AM,8/6/1996 12:00:00 AM
10275,MAGAA,8/7/1996 12:00:00 AM,8/9/1996 12:00:00 AM
10278,BERGS,8/12/1996 12:00:00 AM,8/16/1996 12:00:00 AM
10279,LEHMS,8/13/1996 12:00:00 AM,8/16/1996 12:00:00 AM
10280,BERGS,8/14/1996 12:00:00 AM,9/12/1996 12:00:00 AM
10283,LILAS,8/16/1996 12:00:00 AM,8/23/1996 12:00:00 AM
10284,LEHMS,8/19/1996 12:00:00 AM,8/27/1996 12:00:00 AM
10289,BSBEV,8/26/1996 12:00:00 AM,8/28/1996 12:00:00 AM
10290,COMM,8/27/1996 12:00:00 AM,9/3/1996 12:00:00 AM
10296,LILAS,9/3/1996 12:00:00 AM,9/11/1996 12:00:00 AM
10297,BLONP,9/4/1996 12:00:00 AM,9/10/1996 12:00:00 AM
10298,HUNGO,9/5/1996 12:00:00 AM,9/11/1996 12:00:00 AM
10300,MAGAA,9/9/1996 12:00:00 AM,9/18/1996 12:00:00 AM
10303,GODOS,9/11/1996 12:00:00 AM,9/18/1996 12:00:00 AM
10307,LONEP,9/17/1996 12:00:00 AM,9/25/1996 12:00:00 AM
10309,HUNGO,9/19/1996 12:00:00 AM,10/23/1996 12:00:00 AM
10315,ISLAT,9/26/1996 12:00:00 AM,10/3/1996 12:00:00 AM
10317,LONEP,9/30/1996 12:00:00 AM,10/10/1996 12:00:00 AM
10318,ISLAT,10/1/1996 12:00:00 AM,10/4/1996 12:00:00 AM
10321,ISLAT,10/3/1996 12:00:00 AM,10/11/1996 12:00:00 AM
10323,KOENE,10/7/1996 12:00:00 AM,10/14/1996 12:00:00 AM
10325,KOENE,10/9/1996 12:00:00 AM,10/14/1996 12:00:00 AM
10327,FOLKO,10/11/1996 12:00:00 AM,10/14/1996 12:00:00 AM
10328,FURIB,10/14/1996 12:00:00 AM,10/17/1996 12:00:00 AM
10330,LILAS,10/16/1996 12:00:00 AM,10/28/1996 12:00:00 AM
10331,BONAP,10/16/1996 12:00:00 AM,10/21/1996 12:00:00 AM
10335,HUNGO,10/22/1996 12:00:00 AM,10/24/1996 12:00:00 AM
10337,FRANK,10/24/1996 12:00:00 AM,10/29/1996 12:00:00 AM
10340,BONAP,10/29/1996 12:00:00 AM,11/8/1996 12:00:00 AM
10342,FRANK,10/30/1996 12:00:00 AM,11/4/1996 12:00:00 AM
10343,LEHMS,10/31/1996 12:00:00 AM,11/6/1996 12:00:00 AM
10347,FAMIA,11/6/1996 12:00:00 AM,11/8/1996 12:00:00 AM
10350,LAMAI,11/11/1996 12:00:00 AM,12/3/1996 12:00:00 AM
10351,ERNSH,11/11/1996 12:00:00 AM,11/20/1996 12:00:00 AM
10352,FURIB,11/12/1996 12:00:00 AM,11/18/1996 12:00:00 AM

10355,AKOUI,11/15/1996 12:00:00 AM,11/20/1996 12:00:00 AM
10357,LILAS,11/19/1996 12:00:00 AM,12/2/1996 12:00:00 AM
10358,LAMAI,11/20/1996 12:00:00 AM,11/27/1996 12:00:00 AM
10360,BLONP,11/22/1996 12:00:00 AM,12/2/1996 12:00:00 AM
10362,BONAP,11/25/1996 12:00:00 AM,11/28/1996 12:00:00 AM
10363,DRACD,11/26/1996 12:00:00 AM,12/4/1996 12:00:00 AM
10364,EASTC,11/26/1996 12:00:00 AM,12/4/1996 12:00:00 AM
10365,ANTON,11/27/1996 12:00:00 AM,12/2/1996 12:00:00 AM
10366,GALED,11/28/1996 12:00:00 AM,12/30/1996 12:00:00 AM
10368,ERNSH,11/29/1996 12:00:00 AM,12/2/1996 12:00:00 AM
10370,CHOPS,12/3/1996 12:00:00 AM,12/27/1996 12:00:00 AM
10371,LAMAI,12/3/1996 12:00:00 AM,12/24/1996 12:00:00 AM
10373,HUNGO,12/5/1996 12:00:00 AM,12/11/1996 12:00:00 AM
10375,HUNGC,12/6/1996 12:00:00 AM,12/9/1996 12:00:00 AM
10378,FOLKO,12/10/1996 12:00:00 AM,12/19/1996 12:00:00 AM
10380,HUNGO,12/12/1996 12:00:00 AM,1/16/1997 12:00:00 AM
10381,LILAS,12/12/1996 12:00:00 AM,12/13/1996 12:00:00 AM
10382,ERNSH,12/13/1996 12:00:00 AM,12/16/1996 12:00:00 AM
10383,AROUT,12/16/1996 12:00:00 AM,12/18/1996 12:00:00 AM
10384,BERGS,12/16/1996 12:00:00 AM,12/20/1996 12:00:00 AM
10386,FAMIA,12/18/1996 12:00:00 AM,12/25/1996 12:00:00 AM
10389,BOTTM,12/20/1996 12:00:00 AM,12/24/1996 12:00:00 AM
10391,DRACD,12/23/1996 12:00:00 AM,12/31/1996 12:00:00 AM
10394,HUNGC,12/25/1996 12:00:00 AM,1/3/1997 12:00:00 AM
10395,HILAA,12/26/1996 12:00:00 AM,1/3/1997 12:00:00 AM
10396,FRANK,12/27/1996 12:00:00 AM,1/6/1997 12:00:00 AM
10400,EASTC,1/1/1997 12:00:00 AM,1/16/1997 12:00:00 AM
10404,MAGAA,1/3/1997 12:00:00 AM,1/8/1997 12:00:00 AM
10405,LINOD,1/6/1997 12:00:00 AM,1/22/1997 12:00:00 AM
10408,FOLIG,1/8/1997 12:00:00 AM,1/14/1997 12:00:00 AM
10410,BOTTM,1/10/1997 12:00:00 AM,1/15/1997 12:00:00 AM
10411,BOTTM,1/10/1997 12:00:00 AM,1/21/1997 12:00:00 AM
10413,LAMAI,1/14/1997 12:00:00 AM,1/16/1997 12:00:00 AM
10414,FAMIA,1/14/1997 12:00:00 AM,1/17/1997 12:00:00 AM
10415,HUNGC,1/15/1997 12:00:00 AM,1/24/1997 12:00:00 AM
10422,FRANS,1/22/1997 12:00:00 AM,1/31/1997 12:00:00 AM
10423,GOURL,1/23/1997 12:00:00 AM,2/24/1997 12:00:00 AM
10425,LAMAI,1/24/1997 12:00:00 AM,2/14/1997 12:00:00 AM
10426,GALED,1/27/1997 12:00:00 AM,2/6/1997 12:00:00 AM
10431,BOTTM,1/30/1997 12:00:00 AM,2/7/1997 12:00:00 AM
10434,FOLKO,2/3/1997 12:00:00 AM,2/13/1997 12:00:00 AM
10436,BLONP,2/5/1997 12:00:00 AM,2/11/1997 12:00:00 AM
10444,BERGS,2/12/1997 12:00:00 AM,2/21/1997 12:00:00 AM
10445,BERGS,2/13/1997 12:00:00 AM,2/20/1997 12:00:00 AM
10449,BLONP,2/18/1997 12:00:00 AM,2/27/1997 12:00:00 AM
10453,AROUT,2/21/1997 12:00:00 AM,2/26/1997 12:00:00 AM
10456,KOENE,2/25/1997 12:00:00 AM,2/28/1997 12:00:00 AM
10457,KOENE,2/25/1997 12:00:00 AM,3/3/1997 12:00:00 AM
10460,FOLKO,2/28/1997 12:00:00 AM,3/3/1997 12:00:00 AM
10464,FURIB,3/4/1997 12:00:00 AM,3/14/1997 12:00:00 AM
10466,COMMI,3/6/1997 12:00:00 AM,3/13/1997 12:00:00 AM
10467,MAGAA,3/6/1997 12:00:00 AM,3/11/1997 12:00:00 AM
10468,KOENE,3/7/1997 12:00:00 AM,3/12/1997 12:00:00 AM
10470,BONAP,3/11/1997 12:00:00 AM,3/14/1997 12:00:00 AM
10471,BSBEV,3/11/1997 12:00:00 AM,3/18/1997 12:00:00 AM
10473,ISLAT,3/13/1997 12:00:00 AM,3/21/1997 12:00:00 AM
10476,HILAA,3/17/1997 12:00:00 AM,3/24/1997 12:00:00 AM
10480,FOLIG,3/20/1997 12:00:00 AM,3/24/1997 12:00:00 AM
10484,BSBEV,3/24/1997 12:00:00 AM,4/1/1997 12:00:00 AM
10485,LINOD,3/25/1997 12:00:00 AM,3/31/1997 12:00:00 AM
10486,HILAA,3/26/1997 12:00:00 AM,4/2/1997 12:00:00 AM
10488,FRANK,3/27/1997 12:00:00 AM,4/2/1997 12:00:00 AM
10490,HILAA,3/31/1997 12:00:00 AM,4/3/1997 12:00:00 AM
10491,FURIB,3/31/1997 12:00:00 AM,4/8/1997 12:00:00 AM
10492,BOTTM,4/1/1997 12:00:00 AM,4/11/1997 12:00:00 AM
10494,COMMI,4/2/1997 12:00:00 AM,4/9/1997 12:00:00 AM
10497,LEHMS,4/4/1997 12:00:00 AM,4/7/1997 12:00:00 AM
10501,BLAUS,4/9/1997 12:00:00 AM,4/16/1997 12:00:00 AM
10507,ANTON,4/15/1997 12:00:00 AM,4/22/1997 12:00:00 AM

10509,BLAUS,4/17/1997 12:00:00 AM,4/29/1997 12:00:00 AM
10511,BONAP,4/18/1997 12:00:00 AM,4/21/1997 12:00:00 AM
10512,FAMIA,4/21/1997 12:00:00 AM,4/24/1997 12:00:00 AM
10519,CHOPS,4/28/1997 12:00:00 AM,5/1/1997 12:00:00 AM
10521,CACTU,4/29/1997 12:00:00 AM,5/2/1997 12:00:00 AM
10522,LEHMS,4/30/1997 12:00:00 AM,5/6/1997 12:00:00 AM
10528,GREAL,5/6/1997 12:00:00 AM,5/9/1997 12:00:00 AM
10529,MAISD,5/7/1997 12:00:00 AM,5/9/1997 12:00:00 AM
10532,EASTC,5/9/1997 12:00:00 AM,5/12/1997 12:00:00 AM
10535,ANTON,5/13/1997 12:00:00 AM,5/21/1997 12:00:00 AM
10538,BSBEV,5/15/1997 12:00:00 AM,5/16/1997 12:00:00 AM
10539,BSBEV,5/16/1997 12:00:00 AM,5/23/1997 12:00:00 AM
10541,HANAR,5/19/1997 12:00:00 AM,5/29/1997 12:00:00 AM
10544,LONEP,5/21/1997 12:00:00 AM,5/30/1997 12:00:00 AM
10550,GODOS,5/28/1997 12:00:00 AM,6/6/1997 12:00:00 AM
10551,FURIB,5/28/1997 12:00:00 AM,6/6/1997 12:00:00 AM
10558,AROUT,6/4/1997 12:00:00 AM,6/10/1997 12:00:00 AM
10568,GALED,6/13/1997 12:00:00 AM,7/9/1997 12:00:00 AM
10573,ANTON,6/19/1997 12:00:00 AM,6/20/1997 12:00:00 AM
10581,FAMIA,6/26/1997 12:00:00 AM,7/2/1997 12:00:00 AM
10582,BLAUS,6/27/1997 12:00:00 AM,7/14/1997 12:00:00 AM
10589,GREAL,7/4/1997 12:00:00 AM,7/14/1997 12:00:00 AM
10600,HUNGC,7/16/1997 12:00:00 AM,7/21/1997 12:00:00 AM
10614,BLAUS,7/29/1997 12:00:00 AM,8/1/1997 12:00:00 AM
10616,GREAL,7/31/1997 12:00:00 AM,8/5/1997 12:00:00 AM
10617,GREAL,7/31/1997 12:00:00 AM,8/4/1997 12:00:00 AM
10621,ISLAT,8/5/1997 12:00:00 AM,8/11/1997 12:00:00 AM
10629,GODOS,8/12/1997 12:00:00 AM,8/20/1997 12:00:00 AM
10634,FOLIG,8/15/1997 12:00:00 AM,8/21/1997 12:00:00 AM
10635,MAGAA,8/18/1997 12:00:00 AM,8/21/1997 12:00:00 AM
10638,LINOD,8/20/1997 12:00:00 AM,9/1/1997 12:00:00 AM
10643,ALFKI,8/25/1997 12:00:00 AM,9/2/1997 12:00:00 AM
10645,HANAR,8/26/1997 12:00:00 AM,9/2/1997 12:00:00 AM
10649,MAISD,8/28/1997 12:00:00 AM,8/29/1997 12:00:00 AM
10652,GOURL,9/1/1997 12:00:00 AM,9/8/1997 12:00:00 AM
10656,GREAL,9/4/1997 12:00:00 AM,9/10/1997 12:00:00 AM
10660,HUNGC,9/8/1997 12:00:00 AM,10/15/1997 12:00:00 AM
10662,LONEP,9/9/1997 12:00:00 AM,9/18/1997 12:00:00 AM
10665,LONEP,9/11/1997 12:00:00 AM,9/17/1997 12:00:00 AM
10677,ANTON,9/22/1997 12:00:00 AM,9/26/1997 12:00:00 AM
10685,GOURL,9/29/1997 12:00:00 AM,10/3/1997 12:00:00 AM
10690,HANAR,10/2/1997 12:00:00 AM,10/3/1997 12:00:00 AM
10692,ALFKI,10/3/1997 12:00:00 AM,10/13/1997 12:00:00 AM
10697,LINOD,10/8/1997 12:00:00 AM,10/14/1997 12:00:00 AM
10702,ALFKI,10/13/1997 12:00:00 AM,10/21/1997 12:00:00 AM
10707,AROUT,10/16/1997 12:00:00 AM,10/23/1997 12:00:00 AM
10709,GOURL,10/17/1997 12:00:00 AM,11/20/1997 12:00:00 AM
10710,FRANS,10/20/1997 12:00:00 AM,10/23/1997 12:00:00 AM
10726,EASTC,11/3/1997 12:00:00 AM,12/5/1997 12:00:00 AM
10729,LINOD,11/4/1997 12:00:00 AM,11/14/1997 12:00:00 AM
10731,CHOPS,11/6/1997 12:00:00 AM,11/14/1997 12:00:00 AM
10734,GOURL,11/7/1997 12:00:00 AM,11/12/1997 12:00:00 AM
10746,CHOPS,11/19/1997 12:00:00 AM,11/21/1997 12:00:00 AM
10753,FRANS,11/25/1997 12:00:00 AM,11/27/1997 12:00:00 AM
10760,MAISD,12/1/1997 12:00:00 AM,12/10/1997 12:00:00 AM
10763,FOLIG,12/3/1997 12:00:00 AM,12/8/1997 12:00:00 AM
10782,CACTU,12/17/1997 12:00:00 AM,12/22/1997 12:00:00 AM
10789,FOLIG,12/22/1997 12:00:00 AM,12/31/1997 12:00:00 AM
10797,DRACD,12/25/1997 12:00:00 AM,1/5/1998 12:00:00 AM
10807,FRANS,12/31/1997 12:00:00 AM,1/30/1998 12:00:00 AM
10819,CACTU,1/7/1998 12:00:00 AM,1/16/1998 12:00:00 AM
10825,DRACD,1/9/1998 12:00:00 AM,1/14/1998 12:00:00 AM
10835,ALFKI,1/15/1998 12:00:00 AM,1/21/1998 12:00:00 AM
10853,BLAUS,1/27/1998 12:00:00 AM,2/3/1998 12:00:00 AM
10872,GODOS,2/5/1998 12:00:00 AM,2/9/1998 12:00:00 AM
10874,GODOS,2/6/1998 12:00:00 AM,2/11/1998 12:00:00 AM
10881,CACTU,2/11/1998 12:00:00 AM,2/18/1998 12:00:00 AM
10887,GALED,2/13/1998 12:00:00 AM,2/16/1998 12:00:00 AM
10892,MAISD,2/17/1998 12:00:00 AM,2/19/1998 12:00:00 AM

10896,MAISD,2/19/1998 12:00:00 AM,2/27/1998 12:00:00 AM
10928,GALED,3/5/1998 12:00:00 AM,3/18/1998 12:00:00 AM
10937,CACTU,3/10/1998 12:00:00 AM,3/13/1998 12:00:00 AM
10952,ALFKI,3/16/1998 12:00:00 AM,3/24/1998 12:00:00 AM
10969,COMMI,3/23/1998 12:00:00 AM,3/30/1998 12:00:00 AM
10987,EASTC,3/31/1998 12:00:00 AM,4/6/1998 12:00:00 AM
11026,FRANS,4/15/1998 12:00:00 AM,4/28/1998 12:00:00 AM
11036,DRACD,4/20/1998 12:00:00 AM,4/22/1998 12:00:00 AM
11042,COMMI,4/22/1998 12:00:00 AM,5/1/1998 12:00:00 AM
END ORDERS

ORDER DETAILS

10250,41,7.7000,10,0
10250,51,42.4000,35,0.15
10250,65,16.8000,15,0.15
10253,31,10.0000,20,0
10253,39,14.4000,42,0
10253,49,16.0000,40,0
10254,24,3.6000,15,0.15
10254,55,19.2000,21,0.15
10254,74,8.0000,21,0
10257,27,35.1000,25,0
10257,39,14.4000,6,0
10257,77,10.4000,15,0
10258,2,15.2000,50,0.2
10258,5,17.0000,65,0.2
10258,32,25.6000,6,0.2
10263,16,13.9000,60,0.25
10263,24,3.6000,28,0
10263,30,20.7000,60,0.25
10263,74,8.0000,36,0.25
10264,2,15.2000,35,0
10264,41,7.7000,25,0.15
10265,17,31.2000,30,0
10265,70,12.0000,20,0
10267,40,14.7000,50,0
10267,59,44.0000,70,0.15
10267,76,14.4000,15,0.15
10275,24,3.6000,12,0.05
10275,59,44.0000,6,0.05
10278,44,15.5000,16,0
10278,59,44.0000,15,0
10278,63,35.1000,8,0
10278,73,12.0000,25,0
10279,17,31.2000,15,0.25
10280,24,3.6000,12,0
10280,55,19.2000,20,0
10280,75,6.2000,30,0
10283,15,12.4000,20,0
10283,19,7.3000,18,0
10283,60,27.2000,35,0
10283,72,27.8000,3,0
10284,27,35.1000,15,0.25
10284,44,15.5000,21,0
10284,60,27.2000,20,0.25
10284,67,11.2000,5,0.25
10289,3,8.0000,30,0
10289,64,26.6000,9,0
10290,5,17.0000,20,0
10290,29,99.0000,15,0
10290,49,16.0000,15,0
10290,77,10.4000,10,0
10296,11,16.8000,12,0
10296,16,13.9000,30,0
10296,69,28.8000,15,0
10297,39,14.4000,60,0
10297,72,27.8000,20,0
10298,2,15.2000,40,0
10298,36,15.2000,40,0.25

10298,59,44.0000,30,0.25
10298,62,39.4000,15,0
10300,66,13.6000,30,0
10300,68,10.0000,20,0
10303,40,14.7000,40,0.1
10303,65,16.8000,30,0.1
10303,68,10.0000,15,0.1
10307,62,39.4000,10,0
10307,68,10.0000,3,0
10309,4,17.6000,20,0
10309,6,20.0000,30,0
10309,42,11.2000,2,0
10309,43,36.8000,20,0
10309,71,17.2000,3,0
10315,34,11.2000,14,0
10315,70,12.0000,30,0
10317,1,14.4000,20,0
10318,41,7.7000,20,0
10318,76,14.4000,6,0
10321,35,14.4000,10,0
10323,15,12.4000,5,0
10323,25,11.2000,4,0
10323,39,14.4000,4,0
10325,6,20.0000,6,0
10325,13,4.8000,12,0
10325,14,18.6000,9,0
10325,31,10.0000,4,0
10325,72,27.8000,40,0
10327,2,15.2000,25,0.2
10327,11,16.8000,50,0.2
10327,30,20.7000,35,0.2
10327,58,10.6000,30,0.2
10328,59,44.0000,9,0
10328,65,16.8000,40,0
10328,68,10.0000,10,0
10330,26,24.9000,50,0.15
10330,72,27.8000,25,0.15
10331,54,5.9000,15,0
10335,2,15.2000,7,0.2
10335,31,10.0000,25,0.2
10335,32,25.6000,6,0.2
10335,51,42.4000,48,0.2
10337,23,7.2000,40,0
10337,26,24.9000,24,0
10337,36,15.2000,20,0
10337,37,20.8000,28,0
10337,72,27.8000,25,0
10340,18,50.0000,20,0.05
10340,41,7.7000,12,0.05
10340,43,36.8000,40,0.05
10342,2,15.2000,24,0.2
10342,31,10.0000,56,0.2
10342,36,15.2000,40,0.2
10342,55,19.2000,40,0.2
10343,64,26.6000,50,0
10343,68,10.0000,4,0.05
10343,76,14.4000,15,0
10347,25,11.2000,10,0
10347,39,14.4000,50,0.15
10347,40,14.7000,4,0
10347,75,6.2000,6,0.15
10350,50,13.0000,15,0.1
10350,69,28.8000,18,0.1
10351,38,210.8000,20,0.05
10351,41,7.7000,13,0
10351,44,15.5000,77,0.05
10351,65,16.8000,10,0.05
10352,24,3.6000,10,0
10352,54,5.9000,20,0.15

10355,24,3.6000,25,0
10355,57,15.6000,25,0
10357,10,24.8000,30,0.2
10357,26,24.9000,16,0
10357,60,27.2000,8,0.2
10358,24,3.6000,10,0.05
10358,34,11.2000,10,0.05
10358,36,15.2000,20,0.05
10360,28,36.4000,30,0
10360,29,99.0000,35,0
10360,38,210.8000,10,0
10360,49,16.0000,35,0
10360,54,5.9000,28,0
10362,25,11.2000,50,0
10362,51,42.4000,20,0
10362,54,5.9000,24,0
10363,31,10.0000,20,0
10363,75,6.2000,12,0
10363,76,14.4000,12,0
10364,69,28.8000,30,0
10364,71,17.2000,5,0
10365,11,16.8000,24,0
10366,65,16.8000,5,0
10366,77,10.4000,5,0
10368,21,8.0000,5,0.1
10368,28,36.4000,13,0.1
10368,57,15.6000,25,0
10368,64,26.6000,35,0.1
10370,1,14.4000,15,0.15
10370,64,26.6000,30,0
10370,74,8.0000,20,0.15
10371,36,15.2000,6,0.2
10373,58,10.6000,80,0.2
10373,71,17.2000,50,0.2
10375,14,18.6000,15,0
10375,54,5.9000,10,0
10378,71,17.2000,6,0
10380,30,20.7000,18,0.1
10380,53,26.2000,20,0.1
10380,60,27.2000,6,0.1
10380,70,12.0000,30,0
10381,74,8.0000,14,0
10382,5,17.0000,32,0
10382,18,50.0000,9,0
10382,29,99.0000,14,0
10382,33,2.0000,60,0
10382,74,8.0000,50,0
10383,13,4.8000,20,0
10383,50,13.0000,15,0
10383,56,30.4000,20,0
10384,20,64.8000,28,0
10384,60,27.2000,15,0
10386,24,3.6000,15,0
10386,34,11.2000,10,0
10389,10,24.8000,16,0
10389,55,19.2000,15,0
10389,62,39.4000,20,0
10389,70,12.0000,30,0
10391,13,4.8000,18,0
10394,13,4.8000,10,0
10394,62,39.4000,10,0
10395,46,9.6000,28,0.1
10395,53,26.2000,70,0.1
10395,69,28.8000,8,0
10396,23,7.2000,40,0
10396,71,17.2000,60,0
10396,72,27.8000,21,0
10400,29,99.0000,21,0
10400,35,14.4000,35,0

10400,49,16.0000,30,0
10404,26,24.9000,30,0.05
10404,42,11.2000,40,0.05
10404,49,16.0000,30,0.05
10405,3,8.0000,50,0
10408,37,20.8000,10,0
10408,54,5.9000,6,0
10408,62,39.4000,35,0
10410,33,2.0000,49,0
10410,59,44.0000,16,0
10411,41,7.7000,25,0.2
10411,44,15.5000,40,0.2
10411,59,44.0000,9,0.2
10413,1,14.4000,24,0
10413,62,39.4000,40,0
10413,76,14.4000,14,0
10414,19,7.3000,18,0.05
10414,33,2.0000,50,0
10415,17,31.2000,2,0
10415,33,2.0000,20,0
10422,26,24.9000,2,0
10423,31,10.0000,14,0
10423,59,44.0000,20,0
10425,55,19.2000,10,0.25
10425,76,14.4000,20,0.25
10426,56,30.4000,5,0
10426,64,26.6000,7,0
10431,17,31.2000,50,0.25
10431,40,14.7000,50,0.25
10431,47,7.6000,30,0.25
10434,11,16.8000,6,0
10434,76,14.4000,18,0.15
10436,46,9.6000,5,0
10436,56,30.4000,40,0.1
10436,64,26.6000,30,0.1
10436,75,6.2000,24,0.1
10444,17,31.2000,10,0
10444,26,24.9000,15,0
10444,35,14.4000,8,0
10444,41,7.7000,30,0
10445,39,14.4000,6,0
10445,54,5.9000,15,0
10449,10,24.8000,14,0
10449,52,5.6000,20,0
10449,62,39.4000,35,0
10453,48,10.2000,15,0.1
10453,70,12.0000,25,0.1
10456,21,8.0000,40,0.15
10456,49,16.0000,21,0.15
10457,59,44.0000,36,0
10460,68,10.0000,21,0.25
10460,75,6.2000,4,0.25
10464,4,17.6000,16,0.2
10464,43,36.8000,3,0
10464,56,30.4000,30,0.2
10464,60,27.2000,20,0
10466,11,16.8000,10,0
10466,46,9.6000,5,0
10467,24,3.6000,28,0
10467,25,11.2000,12,0
10468,30,20.7000,8,0
10468,43,36.8000,15,0
10470,18,50.0000,30,0
10470,23,7.2000,15,0
10470,64,26.6000,8,0
10471,7,24.0000,30,0
10471,56,30.4000,20,0
10473,33,2.0000,12,0

10473,11,1/.2000,12,0
10476,55,19.2000,2,0.05
10476,70,12.0000,12,0
10480,47,7.6000,30,0
10480,59,44.0000,12,0
10484,21,8.0000,14,0
10484,40,14.7000,10,0
10484,51,42.4000,3,0
10485,2,15.2000,20,0.1
10485,3,8.0000,20,0.1
10485,55,19.2000,30,0.1
10485,70,12.0000,60,0.1
10486,11,16.8000,5,0
10486,51,42.4000,25,0
10486,74,8.0000,16,0
10488,59,44.0000,30,0
10488,73,12.0000,20,0.2
10490,59,44.0000,60,0
10490,68,10.0000,30,0
10490,75,6.2000,36,0
10491,44,15.5000,15,0.15
10491,77,10.4000,7,0.15
10492,25,11.2000,60,0.05
10492,42,11.2000,20,0.05
10494,56,30.4000,30,0
10497,56,30.4000,14,0
10497,72,27.8000,25,0
10497,77,10.4000,25,0
10501,54,7.4500,20,0
10507,43,46.0000,15,0.15
10507,48,12.7500,15,0.15
10509,28,45.6000,3,0
10511,4,22.0000,50,0.15
10511,7,30.0000,50,0.15
10511,8,40.0000,10,0.15
10512,24,4.5000,10,0.15
10512,46,12.0000,9,0.15
10512,47,9.5000,6,0.15
10512,60,34.0000,12,0.15
10519,10,31.0000,16,0.05
10519,56,38.0000,40,0
10519,60,34.0000,10,0.05
10521,35,18.0000,3,0
10521,41,9.6500,10,0
10521,68,12.5000,6,0
10522,1,18.0000,40,0.2
10522,8,40.0000,24,0
10522,30,25.8900,20,0.2
10522,40,18.4000,25,0.2
10528,11,21.0000,3,0
10528,33,2.5000,8,0.2
10528,72,34.8000,9,0
10529,55,24.0000,14,0
10529,68,12.5000,20,0
10529,69,36.0000,10,0
10532,30,25.8900,15,0
10532,66,17.0000,24,0
10535,11,21.0000,50,0.1
10535,40,18.4000,10,0.1
10535,57,19.5000,5,0.1
10535,59,55.0000,15,0.1
10538,70,15.0000,7,0
10538,72,34.8000,1,0
10539,13,6.0000,8,0
10539,21,10.0000,15,0
10539,33,2.5000,15,0
10539,49,20.0000,6,0
10541,24,4.5000,35,0.1
10541,38,263.5000,4,0.1

10541,65,21.0500,36,0.1
10541,71,21.5000,9,0.1
10544,28,45.6000,7,0
10544,67,14.0000,7,0
10550,17,39.0000,8,0.1
10550,19,9.2000,10,0
10550,21,10.0000,6,0.1
10550,61,28.5000,10,0.1
10551,16,17.4500,40,0.15
10551,35,18.0000,20,0.15
10551,44,19.4500,40,0
10558,47,9.5000,25,0
10558,51,53.0000,20,0
10558,52,7.0000,30,0
10558,53,32.8000,18,0
10558,73,15.0000,3,0
10568,10,31.0000,5,0
10573,17,39.0000,18,0
10573,34,14.0000,40,0
10573,53,32.8000,25,0
10581,75,7.7500,50,0.2
10582,57,19.5000,4,0
10582,76,18.0000,14,0
10589,35,18.0000,4,0
10600,54,7.4500,4,0
10600,73,15.0000,30,0
10614,11,21.0000,14,0
10614,21,10.0000,8,0
10614,39,18.0000,5,0
10616,38,263.5000,15,0.05
10616,56,38.0000,14,0
10616,70,15.0000,15,0.05
10616,71,21.5000,15,0.05
10617,59,55.0000,30,0.15
10621,19,9.2000,5,0
10621,23,9.0000,10,0
10621,70,15.0000,20,0
10621,71,21.5000,15,0
10629,29,123.7900,20,0
10629,64,33.2500,9,0
10634,7,30.0000,35,0
10634,18,62.5000,50,0
10634,51,53.0000,15,0
10634,75,7.7500,2,0
10635,4,22.0000,10,0.1
10635,5,21.3500,15,0.1
10635,22,21.0000,40,0
10638,45,9.5000,20,0
10638,65,21.0500,21,0
10638,72,34.8000,60,0
10643,28,45.6000,15,0.25
10643,39,18.0000,21,0.25
10643,46,12.0000,2,0.25
10645,18,62.5000,20,0
10645,36,19.0000,15,0
10649,28,45.6000,20,0
10649,72,34.8000,15,0
10652,30,25.8900,2,0.25
10652,42,14.0000,20,0
10656,14,23.2500,3,0.1
10656,44,19.4500,28,0.1
10656,47,9.5000,6,0.1
10660,20,81.0000,21,0
10662,68,12.5000,10,0
10665,51,53.0000,20,0
10665,59,55.0000,1,0
10665,76,18.0000,10,0
10677,26,31.2300,30,0.15
10677,33,2.5000,8,0.15

10685,10,31.0000,20,0
10685,41,9.6500,4,0
10685,47,9.5000,15,0
10690,56,38.0000,20,0.25
10690,77,13.0000,30,0.25
10692,63,43.9000,20,0
10697,19,9.2000,7,0.25
10697,35,18.0000,9,0.25
10697,58,13.2500,30,0.25
10697,70,15.0000,30,0.25
10702,3,10.0000,6,0
10702,76,18.0000,15,0
10707,55,24.0000,21,0
10707,57,19.5000,40,0
10707,70,15.0000,28,0.15
10709,8,40.0000,40,0
10709,51,53.0000,28,0
10709,60,34.0000,10,0
10710,19,9.2000,5,0
10710,47,9.5000,5,0
10726,4,22.0000,25,0
10726,11,21.0000,5,0
10729,1,18.0000,50,0
10729,21,10.0000,30,0
10729,50,16.2500,40,0
10731,21,10.0000,40,0.05
10731,51,53.0000,30,0.05
10734,6,25.0000,30,0
10734,30,25.8900,15,0
10734,76,18.0000,20,0
10746,13,6.0000,6,0
10746,42,14.0000,28,0
10746,62,49.3000,9,0
10746,69,36.0000,40,0
10753,45,9.5000,4,0
10753,74,10.0000,5,0
10760,25,14.0000,12,0.25
10760,27,43.9000,40,0
10760,43,46.0000,30,0.25
10763,21,10.0000,40,0
10763,22,21.0000,6,0
10763,24,4.5000,20,0
10782,31,12.5000,1,0
10789,18,62.5000,30,0
10789,35,18.0000,15,0
10789,63,43.9000,30,0
10789,68,12.5000,18,0
10797,11,21.0000,20,0
10807,40,18.4000,1,0
10819,43,46.0000,7,0
10819,75,7.7500,20,0
10825,26,31.2300,12,0
10825,53,32.8000,20,0
10835,59,55.0000,15,0
10835,77,13.0000,2,0.2
10853,18,62.5000,10,0
10872,55,24.0000,10,0.05
10872,62,49.3000,20,0.05
10872,64,33.2500,15,0.05
10872,65,21.0500,21,0.05
10874,10,31.0000,10,0
10881,73,15.0000,10,0
10887,25,14.0000,5,0
10892,59,55.0000,40,0.05
10896,45,9.5000,15,0
10896,56,38.0000,16,0
10928,47,9.5000,5,0
10928,76,18.0000,5,0
10937,28,45.6000,8,0

10937,34,14.0000,20,0
10952,6,25.0000,16,0.05
10952,28,45.6000,2,0
10969,46,12.0000,9,0
10987,7,30.0000,60,0
10987,43,46.0000,6,0
10987,72,34.8000,20,0
11026,18,62.5000,8,0
11026,51,53.0000,10,0
11036,13,6.0000,7,0
11036,59,55.0000,30,0
11042,44,19.4500,15,0
11042,61,28.5000,4,0
END ORDER DETAILS

PRODUCTS

1,Chai,18.0000
2,Chang,19.0000
3,Aniseed Syrup,10.0000
4,Chef Anton's Cajun Seasoning,22.0000
5,Chef Anton's Gumbo Mix,21.3500
6,Grandma's Boysenberry Spread,25.0000
7,Uncle Bob's Organic Dried Pears,30.0000
8,Northwoods Cranberry Sauce,40.0000
9,Mishi Kobe Niku,97.0000
10,Ikura,31.0000
11,Queso Cabrales,21.0000
12,Queso Manchego La Pastora,38.0000
13,Konbu,6.0000
14,Tofu,23.2500
15,Genen Shouyu,15.5000
16,Pavlova,17.4500
17,Alice Mutton,39.0000
18,Carnarvon Tigers,62.5000
19,Teatime Chocolate Biscuits,9.2000
20,Sir Rodney's Marmalade,81.0000
21,Sir Rodney's Scones,10.0000
22,Gustaf's Knäckebröd,21.0000
23,Tunnbröd,9.0000
24,Guaraná Fantástica,4.5000
25,NuNuCa Nuß-Nougat-Creme,14.0000
26,Gumbär Gummibärchen,31.2300
27,Schoggi Schokolade,43.9000
28,Rössle Sauerkraut,45.6000
29,Thüringer Rostbratwurst,123.7900
30,Nord-Ost Matjeshering,25.8900
31,Gorgonzola Telino,12.5000
32,Mascarpone Fabioli,32.0000
33,Geitost,2.5000
34,Sasquatch Ale,14.0000
35,Steeleye Stout,18.0000
36,Inlagd Sill,19.0000
37,Gravad lax,26.0000
38,Côte de Blaye,263.5000
39,Chartreuse verte,18.0000
40,Boston Crab Meat,18.4000
41,Jack's New England Clam Chowder,9.6500
42,Singaporean Hokkien Fried Mee,14.0000
43,Ipoh Coffee,46.0000
44,Gula Malacca,19.4500
45,Rogede sild,9.5000
46,Spegesild,12.0000
47,Zaanse koeken,9.5000
48,Chocolade,12.7500
49,Maxilaku,20.0000
50,Valkoinen suklaa,16.2500
51,Manjimup Dried Apples,53.0000
52,Filo Mix,7.0000
53,Perth Pasties,32.8000

54,Tourtière,7.4500
55,Pâté chinois,24.0000
56,Gnocchi di nonna Alice,38.0000
57,Ravioli Angelo,19.5000
58,Escargots de Bourgogne,13.2500
59,Raclette Courdavault,55.0000
60,Camembert Pierrot,34.0000
61,Sirop d'érable,28.5000
62,Tarte au sucre,49.3000
63,Vegie-spread,43.9000
64,Wimmers gute Semmelknödel,33.2500
65,Louisiana Fiery Hot Pepper Sauce,21.0500
66,Louisiana Hot Spiced Okra,17.0000
67,Laughing Lumberjack Lager,14.0000
68,Scottish Longbreads,12.5000
69,Gudbrandsdalsost,36.0000
70,Outback Lager,15.0000
71,Flotemysost,21.5000
72,Mozzarella di Giovanni,34.8000
73,Röd Kaviar,15.0000
74,Longlife Tofu,10.0000
75,Rhönbräu Klosterbier,7.7500
76,Lakkaliköri,18.0000
77,Original Frankfurter grüne Soße,13.0000
END PRODUCTS

See also

- [Parallel LINQ \(PLINQ\)](#)

Data Structures for Parallel Programming

9/20/2022 • 3 minutes to read • [Edit Online](#)

.NET provides several types that are useful in parallel programming, including a set of concurrent collection classes, lightweight synchronization primitives, and types for lazy initialization. You can use these types with any multithreaded application code, including the Task Parallel Library and PLINQ.

Concurrent Collection Classes

The collection classes in the [System.Collections.Concurrent](#) namespace provide thread-safe add and remove operations that avoid locks wherever possible and use fine-grained locking where locks are necessary. A concurrent collection class does not require user code to take any locks when it accesses items. The concurrent collection classes can significantly improve performance over types such as [System.Collections.ArrayList](#) and [System.Collections.Generic.List<T>](#) (with user-implemented locking) in scenarios where multiple threads add and remove items from a collection.

The following table lists the concurrent collection classes:

TYPE	DESCRIPTION
System.Collections.Concurrent.BlockingCollection<T>	Provides blocking and bounding capabilities for thread-safe collections that implement System.Collections.Concurrent.IProducerConsumerCollection<T> . Producer threads block if no slots are available or if the collection is full. Consumer threads block if the collection is empty. This type also supports non-blocking access by consumers and producers. BlockingCollection<T> can be used as a base class or backing store to provide blocking and bounding for any collection class that supports IEnumerable<T> .
System.Collections.Concurrent.ConcurrentBag<T>	A thread-safe bag implementation that provides scalable add and get operations.
System.Collections.Concurrent.ConcurrentDictionary<TKey, TValue>	A concurrent and scalable dictionary type.
System.Collections.Concurrent.ConcurrentQueue<T>	A concurrent and scalable FIFO queue.
System.Collections.Concurrent.ConcurrentStack<T>	A concurrent and scalable LIFO stack.

For more information, see [Thread-Safe Collections](#).

Synchronization Primitives

The synchronization primitives in the [System.Threading](#) namespace enable fine-grained concurrency and faster performance by avoiding expensive locking mechanisms found in legacy multithreading code.

The following table lists the synchronization types:

TYPE	DESCRIPTION
System.Threading.Barrier	Enables multiple threads to work on an algorithm in parallel by providing a point at which each task can signal its arrival and then block until some or all tasks have arrived. For more information, see Barrier .
System.Threading.CountdownEvent	Simplifies fork and join scenarios by providing an easy rendezvous mechanism. For more information, see CountdownEvent .
System.Threading.ManualResetEventSlim	A synchronization primitive similar to System.Threading.ManualResetEvent . ManualResetEventSlim is lighter-weight but can only be used for intra-process communication.
System.ThreadingSemaphoreSlim	A synchronization primitive that limits the number of threads that can concurrently access a resource or a pool of resources. For more information, see Semaphore and SemaphoreSlim .
System.Threading.SpinLock	A mutual exclusion lock primitive that causes the thread that is trying to acquire the lock to wait in a loop, or <i>spin</i> , for a period of time before yielding its quantum. In scenarios where the wait for the lock is expected to be short, SpinLock offers better performance than other forms of locking. For more information, see SpinLock .
System.Threading.SpinWait	A small, lightweight type that will spin for a specified time and eventually put the thread into a wait state if the spin count is exceeded. For more information, see SpinWait .

For more information, see:

- [How to: Use SpinLock for Low-Level Synchronization](#)
- [How to: Synchronize Concurrent Operations with a Barrier](#).

Lazy Initialization Classes

With lazy initialization, the memory for an object is not allocated until it is needed. Lazy initialization can improve performance by spreading object allocations evenly across the lifetime of a program. You can enable lazy initialization for any custom type by wrapping the type [Lazy<T>](#).

The following table lists the lazy initialization types:

TYPE	DESCRIPTION
System.Lazy<T>	Provides lightweight, thread-safe lazy-initialization.
System.Threading.ThreadLocal<T>	Provides a lazily-initialized value on a per-thread basis, with each thread lazily-invoking the initialization function.
System.Threading.LazyInitializer	Provides static methods that avoid the need to allocate a dedicated, lazy-initialization instance. Instead, they use references to ensure targets have been initialized as they are accessed.

For more information, see [Lazy Initialization](#).

Aggregate Exceptions

The [System.AggregateException](#) type can be used to capture multiple exceptions that are thrown concurrently on separate threads, and return them to the joining thread as a single exception. The [System.Threading.Tasks.Task](#) and [System.Threading.Tasks.Parallel](#) types and PLINQ use [AggregateException](#) extensively for this purpose. For more information, see [Exception Handling](#) and [How to: Handle Exceptions in a PLINQ Query](#).

See also

- [System.Collections.Concurrent](#)
- [System.Threading](#)
- [Parallel Programming](#)

Parallel Diagnostic Tools

9/20/2022 • 2 minutes to read • [Edit Online](#)

Visual Studio provides extensive support for debugging and profiling multi-threaded applications.

Debugging

The Visual Studio debugger adds new windows for debugging parallel applications. For more information, see the following topics:

- [Using the Parallel Stacks Window](#)
- [Using the Tasks Window](#)
- [Walkthrough: Debugging a Parallel Application.](#)

Profiling

The Concurrency Visualizer report views enable you to visualize how the threads in a parallel program interact with each other and with threads from other processes on the system. For more information, see [Concurrency Visualizer](#).

See also

- [Parallel Programming](#)

Custom Partitioners for PLINQ and TPL

9/20/2022 • 10 minutes to read • [Edit Online](#)

To parallelize an operation on a data source, one of the essential steps is to *partition* the source into multiple sections that can be accessed concurrently by multiple threads. PLINQ and the Task Parallel Library (TPL) provide default partitioners that work transparently when you write a parallel query or `ForEach` loop. For more advanced scenarios, you can plug in your own partitioner.

Kinds of Partitioning

There are many ways to partition a data source. In the most efficient approaches, multiple threads cooperate to process the original source sequence, rather than physically separating the source into multiple subsequences. For arrays and other indexed sources such as `IList` collections where the length is known in advance, *range partitioning* is the simplest kind of partitioning. Every thread receives unique beginning and ending indexes, so that it can process its range of the source without overwriting or being overwritten by any other thread. The only overhead involved in range partitioning is the initial work of creating the ranges; no additional synchronization is required after that. Therefore, it can provide good performance as long as the workload is divided evenly. A disadvantage of range partitioning is that if one thread finishes early, it cannot help the other threads finish their work.

For linked lists or other collections whose length is not known, you can use *chunk partitioning*. In chunk partitioning, every thread or task in a parallel loop or query consumes some number of source elements in one chunk, processes them, and then comes back to retrieve additional elements. The partitioner ensures that all elements are distributed and that there are no duplicates. A chunk may be any size. For example, the partitioner that is demonstrated in [How to: Implement Dynamic Partitions](#) creates chunks that contain just one element. As long as the chunks are not too large, this kind of partitioning is inherently load-balancing because the assignment of elements to threads is not pre-determined. However, the partitioner does incur the synchronization overhead each time the thread needs to get another chunk. The amount of synchronization incurred in these cases is inversely proportional to the size of the chunks.

In general, range partitioning is only faster when the execution time of the delegate is small to moderate, and the source has a large number of elements, and the total work of each partition is roughly equivalent. Chunk partitioning is therefore generally faster in most cases. On sources with a small number of elements or longer execution times for the delegate, then the performance of chunk and range partitioning is about equal.

The TPL partitioners also support a dynamic number of partitions. This means they can create partitions on-the-fly, for example, when the `ForEach` loop spawns a new task. This feature enables the partitioner to scale together with the loop itself. Dynamic partitioners are also inherently load-balancing. When you create a custom partitioner, you must support dynamic partitioning to be consumable from a `ForEach` loop.

Configuring Load Balancing Partitioners for PLINQ

Some overloads of the `Partitioner.Create` method let you create a partitioner for an array or `IList` source and specify whether it should attempt to balance the workload among the threads. When the partitioner is configured to load-balance, chunk partitioning is used, and the elements are handed off to each partition in small chunks as they are requested. This approach helps ensure that all partitions have elements to process until the entire loop or query is completed. An additional overload can be used to provide load-balancing partitioning of any `IEnumerable` source.

In general, load balancing requires the partitions to request elements relatively frequently from the partitioner. By contrast, a partitioner that does static partitioning can assign the elements to each partitioner all at once by using either range or chunk partitioning. This requires less overhead than load balancing, but it might take

longer to execute if one thread ends up with significantly more work than the others. By default when it is passed an `IList` or an array, PLINQ always uses range partitioning without load balancing. To enable load balancing for PLINQ, use the `Partitioner.Create` method, as shown in the following example.

```
// Static partitioning requires indexable source. Load balancing
// can use any IEnumerable.
var nums = Enumerable.Range(0, 100000000).ToArray();

// Create a load-balancing partitioner. Or specify false for static partitioning.
Partitioner<int> customPartitioner = Partitioner.Create(nums, true);

// The partitioner is the query's data source.
var q = from x in customPartitioner.AsParallel()
        select x * Math.PI;

q.ForAll((x) =>
{
    ProcessData(x);
});
```

```
' Static number of partitions requires indexable source.
Dim nums = Enumerable.Range(0, 100000000).ToArray()

' Create a load-balancing partitioner. Or specify false For Shared partitioning.
Dim customPartitioner = Partitioner.Create(nums, True)

' The partitioner is the query's data source.
Dim q = From x In customPartitioner.AsParallel()
        Select x * Math.PI

q.ForAll(Sub(x) ProcessData(x))
```

The best way to determine whether to use load balancing in any given scenario is to experiment and measure how long it takes operations to complete under representative loads and computer configurations. For example, static partitioning might provide significant speedup on a multi-core computer that has only a few cores, but it might result in slowdowns on computers that have relatively many cores.

The following table lists the available overloads of the `Create` method. These partitioners are not limited to use only with PLINQ or `Task`. They can also be used with any custom parallel construct.

OVERLOAD	USES LOAD BALANCING
<code>Create<TSource>(IEnumerable<TSource>)</code>	Always
<code>Create<TSource>(TSource[], Boolean)</code>	When the Boolean argument is specified as true
<code>Create<TSource>(IList<TSource>, Boolean)</code>	When the Boolean argument is specified as true
<code>Create(Int32, Int32)</code>	Never
<code>Create(Int32, Int32, Int32)</code>	Never
<code>Create(Int64, Int64)</code>	Never
<code>Create(Int64, Int64, Int64)</code>	Never

Configuring Static Range Partitioners for Parallel.ForEach

In a [For](#) loop, the body of the loop is provided to the method as a delegate. The cost of invoking that delegate is about the same as a virtual method call. In some scenarios, the body of a parallel loop might be small enough that the cost of the delegate invocation on each loop iteration becomes significant. In such situations, you can use one of the [Create](#) overloads to create an [IEnumerable<T>](#) of range partitions over the source elements. Then, you can pass this collection of ranges to a [ForEach](#) method whose body consists of a regular [for](#) loop. The benefit of this approach is that the delegate invocation cost is incurred only once per range, rather than once per element. The following example demonstrates the basic pattern.

```
using System;
using System.Collections.Concurrent;
using System.Linq;
using System.Threading;
using System.Threading.Tasks;

class Program
{
    static void Main()
    {

        // Source must be array or IList.
        var source = Enumerable.Range(0, 100000).ToArray();

        // Partition the entire source array.
        var rangePartitioner = Partitioner.Create(0, source.Length);

        double[] results = new double[source.Length];

        // Loop over the partitions in parallel.
        Parallel.ForEach(rangePartitioner, (range, loopState) =>
        {
            // Loop over each range element without a delegate invocation.
            for (int i = range.Item1; i < range.Item2; i++)
            {
                results[i] = source[i] * Math.PI;
            }
        });
    }

    Console.WriteLine("Operation complete. Print results? y/n");
    char input = Console.ReadKey().KeyChar;
    if (input == 'y' || input == 'Y')
    {
        foreach(double d in results)
        {
            Console.Write("{0} ", d);
        }
    }
}
}
```

```

Imports System.Threading.Tasks
Imports System.Collections.Concurrent

Module PartitionDemo

Sub Main()
    ' Source must be array or IList.
    Dim source = Enumerable.Range(0, 100000).ToArray()

    ' Partition the entire source array.
    ' Let the partitioner size the ranges.
    Dim rangePartitioner = Partitioner.Create(0, source.Length)

    Dim results(source.Length - 1) As Double

    ' Loop over the partitions in parallel. The Sub is invoked
    ' once per partition.
    Parallel.ForEach(rangePartitioner, Sub(range, loopState)

        ' Loop over each range element without a delegate invocation.
        For i As Integer = range.Item1 To range.Item2 - 1
            results(i) = source(i) * Math.PI
        Next
    End Sub)
    Console.WriteLine("Operation complete. Print results? y/n")
    Dim input As Char = Console.ReadKey().KeyChar
    If input = "y"c Or input = "Y"c Then
        For Each d As Double In results
            Console.Write("{0} ", d)
        Next
    End If

    End Sub
End Module

```

Every thread in the loop receives its own `Tuple<T1,T2>` that contains the starting and ending index values in the specified sub-range. The inner `for` loop uses the `fromInclusive` and `toExclusive` values to loop over the array or the `IList` directly.

One of the `Create` overloads lets you specify the size of the partitions, and the number of partitions. This overload can be used in scenarios where the work per element is so low that even one virtual method call per element has a noticeable impact on performance.

Custom Partitioners

In some scenarios, it might be worthwhile or even required to implement your own partitioner. For example, you might have a custom collection class that you can partition more efficiently than the default partitioners can, based on your knowledge of the internal structure of the class. Or, you may want to create range partitions of varying sizes based on your knowledge of how long it will take to process elements at different locations in the source collection.

To create a basic custom partitioner, derive a class from `System.Collections.Concurrent.Partitioner<TSource>` and override the virtual methods, as described in the following table.

METHOD	DESCRIPTION
<code>GetPartitions</code>	This method is called once by the main thread and returns an <code>IList(IEnumerator(TSource))</code> . Each worker thread in the loop or query can call <code>GetEnumerator</code> on the list to retrieve a <code>IEnumerator<T></code> over a distinct partition.

METHOD	DESCRIPTION
SupportsDynamicPartitions	Return <code>true</code> if you implement <code>GetDynamicPartitions</code> , otherwise, <code>false</code> .
GetDynamicPartitions	If <code>SupportsDynamicPartitions</code> is <code>true</code> , this method can optionally be called instead of <code>GetPartitions</code> .

If the results must be sortable or you require indexed access into the elements, then derive from `System.Collections.Concurrent.OrderablePartitioner<TSource>` and override its virtual methods as described in the following table.

METHOD	DESCRIPTION
GetPartitions	This method is called once by the main thread and returns an <code>IList(IEnumerator(TSource))</code> . Each worker thread in the loop or query can call <code>GetEnumerator</code> on the list to retrieve a <code>IEnumerator<T></code> over a distinct partition.
SupportsDynamicPartitions	Return <code>true</code> if you implement <code>GetDynamicPartitions</code> ; otherwise, <code>false</code> .
GetDynamicPartitions	Typically, this just calls <code>GetOrderableDynamicPartitions</code> .
GetOrderableDynamicPartitions	If <code>SupportsDynamicPartitions</code> is <code>true</code> , this method can optionally be called instead of <code>GetPartitions</code> .

The following table provides additional details about how the three kinds of load-balancing partitioners implement the `OrderablePartitioner<TSource>` class.

METHOD/PROPERTY	ILIST / ARRAY WITHOUT LOAD BALANCING	ILIST / ARRAY WITH LOAD BALANCING	IENUMERABLE
GetOrderablePartitions	Uses range partitioning	Uses chunk partitioning optimized for Lists for the <code>partitionCount</code> specified	Uses chunk partitioning by creating a static number of partitions.
<code>OrderablePartitioner<TSource>.GetOrderableDynamicPartitions</code>	Throws not-supported exception	Uses chunk partitioning optimized for Lists and dynamic partitions	Uses chunk partitioning by creating a dynamic number of partitions.
KeysOrderedInEachPartition	Returns <code>true</code>	Returns <code>true</code>	Returns <code>true</code>
KeysOrderedAcrossPartitions	Returns <code>true</code>	Returns <code>false</code>	Returns <code>false</code>
KeysNormalized	Returns <code>true</code>	Returns <code>true</code>	Returns <code>true</code>
SupportsDynamicPartitions	Returns <code>false</code>	Returns <code>true</code>	Returns <code>true</code>

Dynamic Partitions

If you intend the partitioner to be used in a `ForEach` method, you must be able to return a dynamic number of partitions. This means that the partitioner can supply an enumerator for a new partition on-demand at any time during loop execution. Basically, whenever the loop adds a new parallel task, it requests a new partition for that

task. If you require the data to be orderable, then derive from [System.Collections.Concurrent.OrderablePartitioner<TSource>](#) so that each item in each partition is assigned a unique index.

For more information, and an example, see [How to: Implement Dynamic Partitions](#).

Contract for Partitioners

When you implement a custom partitioner, follow these guidelines to help ensure correct interaction with PLINQ and [ForEach](#) in the TPL:

- If [GetPartitions](#) is called with an argument of zero or less for `partitionsCount`, throw [ArgumentOutOfRangeException](#). Although PLINQ and TPL will never pass in a `partitionCount` equal to 0, we nevertheless recommend that you guard against the possibility.
- [GetPartitions](#) and [GetOrderablePartitions](#) should always return `partitionsCount` number of partitions. If the partitioner runs out of data and cannot create as many partitions as requested, then the method should return an empty enumerator for each of the remaining partitions. Otherwise, both PLINQ and TPL will throw an [InvalidOperationException](#).
- [GetPartitions](#), [GetOrderablePartitions](#), [GetDynamicPartitions](#), and [GetOrderableDynamicPartitions](#) should never return `null` (`Nothing` in Visual Basic). If they do, PLINQ / TPL will throw an [InvalidOperationException](#).
- Methods that return partitions should always return partitions that can fully and uniquely enumerate the data source. There should be no duplication in the data source or skipped items unless specifically required by the design of the partitioner. If this rule is not followed, then the output order may be scrambled.
- The following Boolean getters must always accurately return the following values so that the output order is not scrambled:
 - `KeysOrderedInEachPartition` : Each partition returns elements with increasing key indices.
 - `KeysOrderedAcrossPartitions` : For all partitions that are returned, the key indices in partition i are higher than the key indices in partition $i-1$.
 - `KeysNormalized` : All key indices are monotonically increasing without gaps, starting from zero.
- All indices must be unique. There may not be duplicate indices. If this rule is not followed, then the output order may be scrambled.
- All indices must be nonnegative. If this rule is not followed, then PLINQ/TPL may throw exceptions.

See also

- [Parallel Programming](#)
- [How to: Implement Dynamic Partitions](#)
- [How to: Implement a Partitioner for Static Partitioning](#)

How to: Implement Dynamic Partitions

9/20/2022 • 4 minutes to read • [Edit Online](#)

The following example shows how to implement a custom `System.Collections.Concurrent.OrderablePartitioner<TSource>` that implements dynamic partitioning and can be used from certain overloads `ForEach` and from PLINQ.

Example

Each time a partition calls `MoveNext` on the enumerator, the enumerator provides the partition with one list element. In the case of PLINQ and `ForEach`, the partition is a `Task` instance. Because requests are happening concurrently on multiple threads, access to the current index is synchronized.

```
//  
// An orderable dynamic partitioner for lists  
//  
using System;  
using System.Collections;  
using System.Collections.Concurrent;  
using System.Collections.Generic;  
using System.Diagnostics;  
using System.Linq;  
using System.Text;  
using System.Threading;  
using System.Threading.Tasks;  
using System.Xml.Linq;  
using System.Numerics;  
  
class OrderableListPartitioner<TSource> : OrderablePartitioner<TSource>  
{  
    private readonly IList<TSource> m_input;  
  
    // Must override to return true.  
    public override bool SupportsDynamicPartitions => true;  
  
    public OrderableListPartitioner(IList<TSource> input) : base(true, false, true) =>  
        m_input = input;  
  
    public override IList<IEnumerator<KeyValuePair<long, TSource>>> GetOrderablePartitions(int  
partitionCount)  
    {  
        var dynamicPartitions = GetOrderableDynamicPartitions();  
        var partitions =  
            new IEnumerator<KeyValuePair<long, TSource>>[partitionCount];  
  
        for (int i = 0; i < partitionCount; i++)  
        {  
            partitions[i] = dynamicPartitions.GetEnumerator();  
        }  
        return partitions;  
    }  
  
    public override IEnumerable<KeyValuePair<long, TSource>> GetOrderableDynamicPartitions() =>  
        new ListDynamicPartitions(m_input);  
  
    private class ListDynamicPartitions : IEnumerable<KeyValuePair<long, TSource>>  
    {  
        private IList<TSource> m_input;  
        private int m_pos = 0;
```

```

internal ListDynamicPartitions(IList<TSource> input) =>
    m_input = input;

public IEnumarator<KeyValuePair<long, TSource>> GetEnumerator()
{
    while (true)
    {
        // Each task gets the next item in the list. The index is
        // incremented in a thread-safe manner to avoid races.
        int elemIndex = Interlocked.Increment(ref m_pos) - 1;

        if (elemIndex >= m_input.Count)
        {
            yield break;
        }

        yield return new KeyValuePair<long, TSource>(
            elemIndex, m_input[elemIndex]);
    }
}

IEnumarable IEnumarable.GetEnumerator() =>
    ((IEnumarable<KeyValuePair<long, TSource>>)this).GetEnumerator();
}
}

```

```

class ConsumerClass
{
    static void Main()
    {
        var nums = Enumerable.Range(0, 10000).ToArray();
        OrderableListPartitioner<int> partitioner = new OrderableListPartitioner<int>(nums);

        // Use with Parallel.ForEach
        Parallel.ForEach(partitioner, (i) => Console.WriteLine(i));

        // Use with PLINQ
        var query = from num in partitioner.AsParallel()
                    where num % 2 == 0
                    select num;

        foreach (var v in query)
            Console.WriteLine(v);
    }
}

```

```

Imports System.Threading
Imports System.Threading.Tasks
Imports System.Collections.Concurrent

Module Module1
    Public Class OrderableListPartitioner(Of TSource)
        Inherits OrderablePartitioner(Of TSource)

        Private ReadOnly m_input As IList(Of TSource)

        Public Sub New(ByVal input As IList(Of TSource))
            MyBase.New(True, False, True)
            m_input = input
        End Sub

        ' Must override to return true.
        Public Overrides ReadOnly Property SupportsDynamicPartitions As Boolean
            Get
                Return True
            End Get
        End Property
    End Class

```

```

End Property

Public Overrides Function GetOrderablePartitions(ByVal partitionCount As Integer) As IList(Of
IEnumerator(Of KeyValuePair(Of Long, TSource)))
    Dim dynamicPartitions = GetOrderableDynamicPartitions()
    Dim partitions(partitionCount - 1) As IEnumerator(Of KeyValuePair(Of Long, TSource))

    For i = 0 To partitionCount - 1
        partitions(i) = dynamicPartitions.GetEnumerator()
    Next

    Return partitions
End Function

Public Overrides Function GetOrderableDynamicPartitions() As IEnumerable(Of KeyValuePair(Of Long,
TSource))
    Return New ListDynamicPartitions(m_input)
End Function

Private Class ListDynamicPartitions
    Implements IEnumerable(Of KeyValuePair(Of Long, TSource))

    Private m_input As IList(Of TSource)

    Friend Sub New(ByVal input As IList(Of TSource))
        m_input = input
    End Sub

    Public Function GetEnumerator() As IEnumerator(Of KeyValuePair(Of Long, TSource)) Implements
IEnumerable(Of KeyValuePair(Of Long, TSource)).GetEnumerator
        Return New ListDynamicPartitionsEnumerator(m_input)
    End Function

    Public Function GetEnumerator1() As IEnumerator Implements IEnumerable.GetEnumerator
        Return CType(Me, IEnumerable).GetEnumerator()
    End Function
End Class

Private Class ListDynamicPartitionsEnumerator
    Implements IEnumerator(Of KeyValuePair(Of Long, TSource))

    Private m_input As IList(Of TSource)
    Shared m_pos As Integer = 0
    Private m_current As KeyValuePair(Of Long, TSource)

    Public Sub New(ByVal input As IList(Of TSource))
        m_input = input
        m_pos = 0
        Me.disposedValue = False
    End Sub

    Public ReadOnly Property Current As KeyValuePair(Of Long, TSource) Implements IEnumerator(Of
KeyValuePair(Of Long, TSource)).Current
        Get
            Return m_current
        End Get
    End Property

    Public ReadOnly Property Current1 As Object Implements IEnumerator.Current
        Get
            Return Me.Current
        End Get
    End Property

    Public Function MoveNext() As Boolean Implements IEnumerator.MoveNext
        Dim elemIndex = Interlocked.Increment(m_pos) - 1
        If elemIndex >= m_input.Count Then
            Return False
        End If
    End Function

```

```

        m_current = New KeyValuePair(Of Long, TSource)(elemIndex, m_input(elemIndex))
        Return True
    End Function

    Public Sub Reset() Implements IEnumerator.Reset
        m_pos = 0
    End Sub

    Private disposedValue As Boolean ' To detect redundant calls

    Protected Overridable Sub Dispose(ByVal disposing As Boolean)
        If Not Me.disposedValue Then
            m_input = Nothing
            m_current = Nothing
        End If
        Me.disposedValue = True
    End Sub

    Public Sub Dispose() Implements IDisposable.Dispose
        Dispose(True)
        GC.SuppressFinalize(Me)
    End Sub

End Class

End Class

Class ConsumerClass

    Shared Sub Main()

        Console.BufferHeight = 20000
        Dim nums = Enumerable.Range(0, 2000).ToArray()

        Dim partitioner = New OrderableListPartitioner(Of Integer)(nums)

        ' Use with Parallel.ForEach
        Parallel.ForEach(partitioner, Sub(i) Console.Write("{0}:{1} ", i,
Thread.CurrentThread.ManagedThreadId))

        Console.WriteLine("PLINQ -----")

        ' create a new partitioner, since Enumerators are not reusable.
        Dim partitioner2 = New OrderableListPartitioner(Of Integer)(nums)
        ' Use with PLINQ
        Dim query = From num In partitioner2.AsParallel()
                    Where num Mod 8 = 0
                    Select num

        For Each v In query
            Console.Write("{0} ", v)
        Next

        Console.WriteLine("press any key")
        Console.ReadKey()
    End Sub
End Class

End Module

```

This is an example of chunk partitioning, with each chunk consisting of one element. By providing more elements at a time, you could reduce the contention over the lock and theoretically achieve faster performance. However, at some point, larger chunks might require additional load-balancing logic in order to keep all threads busy until all the work is done.

See also

- [Custom Partitioners for PLINQ and TPL](#)
- [How to: Implement a Partitioner for Static Partitioning](#)

How to: Implement a Partitioner for Static Partitioning

9/20/2022 • 3 minutes to read • [Edit Online](#)

The following example shows one way to implement a simple custom partitioner for PLINQ that performs static partitioning. Because the partitioner does not support dynamic partitions, it is not consumable from [Parallel.ForEach](#). This particular partitioner might provide speedup over the default range partitioner for data sources for which each element requires an increasing amount of processing time.

Example

```
// A static range partitioner for sources that require
// a linear increase in processing time for each succeeding element.
// The range sizes are calculated based on the rate of increase
// with the first partition getting the most elements and the
// last partition getting the least.
class MyPartitioner : Partitioner<int>
{
    int[] source;
    double rateOfIncrease = 0;

    public MyPartitioner(int[] source, double rate)
    {
        this.source = source;
        rateOfIncrease = rate;
    }

    public override IEnumerable<int> GetDynamicPartitions()
    {
        throw new NotImplementedException();
    }

    // Not consumable from Parallel.ForEach.
    public override bool SupportsDynamicPartitions
    {
        get
        {
            return false;
        }
    }

    public override IList<IEnumerator<int>> GetPartitions(int partitionCount)
    {
        List<IEnumerator<int>> _list = new List<IEnumerator<int>>();
        int end = 0;
        int start = 0;
        int[] nums = CalculatePartitions(partitionCount, source.Length);

        for (int i = 0; i < nums.Length; i++)
        {
            start = nums[i];
            if (i < nums.Length - 1)
                end = nums[i + 1];
            else
                end = source.Length;

            _list.Add(GetItemsForPartition(start, end));
        }

        // For demonstration.
    }
}
```

```

        Console.WriteLine("start = {0} b (end) = {1}", start, end);
    }
    return (IList<IEnumerator<int>>) _list;
}
/*
*
*
*/
// Model increasing workloads as a right triangle           B
// divided into equal areas along vertical lines.          / | |
// Each partition is taller and skinnier                  / | | |
// than the last.                                         / | | | |
//                                                       / | | | |
//                                                       / | | | |
//                                                       / | | | |
//                                                       / | | | |
A   /_____|_____|_____|_____| C
*/
private int[] CalculatePartitions(int partitionCount, int sourceLength)
{
    // Corresponds to the opposite side of angle A, which corresponds
    // to an index into the source array.
    int[] partitionLimits = new int[partitionCount];
    partitionLimits[0] = 0;

    // Represent total work as rectangle of source length times "most expensive element"
    // Note: RateOfIncrease can be factored out of equation.
    double totalWork = sourceLength * (sourceLength * rateOfIncrease);
    // Divide by two to get the triangle whose slope goes from zero on the left to "most"
    // on the right. Then divide by number of partitions to get area of each partition.
    totalWork /= 2;
    double partitionArea = totalWork / partitionCount;

    // Draw the next partitionLimit on the vertical coordinate that gives
    // an area of partitionArea * currentPartition.
    for (int i = 1; i < partitionLimits.Length; i++)
    {
        double area = partitionArea * i;

        // Solve for base given the area and the slope of the hypotenuse.
        partitionLimits[i] = (int)Math.Floor(Math.Sqrt((2 * area) / rateOfIncrease));
    }
    return partitionLimits;
}

IEnumerator<int> GetItemsForPartition(int start, int end)
{
    // For demonstration purposes. Each thread receives its own enumerator.
    Console.WriteLine("called on thread {0}", Thread.CurrentThread.ManagedThreadId);
    for (int i = start; i < end; i++)
        yield return source[i];
}

class Consumer
{
    public static void Main2()
    {
        var source = Enumerable.Range(0, 10000).ToArray();

        Stopwatch sw = Stopwatch.StartNew();
        MyPartitioner partitioner = new MyPartitioner(source, .5);

        var query = from n in partitioner.AsParallel()
                    select ProcessData(n);

        foreach (var v in query) { }
        Console.WriteLine("Processing time with custom partitioner {0}", sw.ElapsedMilliseconds);
    }
}
```

```
var source2 = Enumerable.Range(0, 10000).ToArray();

sw = Stopwatch.StartNew();

var query2 = from n in source2.AsParallel()
            select ProcessData(n);

foreach (var v in query2) { }
Console.WriteLine("Processing time with default partitioner {0}", sw.ElapsedMilliseconds);
}

// Consistent processing time for measurement purposes.
static int ProcessData(int i)
{
    Thread.Sleep(i * 1000);
    return i;
}
}
```

The partitions in this example are based on the assumption of a linear increase in processing time for each element. In the real world, it might be difficult to predict processing times in this way. If you are using a static partitioner with a specific data source, you can optimize the partitioning formula for the source, add load-balancing logic, or use a chunk partitioning approach as demonstrated in [How to: Implement Dynamic Partitions](#).

See also

- [Custom Partitioners for PLINQ and TPL](#)

Lambda Expressions in PLINQ and TPL

9/20/2022 • 3 minutes to read • [Edit Online](#)

The Task Parallel Library (TPL) contains many methods that take one of the `System.Func<TResult>` or `System.Action` family of delegates as input parameters. You use these delegates to pass in your custom program logic to the parallel loop, task or query. The code examples for TPL as well as PLINQ use lambda expressions to create instances of those delegates as inline code blocks. This topic provides a brief introduction to Func and Action and shows you how to use lambda expressions in the Task Parallel Library and PLINQ.

NOTE

For more information about delegates in general, see [Delegates](#) and [Delegates](#). For more information about lambda expressions in C# and Visual Basic, see [Lambda Expressions](#) and [Lambda Expressions](#).

Func Delegate

A `Func` delegate encapsulates a method that returns a value. In a `Func` signature, the last, or rightmost, type parameter always specifies the return type. One common cause of compiler errors is to attempt to pass in two input parameters to a `System.Func<T,TResult>`; in fact this type takes only one input parameter. .NET defines 17 versions of `Func`: `System.Func<TResult>`, `System.Func<T,TResult>`, `System.Func<T1,T2,TResult>`, and so on up through `System.Func<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16,TResult>`.

Action Delegate

A `System.Action` delegate encapsulates a method (Sub in Visual Basic) that does not return a value. In an `Action` type signature, the type parameters represent only input parameters. Like `Func`, .NET defines 17 versions of `Action`, from a version that has no type parameters up through a version that has 16 type parameters.

Example

The following example for the `Parallel.ForEach<TSource,TLocal>(IEnumerable<TSource>, Func<TLocal>, Func<TSource,ParallelLoopState,TLocal,TLocal>, Action<TLocal>)` method shows how to express both Func and Action delegates by using lambda expressions.

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;
using System.Threading.Tasks;

class ForEachWithThreadLocal
{
    // Demonstrated features:
    // Parallel.ForEach()
    // Thread-local state
    // Expected results:
    //     This example sums up the elements of an int[] in parallel.
    //     Each thread maintains a local sum. When a thread is initialized, that local sum is set to 0.
    //     On every iteration the current element is added to the local sum.
    //     When a thread is done, it safely adds its local sum to the global sum.
    //     After the loop is complete, the global sum is printed out.
    // Documentation:
    // http://msdn.microsoft.com/library/dd990270(VS.100).aspx
    static void Main()
    {
        // The sum of these elements is 40.
        int[] input = { 4, 1, 6, 2, 9, 5, 10, 3 };
        int sum = 0;

        try
        {
            Parallel.ForEach(
                input,                  // source collection
                () => 0,              // thread local initializer
                (n, loopState, localSum) => // body
                {
                    localSum += n;
                    Console.WriteLine("Thread={0}, n={1}, localSum={2}",
Thread.CurrentThread.ManagedThreadId, n, localSum);
                    return localSum;
                },
                (localSum) => Interlocked.Add(ref sum, localSum)      // thread local aggregator
            );

            Console.WriteLine("\nSum={0}", sum);
        }
        // No exception is expected in this example, but if one is still thrown from a task,
        // it will be wrapped in AggregateException and propagated to the main thread.
        catch (AggregateException e)
        {
            Console.WriteLine("Parallel.ForEach has thrown an exception. THIS WAS NOT EXPECTED.\n{0}", e);
        }
    }
}

```

```

Imports System.Threading
Imports System.Threading.Tasks

Module ForEachDemo

    ' Demonstrated features:
    ' Parallel.ForEach()
    ' Thread-local state
    ' Expected results:
    ' This example sums up the elements of an int[] in parallel.
    ' Each thread maintains a local sum. When a thread is initialized, that local sum is set to 0.
    ' On every iteration the current element is added to the local sum.
    ' When a thread is done, it safely adds its local sum to the global sum.
    ' After the loop is complete, the global sum is printed out.
    ' Documentation:
    ' http://msdn.microsoft.com/library/dd990270(VS.100).aspx
Private Sub ForEachDemo()
    ' The sum of these elements is 40.
    Dim input As Integer() = {4, 1, 6, 2, 9, 5, _
    10, 3}
    Dim sum As Integer = 0

    Try
        ' source collection
        Parallel.ForEach(input,
            Function()
                ' thread local initializer
                Return 0
            End Function,
            Function(n, loopState, localSum)
                ' body
                localSum += n
                Console.WriteLine("Thread={0}, n={1}, localSum={2}",
                    Thread.CurrentThread.ManagedThreadId, n, localSum)
                Return localSum
            End Function,
            Sub(localSum)
                ' thread local aggregator
                Interlocked.Add(sum, localSum)
            End Sub)
    Catch e As AggregateException
        ' No exception is expected in this example, but if one is still thrown from a task,
        ' it will be wrapped in AggregateException and propagated to the main thread.
        Console.WriteLine("Parallel.ForEach has thrown an exception. THIS WAS NOT EXPECTED." & vbCrLf & "{0}", e)
    End Try
End Sub

End Module

```

See also

- [Parallel Programming](#)

For Further Reading (Parallel Programming)

9/20/2022 • 2 minutes to read • [Edit Online](#)

The following resources contain additional information about parallel programming in .NET:

- The [Patterns for Parallel Programming: Understanding and Applying Parallel Patterns with the .NET Framework 4](#) document describes common parallel patterns and best practices for developing parallel components utilizing those patterns.
- The [Design Patterns for Decomposition and Coordination on Multicore Architectures](#) book describes patterns for parallel programming that use the parallel programming support introduced in the .NET Framework 4.
- The [Parallel Programming with .NET](#) blog contains many in-depth articles about parallel programming in .NET.
- The [Samples for Parallel Programming with the .NET Core & .NET Standard](#) page contains many samples that demonstrate intermediate and advanced parallel programming techniques.

See also

- [Parallel Computing Developer Center](#)
- [Parallel Programming in Visual C++](#)

Managed threading basics

9/20/2022 • 2 minutes to read • [Edit Online](#)

The first five articles of this section are designed to help you determine when to use managed threading and to explain some basic features. For information on classes that provide additional features, see [Threading Objects and Features](#) and [Overview of Synchronization Primitives](#).

The remaining articles in this section cover advanced topics, including the interaction of managed threading with the Windows operating system.

NOTE

Starting with .NET Framework 4, the Task Parallel Library and PLINQ provide APIs for task and data parallelism in multi-threaded programs. For more information, see [Parallel Programming](#).

In this section

[Threads and Threading](#)

Discusses the advantages and drawbacks of multiple threads, and outlines the scenarios in which you might create threads or use thread pool threads.

[Exceptions in Managed Threads](#)

Describes the behavior of unhandled exceptions in threads for different versions of .NET, in particular the situations in which they result in termination of the application.

[Synchronizing Data for Multithreading](#)

Describes strategies for synchronizing data in classes that will be used with multiple threads.

[Foreground and Background Threads](#)

Explains the differences between foreground and background threads.

[Managed and Unmanaged Threading in Windows](#)

Discusses the relationship between managed and unmanaged threading, lists managed equivalents for Windows threading APIs, and discusses the interaction of COM apartments and managed threads.

[Thread Local Storage: Thread-Relative Static Fields and Data Slots](#)

Describes thread-relative storage mechanisms.

Reference

[Thread](#) Provides reference documentation for the **Thread** class, which represents a managed thread, whether it came from unmanaged code or was created in a managed application.

[BackgroundWorker](#) Provides a safe way to implement multithreading in conjunction with user-interface objects.

Related sections

[Overview of Synchronization Primitives](#)

Describes the managed classes used to synchronize the activities of multiple threads.

[Managed Threading Best Practices](#)

Describes common problems with multithreading and strategies for avoiding problems.

[Parallel Programming](#)

Describes the Task Parallel Library and PLINQ which greatly simplify the work of creating asynchronous and multi-threaded .NET applications.

[System.Threading.Channels library](#)

Describes the System.Threading.Channels library, which provides a set of synchronization data structures for passing data between producers and consumers asynchronously.

Testing in .NET

9/20/2022 • 3 minutes to read • [Edit Online](#)

This article introduces the concept of testing, and illustrates how different kinds of tests can be used to validate code. There are various tools available for testing .NET applications, such as the [.NET CLI](#) or [Integrated Development Environments \(IDEs\)](#).

Test types

Having automated tests is a great way to ensure that application code does what its authors intend it to do. This article covers unit tests, integration tests, and load tests.

Unit tests

A *unit test* is a test that exercises individual software components or methods, also known as "unit of work". Unit tests should only test code within the developer's control. They do not test infrastructure concerns. Infrastructure concerns include interacting with databases, file systems, and network resources.

For more information on creating unit tests, see [Testing tools](#).

Integration tests

An *integration test* differs from a unit test in that it exercises two or more software components' ability to function together, also known as their "integration." These tests operate on a broader spectrum of the system under test, whereas unit tests focus on individual components. Often, integration tests do include infrastructure concerns.

Load tests

A *load test* aims to determine whether or not a system can handle a specified load, for example, the number of concurrent users using an application and the app's ability to handle interactions responsively. For more information on load testing of web applications, see [ASP.NET Core load/stress testing](#).

Test considerations

Keep in mind there are [best practices](#) for writing tests. For example, [Test Driven Development \(TDD\)](#) is when a unit test is written before the code it's meant to check. TDD is like creating an outline for a book before you write it. It is meant to help developers write simpler, more readable, and efficient code.

Testing tools

.NET is a multi-language development platform, and you can write various test types for [C#](#), [F#](#), and [Visual Basic](#). For each of these languages, you can choose between several test frameworks.

xUnit

[xUnit](#) is a free, open source, community-focused unit testing tool for .NET. Written by the original inventor of NUnit v2, xUnit.net is the latest technology for unit testing .NET apps. xUnit.net works with ReSharper, CodeRush, TestDriven.NET, and [Xamarin](#). It is a project of the [.NET Foundation](#) and operates under their code of conduct.

For more information, see the following resources:

- [Unit testing with C#](#)
- [Unit testing with F#](#)
- [Unit testing with Visual Basic](#)

NUnit

[NUnit](#) is a unit-testing framework for all .NET languages. Initially ported from JUnit, the current production release has been rewritten with many new features and support for a wide range of .NET platforms. It is a project of the [.NET Foundation](#).

For more information, see the following resources:

- [Unit testing with C#](#)
- [Unit testing with F#](#)
- [Unit testing with Visual Basic](#)

MSTest

[MSTest](#) is the Microsoft test framework for all .NET languages. It's extensible and works with both .NET CLI and Visual Studio. For more information, see the following resources:

- [Unit testing with C#](#)
- [Unit testing with F#](#)
- [Unit testing with Visual Basic](#)

.NET CLI

You can run a solutions unit tests from the [.NET CLI](#), with the `dotnet test` command. The .NET CLI exposes a majority of the functionality that [Integrated Development Environments \(IDEs\)](#) make available through user interfaces. The .NET CLI is cross-platform and available to use as part of continuous integration and delivery pipelines. The .NET CLI is used with scripted processes to automate common tasks.

IDE

Whether you're using Visual Studio, Visual Studio for Mac, or Visual Studio Code, there are graphical user interfaces for testing functionality. There are more features available to IDEs than the CLI, for example [Live Unit Testing](#). For more information, see [Including and excluding tests with Visual Studio](#).

See also

For more information, see the following articles:

- [Unit testing best practices with .NET](#)
- [Integration tests in ASP.NET Core](#)
- [Running selective unit tests](#)
- [Use code coverage for unit testing](#)

Unit testing best practices with .NET Core and .NET Standard

9/20/2022 • 14 minutes to read • [Edit Online](#)

There are numerous benefits to writing unit tests; they help with regression, provide documentation, and facilitate good design. However, hard to read and brittle unit tests can wreak havoc on your code base. This article describes some best practices regarding unit test design for your .NET Core and .NET Standard projects.

In this guide, you'll learn some best practices when writing unit tests to keep your tests resilient and easy to understand.

By [John Reese](#) with special thanks to [Roy Osherove](#)

Why unit test?

Less time performing functional tests

Functional tests are expensive. They typically involve opening up the application and performing a series of steps that you (or someone else), must follow in order to validate the expected behavior. These steps may not always be known to the tester, which means they will have to reach out to someone more knowledgeable in the area in order to carry out the test. Testing itself could take seconds for trivial changes, or minutes for larger changes. Lastly, this process must be repeated for every change that you make in the system.

Unit tests, on the other hand, take milliseconds, can be run at the press of a button, and don't necessarily require any knowledge of the system at large. Whether or not the test passes or fails is up to the test runner, not the individual.

Protection against regression

Regression defects are defects that are introduced when a change is made to the application. It is common for testers to not only test their new feature but also features that existed beforehand in order to verify that previously implemented features still function as expected.

With unit testing, it's possible to rerun your entire suite of tests after every build or even after you change a line of code. Giving you confidence that your new code does not break existing functionality.

Executable documentation

It may not always be obvious what a particular method does or how it behaves given a certain input. You may ask yourself: How does this method behave if I pass it a blank string? Null?

When you have a suite of well-named unit tests, each test should be able to clearly explain the expected output for a given input. In addition, it should be able to verify that it actually works.

Less coupled code

When code is tightly coupled, it can be difficult to unit test. Without creating unit tests for the code that you're writing, coupling may be less apparent.

Writing tests for your code will naturally decouple your code, because it would be more difficult to test otherwise.

Characteristics of a good unit test

- **Fast.** It is not uncommon for mature projects to have thousands of unit tests. Unit tests should take very little

time to run. Milliseconds.

- **Isolated**. Unit tests are standalone, can be run in isolation, and have no dependencies on any outside factors such as a file system or database.
- **Repeatable**. Running a unit test should be consistent with its results, that is, it always returns the same result if you do not change anything in between runs.
- **Self-Checking**. The test should be able to automatically detect if it passed or failed without any human interaction.
- **Timely**. A unit test should not take a disproportionately long time to write compared to the code being tested. If you find testing the code taking a large amount of time compared to writing the code, consider a design that is more testable.

Code coverage

A high code coverage percentage is often associated with a higher quality of code. However, the measurement itself *cannot* determine the quality of code. Setting an overly ambitious code coverage percentage goal can be counterproductive. Imagine a complex project with thousands of conditional branches, and imagine that you set a goal of 95% code coverage. Currently the project maintains 90% code coverage. The amount of time it takes to account for all of the edge cases in the remaining 5% could be a massive undertaking, and the value proposition quickly diminishes.

A high code coverage percentage is not an indicator of success, nor does it imply high code quality. It just represents the amount of code that is covered by unit tests. For more information, see [unit testing code coverage](#).

Let's speak the same language

The term *mock* is unfortunately often misused when talking about testing. The following points define the most common types of *fakes* when writing unit tests:

Fake - A fake is a generic term that can be used to describe either a stub or a mock object. Whether it's a stub or a mock depends on the context in which it's used. So in other words, a fake can be a stub or a mock.

Mock - A mock object is a fake object in the system that decides whether or not a unit test has passed or failed. A mock starts out as a Fake until it's asserted against.

Stub - A stub is a controllable replacement for an existing dependency (or collaborator) in the system. By using a stub, you can test your code without dealing with the dependency directly. By default, a stub starts out as a fake.

Consider the following code snippet:

```
var mockOrder = new MockOrder();
var purchase = new Purchase(mockOrder);

purchase.ValidateOrders();

Assert.True(purchase.CanBeShipped);
```

This would be an example of stub being referred to as a mock. In this case, it is a stub. You're just passing in the Order as a means to be able to instantiate `Purchase` (the system under test). The name `MockOrder` is also misleading because again, the order is not a mock.

A better approach would be

```
var stubOrder = new FakeOrder();
var purchase = new Purchase(stubOrder);

purchase.ValidateOrders();

Assert.True(purchase.CanBeShipped);
```

By renaming the class to `FakeOrder`, you've made the class a lot more generic, the class can be used as a mock or a stub. Whichever is better for the test case. In the above example, `FakeOrder` is used as a stub. You're not using the `FakeOrder` in any shape or form during the assert. `FakeOrder` was passed into the `Purchase` class to satisfy the requirements of the constructor.

To use it as a Mock, you could do something like this

```
var mockOrder = new FakeOrder();
var purchase = new Purchase(mockOrder);

purchase.ValidateOrders();

Assert.True(mockOrder.Validated);
```

In this case, you are checking a property on the Fake (asserting against it), so in the above code snippet, the `mockOrder` is a Mock.

IMPORTANT

It's important to get this terminology correct. If you call your stubs "mocks", other developers are going to make false assumptions about your intent.

The main thing to remember about mocks versus stubs is that mocks are just like stubs, but you assert against the mock object, whereas you do not assert against a stub.

Best practices

Try not to introduce dependencies on infrastructure when writing unit tests. These make the tests slow and brittle and should be reserved for integration tests. You can avoid these dependencies in your application by following the [Explicit Dependencies Principle](#) and using [Dependency Injection](#). You can also keep your unit tests in a separate project from your integration tests. This ensures your unit test project doesn't have references to or dependencies on infrastructure packages.

Naming your tests

The name of your test should consist of three parts:

- The name of the method being tested.
- The scenario under which it's being tested.
- The expected behavior when the scenario is invoked.

Why?

- Naming standards are important because they explicitly express the intent of the test.

Tests are more than just making sure your code works, they also provide documentation. Just by looking at the suite of unit tests, you should be able to infer the behavior of your code without even looking at the code itself. Additionally, when tests fail, you can see exactly which scenarios do not meet your expectations.

Bad:

```
[Fact]
public void Test_Single()
{
    var stringCalculator = new StringCalculator();

    var actual = stringCalculator.Add("0");

    Assert.Equal(0, actual);
}
```

Better:

```
[Fact]
public void Add_SingleNumber_ReturnsSameNumber()
{
    var stringCalculator = new StringCalculator();

    var actual = stringCalculator.Add("0");

    Assert.Equal(0, actual);
}
```

Arranging your tests

Arrange, Act, Assert is a common pattern when unit testing. As the name implies, it consists of three main actions:

- *Arrange* your objects, creating and setting them up as necessary.
- *Act* on an object.
- *Assert* that something is as expected.

Why?

- Clearly separates what is being tested from the *arrange* and *assert* steps.
- Less chance to intermix assertions with "Act" code.

Readability is one of the most important aspects when writing a test. Separating each of these actions within the test clearly highlight the dependencies required to call your code, how your code is being called, and what you are trying to assert. While it may be possible to combine some steps and reduce the size of your test, the primary goal is to make the test as readable as possible.

Bad:

```
[Fact]
public void Add_EmptyString_ReturnsZero()
{
    // Arrange
    var stringCalculator = new StringCalculator();

    // Assert
    Assert.Equal(0, stringCalculator.Add(""));
}
```

Better:

```
[Fact]
public void Add_EmptyString_ReturnsZero()
{
    // Arrange
    var stringCalculator = new StringCalculator();

    // Act
    var actual = stringCalculator.Add("");

    // Assert
    Assert.Equal(0, actual);
}
```

Write minimally passing tests

The input to be used in a unit test should be the simplest possible in order to verify the behavior that you are currently testing.

Why?

- Tests become more resilient to future changes in the codebase.
- Closer to testing behavior over implementation.

Tests that include more information than required to pass the test have a higher chance of introducing errors into the test and can make the intent of the test less clear. When writing tests, you want to focus on the behavior. Setting extra properties on models or using non-zero values when not required, only detracts from what you are trying to prove.

Bad:

```
[Fact]
public void Add_SingleNumber_ReturnsSameNumber()
{
    var stringCalculator = new StringCalculator();

    var actual = stringCalculator.Add("42");

    Assert.Equal(42, actual);
}
```

Better:

```
[Fact]
public void Add_SingleNumber_ReturnsSameNumber()
{
    var stringCalculator = new StringCalculator();

    var actual = stringCalculator.Add("0");

    Assert.Equal(0, actual);
}
```

Avoid magic strings

Naming variables in unit tests is as important, if not more important, than naming variables in production code. Unit tests should not contain magic strings.

Why?

- Prevents the need for the reader of the test to inspect the production code in order to figure out what makes the value special.
- Explicitly shows what you're trying to *prove* rather than trying to *accomplish*.

Magic strings can cause confusion to the reader of your tests. If a string looks out of the ordinary, they may wonder why a certain value was chosen for a parameter or return value. This may lead them to take a closer look at the implementation details, rather than focus on the test.

TIP

When writing tests, you should aim to express as much intent as possible. In the case of magic strings, a good approach is to assign these values to constants.

Bad:

```
[Fact]
public void Add_BigNumber_ThrowsException()
{
    var stringCalculator = new StringCalculator();

    Action actual = () => stringCalculator.Add("1001");

    Assert.Throws<OverflowException>(actual);
}
```

Better:

```
[Fact]
void Add_MaximumSumResult_ThrowsOverflowException()
{
    var stringCalculator = new StringCalculator();
    const string MAXIMUM_RESULT = "1001";

    Action actual = () => stringCalculator.Add(MAXIMUM_RESULT);

    Assert.Throws<OverflowException>(actual);
}
```

Avoid logic in tests

When writing your unit tests avoid manual string concatenation and logical conditions such as `if`, `while`, `for`, `switch`, etc.

Why?

- Less chance to introduce a bug inside of your tests.
- Focus on the end result, rather than implementation details.

When you introduce logic into your test suite, the chance of introducing a bug into it increases dramatically. The last place that you want to find a bug is within your test suite. You should have a high level of confidence that your tests work, otherwise, you will not trust them. Tests that you do not trust, do not provide any value. When a test fails, you want to have a sense that something is actually wrong with your code and that it cannot be ignored.

TIP

If logic in your test seems unavoidable, consider splitting the test up into two or more different tests.

Bad:

```
[Fact]
public void Add_MultipleNumbers_ReturnsCorrectResults()
{
    var stringCalculator = new StringCalculator();
    var expected = 0;
    var testCases = new[]
    {
        "0,0,0",
        "0,1,2",
        "1,2,3"
    };

    foreach (var test in testCases)
    {
        Assert.Equal(expected, stringCalculator.Add(test));
        expected += 3;
    }
}
```

Better:

```
[Theory]
[InlineData("0,0,0", 0)]
[InlineData("0,1,2", 3)]
[InlineData("1,2,3", 6)]
public void Add_MultipleNumbers_ReturnsSumOfNumbers(string input, int expected)
{
    var stringCalculator = new StringCalculator();

    var actual = stringCalculator.Add(input);

    Assert.Equal(expected, actual);
}
```

Prefer helper methods to setup and teardown

If you require a similar object or state for your tests, prefer a helper method than leveraging `Setup` and `Teardown` attributes if they exist.

Why?

- Less confusion when reading the tests since all of the code is visible from within each test.
- Less chance of setting up too much or too little for the given test.
- Less chance of sharing state between tests, which creates unwanted dependencies between them.

In unit testing frameworks, `Setup` is called before each and every unit test within your test suite. While some may see this as a useful tool, it generally ends up leading to bloated and hard to read tests. Each test will generally have different requirements in order to get the test up and running. Unfortunately, `Setup` forces you to use the exact same requirements for each test.

NOTE

xUnit has removed both `SetUp` and `TearDown` as of version 2.x

Bad:

```
private readonly StringCalculator stringCalculator;
public StringCalculatorTests()
{
    stringCalculator = new StringCalculator();
}
```

```
// more tests...
```

```
[Fact]
public void Add_TwoNumbers_ReturnsSumOfNumbers()
{
    var result = stringCalculator.Add("0,1");

    Assert.Equal(1, result);
}
```

Better:

```
[Fact]
public void Add_TwoNumbers_ReturnsSumOfNumbers()
{
    var stringCalculator = CreateDefaultStringCalculator();

    var actual = stringCalculator.Add("0,1");

    Assert.Equal(1, actual);
}
```

```
// more tests...
```

```
private StringCalculator CreateDefaultStringCalculator()
{
    return new StringCalculator();
}
```

Avoid multiple acts

When writing your tests, try to only include one Act per test. Common approaches to using only one act include:

- Create a separate test for each act.
- Use parameterized tests.

Why?

- When the test fails it is not clear which Act is failing.
- Ensures the test is focussed on just a single case.
- Gives you the entire picture as to why your tests are failing.

Multiple Acts need to be individually Asserted and it is not guaranteed that all of the Asserts will be executed. In most unit testing frameworks, once an Assert fails in a unit test, the proceeding tests are automatically considered to be failing. This can be confusing as functionality that is actually working, will be shown as failing.

Bad:

```
[Fact]
public void Add_EmptyEntries_ShouldBeTreatedAsZero()
{
    // Act
    var actual1 = stringCalculator.Add("");
    var actual2 = stringCalculator.Add(",,");

    // Assert
    Assert.Equal(0, actual1);
    Assert.Equal(0, actual2);
}
```

Better:

```
[Theory]
[InlineData("", 0)]
[InlineData(",", 0)]
public void Add_EmptyEntries_ShouldBeTreatedAsZero(string input, int expected)
{
    // Arrange
    var stringCalculator = new StringCalculator();

    // Act
    var actual = stringCalculator.Add(input);

    // Assert
    Assert.Equal(expected, actual);
}
```

Validate private methods by unit testing public methods

In most cases, there should not be a need to test a private method. Private methods are an implementation detail. You can think of it this way: private methods never exist in isolation. At some point, there is going to be a public facing method that calls the private method as part of its implementation. What you should care about is the end result of the public method that calls into the private one.

Consider the following case

```
public string ParseLogLine(string input)
{
    var sanitizedInput = TrimInput(input);
    return sanitizedInput;
}

private string TrimInput(string input)
{
    return input.Trim();
}
```

Your first reaction may be to start writing a test for `TrimInput` because you want to make sure that the method is working as expected. However, it is entirely possible that `ParseLogLine` manipulates `sanitizedInput` in such a way that you do not expect, rendering a test against `TrimInput` useless.

The real test should be done against the public facing method `ParseLogLine` because that is what you should ultimately care about.

```

public void ParseLogLine_StartsAndEndsWithSpace_ReturnsTrimmedResult()
{
    var parser = new Parser();

    var result = parser.ParseLogLine(" a ");

    Assert.Equals("a", result);
}

```

With this viewpoint, if you see a private method, find the public method and write your tests against that method. Just because a private method returns the expected result, does not mean the system that eventually calls the private method uses the result correctly.

Stub static references

One of the principles of a unit test is that it must have full control of the system under test. This can be problematic when production code includes calls to static references (for example, `DateTime.Now`). Consider the following code

```

public int GetDiscountedPrice(int price)
{
    if (DateTime.Now.DayOfWeek == DayOfWeek.Tuesday)
    {
        return price / 2;
    }
    else
    {
        return price;
    }
}

```

How can this code possibly be unit tested? You may try an approach such as

```

public void GetDiscountedPrice_NotTuesday_ReturnsFullPrice()
{
    var priceCalculator = new PriceCalculator();

    var actual = priceCalculator.GetDiscountedPrice(2);

    Assert.Equals(2, actual)
}

public void GetDiscountedPrice_OnTuesday_ReturnsHalfPrice()
{
    var priceCalculator = new PriceCalculator();

    var actual = priceCalculator.GetDiscountedPrice(2);

    Assert.Equals(1, actual);
}

```

Unfortunately, you will quickly realize that there are a couple problems with your tests.

- If the test suite is run on a Tuesday, the second test will pass, but the first test will fail.
- If the test suite is run on any other day, the first test will pass, but the second test will fail.

To solve these problems, you'll need to introduce a *seam* into your production code. One approach is to wrap the code that you need to control in an interface and have the production code depend on that interface.

```

public interface IDateTimeProvider
{
    DayOfWeek DayOfWeek();
}

public int GetDiscountedPrice(int price, IDateTimeProvider dateTimeProvider)
{
    if (dateTimeProvider.DayOfWeek() == DayOfWeek.Tuesday)
    {
        return price / 2;
    }
    else
    {
        return price;
    }
}

```

Your test suite now becomes

```

public void GetDiscountedPrice_NotTuesday_ReturnsFullPrice()
{
    var priceCalculator = new PriceCalculator();
    var dateTimeProviderStub = new Mock<IDateTimeProvider>();
    dateTimeProviderStub.Setup(dtp => dtp.DayOfWeek()).Returns(DayOfWeek.Monday);

    var actual = priceCalculator.GetDiscountedPrice(2, dateTimeProviderStub);

    Assert.Equals(2, actual);
}

public void GetDiscountedPrice_OnTuesday_ReturnsHalfPrice()
{
    var priceCalculator = new PriceCalculator();
    var dateTimeProviderStub = new Mock<IDateTimeProvider>();
    dateTimeProviderStub.Setup(dtp => dtp.DayOfWeek()).Returns(DayOfWeek.Tuesday);

    var actual = priceCalculator.GetDiscountedPrice(2, dateTimeProviderStub);

    Assert.Equals(1, actual);
}

```

Now the test suite has full control over `DateTime.Now` and can stub any value when calling into the method.

Unit testing C# in .NET Core using dotnet test and xUnit

9/20/2022 • 4 minutes to read • [Edit Online](#)

This tutorial shows how to build a solution containing a unit test project and source code project. To follow the tutorial using a pre-built solution, [view or download the sample code](#). For download instructions, see [Samples and Tutorials](#).

Create the solution

In this section, a solution is created that contains the source and test projects. The completed solution has the following directory structure:

```
/unit-testing-using-dotnet-test
    unit-testing-using-dotnet-test.sln
    /PrimeService
        PrimeService.cs
        PrimeService.csproj
    /PrimeService.Tests
        PrimeService_IsPrimeShould.cs
        PrimeServiceTests.csproj
```

The following instructions provide the steps to create the test solution. See [Commands to create test solution](#) for instructions to create the test solution in one step.

- Open a shell window.
- Run the following command:

```
dotnet new sln -o unit-testing-using-dotnet-test
```

The `dotnet new sln` command creates a new solution in the *unit-testing-using-dotnet-test* directory.

- Change directory to the *unit-testing-using-dotnet-test* folder.
- Run the following command:

```
dotnet new classlib -o PrimeService
```

The `dotnet new classlib` command creates a new class library project in the *PrimeService* folder. The new class library will contain the code to be tested.

- Rename *Class1.cs* to *PrimeService.cs*.
- Replace the code in *PrimeService.cs* with the following code:

```
using System;

namespace Prime.Services
{
    public class PrimeService
    {
        public bool IsPrime(int candidate)
        {
            throw new NotImplementedException("Not implemented.");
        }
    }
}
```

- The preceding code:
 - Throws a [NotImplementedException](#) with a message indicating it's not implemented.
 - Is updated later in the tutorial.
- In the *unit-testing-using-dotnet-test* directory, run the following command to add the class library project to the solution:

```
dotnet sln add ./PrimeService/PrimeService.csproj
```

- Create the *PrimeService.Tests* project by running the following command:

```
dotnet new xunit -o PrimeService.Tests
```

- The preceding command:
 - Creates the *PrimeService.Tests* project in the *PrimeService.Tests* directory. The test project uses [xUnit](#) as the test library.
 - Configures the test runner by adding the following `<PackageReference />` elements to the project file:
 - `Microsoft.NET.Test.Sdk`
 - `xunit`
 - `xunit.runner.visualstudio`
 - `coverlet.collector`
- Add the test project to the solution file by running the following command:

```
dotnet sln add ./PrimeService.Tests/PrimeService.Tests.csproj
```

- Add the `PrimeService` class library as a dependency to the *PrimeService.Tests* project:

```
dotnet add ./PrimeService.Tests/PrimeService.Tests.csproj reference
./PrimeService/PrimeService.csproj
```

Commands to create the solution

This section summarizes all the commands in the previous section. Skip this section if you've completed the steps in the previous section.

The following commands create the test solution on a windows machine. For macOS and Unix, update the `ren` command to the OS version of `mv` to rename a file:

```
dotnet new sln -o unit-testing-using-dotnet-test
cd unit-testing-using-dotnet-test
dotnet new classlib -o PrimeService
ren .\PrimeService\Class1.cs PrimeService.cs
dotnet sln add ./PrimeService/PrimeService.csproj
dotnet new xunit -o PrimeService.Tests
dotnet add ./PrimeService.Tests/PrimeService.Tests.csproj reference ./PrimeService/PrimeService.csproj
dotnet sln add ./PrimeService.Tests/PrimeService.Tests.csproj
```

Follow the instructions for "Replace the code in *PrimeService.cs* with the following code" in the previous section.

Create a test

A popular approach in test driven development (TDD) is to write a test before implementing the target code. This tutorial uses the TDD approach. The `IsPrime` method is callable, but not implemented. A test call to `IsPrime` fails. With TDD, a test is written that is known to fail. The target code is updated to make the test pass. You keep repeating this approach, writing a failing test and then updating the target code to pass.

Update the *PrimeService.Tests* project:

- Delete *PrimeService.Tests/UnitTest1.cs*.
- Create a *PrimeService.Tests/PrimeService_IsPrimeShould.cs* file.
- Replace the code in *PrimeService_IsPrimeShould.cs* with the following code:

```
using Xunit;
using Prime.Services;

namespace Prime.UnitTests.Services
{
    public class PrimeService_IsPrimeShould
    {
        [Fact]
        public void IsPrime_InputIs1_ReturnFalse()
        {
            var primeService = new PrimeService();
            bool result = primeService.IsPrime(1);

            Assert.False(result, "1 should not be prime");
        }
    }
}
```

The `[Fact]` attribute declares a test method that's run by the test runner. From the *PrimeService.Tests* folder, run `dotnet test`. The `dotnet test` command builds both projects and runs the tests. The xUnit test runner contains the program entry point to run the tests. `dotnet test` starts the test runner using the unit test project.

The test fails because `IsPrime` hasn't been implemented. Using the TDD approach, write only enough code so this test passes. Update `IsPrime` with the following code:

```
public bool IsPrime(int candidate)
{
    if (candidate == 1)
    {
        return false;
    }
    throw new NotImplementedException("Not fully implemented.");
}
```

Run `dotnet test`. The test passes.

Add more tests

Add prime number tests for 0 and -1. You could copy the test created in the preceding step and make copies of the following code to test 0 and -1. But don't do it, as there's a better way.

```
var primeService = new PrimeService();
bool result = primeService.IsPrime(1);

Assert.False(result, "1 should not be prime");
```

Copying test code when only a parameter changes results in code duplication and test bloat. The following xUnit attributes enable writing a suite of similar tests:

- `[Theory]` represents a suite of tests that execute the same code but have different input arguments.
- `[InlineData]` attribute specifies values for those inputs.

Rather than creating new tests, apply the preceding xUnit attributes to create a single theory. Replace the following code:

```
[Fact]
public void IsPrime_InputIs1_ReturnFalse()
{
    var primeService = new PrimeService();
    bool result = primeService.IsPrime(1);

    Assert.False(result, "1 should not be prime");
}
```

with the following code:

```
[Theory]
[InlineData(-1)]
[InlineData(0)]
[InlineData(1)]
public void IsPrime_ValuesLessThan2_ReturnFalse(int value)
{
    var result = _primeService.IsPrime(value);

    Assert.False(result, $"{value} should not be prime");
}
```

In the preceding code, `[Theory]` and `[InlineData]` enable testing several values less than two. Two is the smallest prime number.

Add the following code after the class declaration and before the `[Theory]` attribute:

```
private readonly PrimeService _primeService;

public PrimeService_IsPrimeShould()
{
    _primeService = new PrimeService();
```

Run `dotnet test`, and two of the tests fail. To make all of the tests pass, update the `IsPrime` method with the following code:

```
public bool IsPrime(int candidate)
{
    if (candidate < 2)
    {
        return false;
    }
    throw new NotImplementedException("Not fully implemented.");
}
```

Following the TDD approach, add more failing tests, then update the target code. See the [finished version of the tests](#) and the [complete implementation of the library](#).

The completed `IsPrime` method is not an efficient algorithm for testing primality.

Additional resources

- [xUnit.net official site](#)
- [Testing controller logic in ASP.NET Core](#)
- [`dotnet add reference`](#)

Unit testing F# libraries in .NET Core using dotnet test and xUnit

9/20/2022 • 4 minutes to read • [Edit Online](#)

This tutorial takes you through an interactive experience building a sample solution step-by-step to learn unit testing concepts. If you prefer to follow the tutorial using a pre-built solution, [view or download the sample code](#) before you begin. For download instructions, see [Samples and Tutorials](#).

This article is about testing a .NET Core project. If you're testing an ASP.NET Core project, see [Integration tests in ASP.NET Core](#).

Creating the source project

Open a shell window. Create a directory called *unit-testing-with-fsharp* to hold the solution. Inside this new directory, run `dotnet new sln` to create a new solution. This makes it easier to manage both the class library and the unit test project. Inside the solution directory, create a *MathService* directory. The directory and file structure thus far is shown below:

```
/unit-testing-with-fsharp
  unit-testing-with-fsharp.sln
  /MathService
```

Make *MathService* the current directory, and run `dotnet new classlib -lang "F#"` to create the source project. You'll create a failing implementation of the math service:

```
module MyMath =
    let squaresOfOdds xs = raise (System.NotImplementedException("You haven't written a test yet!"))
```

Change the directory back to the *unit-testing-with-fsharp* directory. Run `dotnet sln add .\MathService\MathService.fsproj` to add the class library project to the solution.

Creating the test project

Next, create the *MathService.Tests* directory. The following outline shows the directory structure:

```
/unit-testing-with-fsharp
  unit-testing-with-fsharp.sln
  /MathService
    Source Files
    MathService.fsproj
  /MathService.Tests
```

Make the *MathService.Tests* directory the current directory and create a new project using `dotnet new xunit -lang "F#"`. This creates a test project that uses xUnit as the test library. The generated template configures the test runner in the *MathServiceTests.fsproj*.

```
<ItemGroup>
  <PackageReference Include="Microsoft.NET.Test.Sdk" Version="15.3.0-preview-20170628-02" />
  <PackageReference Include="xunit" Version="2.2.0" />
  <PackageReference Include="xunit.runner.visualstudio" Version="2.2.0" />
</ItemGroup>
```

The test project requires other packages to create and run unit tests. `dotnet new` in the previous step added xUnit and the xUnit runner. Now, add the `MathService` class library as another dependency to the project. Use the `dotnet add reference` command:

```
dotnet add reference ../MathService/MathService.fsproj
```

You can see the entire file in the [samples repository](#) on GitHub.

You have the following final solution layout:

```
/unit-testing-with-fsharp
  unit-testing-with-fsharp.sln
  /MathService
    Source Files
    MathService.fsproj
  /MathService.Tests
    Test Source Files
    MathServiceTests.fsproj
```

Execute `dotnet sln add .\MathService.Tests\MathService.Tests.fsproj` in the `unit-testing-with-fsharp` directory.

Creating the first test

You write one failing test, make it pass, then repeat the process. Open `Tests.fs` and add the following code:

```
[<Fact>]
let ``My test`` () =
  Assert.True(true)

[<Fact>]
let ``Fail every time`` () = Assert.True(false)
```

The `[<Fact>]` attribute denotes a test method that is run by the test runner. From the `unit-testing-with-fsharp`, execute `dotnet test` to build the tests and the class library and then run the tests. The xUnit test runner contains the program entry point to run your tests. `dotnet test` starts the test runner using the unit test project you've created.

These two tests show the most basic passing and failing tests. `My test` passes, and `Fail every time` fails. Now, create a test for the `squaresOfOdds` method. The `squaresOfOdds` method returns a sequence of the squares of all odd integer values that are part of the input sequence. Rather than trying to write all of those functions at once, you can iteratively create tests that validate the functionality. Making each test pass means creating the necessary functionality for the method.

The simplest test we can write is to call `squaresOfOdds` with all even numbers, where the result should be an empty sequence of integers. Here's that test:

```
[<Fact>]
let ``Sequence of Evens returns empty collection`` () =
    let expected = Seq.empty<int>
    let actual = MyMath.squaresOfOdds [2; 4; 6; 8; 10]
    Assert.Equal<Collections.Generic.IEnumerable<int>>(expected, actual)
```

Your test fails. You haven't created the implementation yet. Make this test pass by writing the simplest code in the `MathService` class that works:

```
let squaresOfOdds xs =
    Seq.empty<int>
```

In the `unit-testing-with-fsharp` directory, run `dotnet test` again. The `dotnet test` command runs a build for the `MathService` project and then for the `MathService.Tests` project. After building both projects, it runs this single test. It passes.

Completing the requirements

Now that you've made one test pass, it's time to write more. The next simple case works with a sequence whose only odd number is `1`. The number 1 is easier because the square of 1 is 1. Here's that next test:

```
[<Fact>]
let ``Sequences of Ones and Evens returns Ones`` () =
    let expected = [1; 1; 1; 1]
    let actual = MyMath.squaresOfOdds [2; 1; 4; 1; 6; 1; 8; 1; 10]
    Assert.Equal<Collections.Generic.IEnumerable<int>>(expected, actual)
```

Executing `dotnet test` runs your tests and shows you that the new test fails. Now, update the `squaresOfOdds` method to handle this new test. You filter all the even numbers out of the sequence to make this test pass. You can do that by writing a small filter function and using `Seq.filter`:

```
let private isOdd x = x % 2 <> 0

let squaresOfOdds xs =
    xs
    |> Seq.filter isOdd
```

There's one more step to go: square each of the odd numbers. Start by writing a new test:

```
[<Fact>]
let ``SquaresOfOdds works`` () =
    let expected = [1; 9; 25; 49; 81]
    let actual = MyMath.squaresOfOdds [1; 2; 3; 4; 5; 6; 7; 8; 9; 10]
    Assert.Equal(expected, actual)
```

You can fix the test by piping the filtered sequence through a map operation to compute the square of each odd number:

```
let private square x = x * x
let private isOdd x = x % 2 <> 0

let squaresOfOdds xs =
    xs
    |> Seq.filter isOdd
    |> Seq.map square
```

You've built a small library and a set of unit tests for that library. You've structured the solution so that adding new packages and tests is part of the normal workflow. You've concentrated most of your time and effort on solving the goals of the application.

See also

- [dotnet new](#)
- [dotnet sln](#)
- [dotnet add reference](#)
- [dotnet test](#)

Unit testing Visual Basic .NET Core libraries using dotnet test and xUnit

9/20/2022 • 4 minutes to read • [Edit Online](#)

This tutorial shows how to build a solution containing a unit test project and library project. To follow the tutorial using a pre-built solution, [view or download the sample code](#). For download instructions, see [Samples and Tutorials](#).

Create the solution

In this section, a solution is created that contains the source and test projects. The completed solution has the following directory structure:

```
/unit-testing-using-dotnet-test
    unit-testing-using-dotnet-test.sln
    /PrimeService
        PrimeService.vb
        PrimeService.vbproj
    /PrimeService.Tests
        PrimeService_IsPrimeShould.vb
        PrimeServiceTests.vbproj
```

The following instructions provide the steps to create the test solution. See [Commands to create test solution](#) for instructions to create the test solution in one step.

- Open a shell window.
- Run the following command:

```
dotnet new sln -o unit-testing-using-dotnet-test
```

The `dotnet new sln` command creates a new solution in the *unit-testing-using-dotnet-test* directory.

- Change directory to the *unit-testing-using-dotnet-test* folder.
- Run the following command:

```
dotnet new classlib -o PrimeService --lang VB
```

The `dotnet new classlib` command creates a new class library project in the *PrimeService* folder. The new class library will contain the code to be tested.

- Rename *Class1.vb* to *PrimeService.vb*.
- Replace the code in *PrimeService.vb* with the following code:

```
Imports System

Namespace Prime.Services
    Public Class PrimeService
        Public Function IsPrime(candidate As Integer) As Boolean
            Throw New NotImplementedException("Not implemented.")
        End Function
    End Class
End Namespace
```

- The preceding code:
 - Throws a [NotImplementedException](#) with a message indicating it's not implemented.
 - Is updated later in the tutorial.
- In the *unit-testing-using-dotnet-test* directory, run the following command to add the class library project to the solution:

```
dotnet sln add ./PrimeService/PrimeService.vbproj
```

- Create the *PrimeService.Tests* project by running the following command:

```
dotnet new xunit -o PrimeService.Tests
```

- The preceding command:
 - Creates the *PrimeService.Tests* project in the *PrimeService.Tests* directory. The test project uses [xUnit](#) as the test library.
 - Configures the test runner by adding the following `<PackageReference />` elements to the project file:
 - "Microsoft.NET.Test.Sdk"
 - "xunit"
 - "xunit.runner.visualstudio"
- Add the test project to the solution file by running the following command:

```
dotnet sln add ./PrimeService.Tests/PrimeService.Tests.vbproj
```

- Add the `PrimeService` class library as a dependency to the *PrimeService.Tests* project:

```
dotnet add ./PrimeService.Tests/PrimeService.Tests.vbproj reference
./PrimeService/PrimeService.vbproj
```

Commands to create the solution

This section summarizes all the commands in the previous section. Skip this section if you've completed the steps in the previous section.

The following commands create the test solution on a windows machine. For macOS and Unix, update the `ren` command to the OS version of `ren` to rename a file:

```
dotnet new sln -o unit-testing-using-dotnet-test
cd unit-testing-using-dotnet-test
dotnet new classlib -o PrimeService
ren .\PrimeService\Class1.vb PrimeService.vb
dotnet sln add ./PrimeService/PrimeService.vbproj
dotnet new xunit -o PrimeService.Tests
dotnet add ./PrimeService.Tests/PrimeService.Tests.vbproj reference ./PrimeService/PrimeService.vbproj
dotnet sln add ./PrimeService.Tests/PrimeService.Tests.vbproj
```

Follow the instructions for "Replace the code in *PrimeService.vb* with the following code" in the previous section.

Create a test

A popular approach in test driven development (TDD) is to write a test before implementing the target code. This tutorial uses the TDD approach. The `IsPrime` method is callable, but not implemented. A test call to `IsPrime` fails. With TDD, a test is written that is known to fail. The target code is updated to make the test pass. You keep repeating this approach, writing a failing test and then updating the target code to pass.

Update the *PrimeService.Tests* project:

- Delete *PrimeService.Tests/UnitTest1.vb*.
- Create a *PrimeService.Tests/PrimeService_IsPrimeShould.vb* file.
- Replace the code in *PrimeService_IsPrimeShould.vb* with the following code:

```
Imports Xunit

Namespace PrimeService.Tests
    Public Class PrimeService_IsPrimeShould
        Private ReadOnly _primeService As Prime.Services.PrimeService

        Public Sub New()
            _primeService = New Prime.Services.PrimeService()
        End Sub

        <Fact>
        Sub IsPrime_InputIs1_ReturnFalse()
            Dim result As Boolean = _primeService.IsPrime(1)

            Assert.False(result, "1 should not be prime")
        End Sub

    End Class
End Namespace
```

The `[Fact]` attribute declares a test method that's run by the test runner. From the *PrimeService.Tests* folder, run `dotnet test`. The `dotnet test` command builds both projects and runs the tests. The xUnit test runner contains the program entry point to run the tests. `dotnet test` starts the test runner using the unit test project.

The test fails because `IsPrime` hasn't been implemented. Using the TDD approach, write only enough code so this test passes. Update `IsPrime` with the following code:

```
Public Function IsPrime(candidate As Integer) As Boolean
    If candidate = 1 Then
        Return False
    End If
    Throw New NotImplementedException("Not implemented.")
End Function
```

Run `dotnet test`. The test passes.

Add more tests

Add prime number tests for 0 and -1. You could copy the preceding test and change the following code to use 0 and -1:

```
Dim result As Boolean = _primeService.IsPrime(1)

Assert.False(result, "1 should not be prime")
```

Copying test code when only a parameter changes results in code duplication and test bloat. The following xUnit attributes enable writing a suite of similar tests:

- `[Theory]` represents a suite of tests that execute the same code but have different input arguments.
- `[InlineData]` attribute specifies values for those inputs.

Rather than creating new tests, apply the preceding xUnit attributes to create a single theory. Replace the following code:

```
<Fact>
Sub IsPrime_InputIs1_ReturnFalse()
    Dim result As Boolean = _primeService.IsPrime(1)

    Assert.False(result, "1 should not be prime")
End Sub
```

with the following code:

```
<Theory>
<InlineData(-1)>
<InlineData(0)>
<InlineData(1)>
Sub IsPrime_ValuesLessThan2_ReturnFalse(ByVal value As Integer)
    Dim result As Boolean = _primeService.IsPrime(value)

    Assert.False(result, $"{value} should not be prime")
End Sub
```

In the preceding code, `[Theory]` and `[InlineData]` enable testing several values less than two. Two is the smallest prime number.

Run `dotnet test`, two of the tests fail. To make all of the tests pass, update the `IsPrime` method with the following code:

```
Public Function IsPrime(candidate As Integer) As Boolean
    If candidate < 2 Then
        Return False
    End If
    Throw New NotImplementedException("Not fully implemented.")
End Function
```

Following the TDD approach, add more failing tests, then update the target code. See the [finished version of the tests](#) and the [complete implementation of the library](#).

The completed `IsPrime` method is not an efficient algorithm for testing primality.

Additional resources

- [xUnit.net official site](#)
- [Testing controller logic in ASP.NET Core](#)
- [`dotnet add reference`](#)

Organizing and testing projects with the .NET CLI

9/20/2022 • 6 minutes to read • [Edit Online](#)

This tutorial follows [Tutorial: Create a console application with .NET using Visual Studio Code](#), taking you beyond the creation of a simple console app to develop advanced and well-organized applications. After showing you how to use folders to organize your code, the tutorial shows you how to extend a console application with the [xUnit](#) testing framework.

NOTE

This tutorial recommends that you place the application project and test project in separate folders. Some developers prefer to keep these projects in the same folder. For more information, see GitHub issue [dotnet/docs #26395](#).

Using folders to organize code

If you want to introduce new types into a console app, you can do so by adding files containing the types to the app. For example, if you add files containing `AccountInformation` and `MonthlyReportRecords` types to your project, the project file structure is flat and easy to navigate:

```
/MyProject
|__AccountInformation.cs
|__MonthlyReportRecords.cs
|__MyProject.csproj
|__Program.cs
```

However, this flat structure only works well when the size of your project is relatively small. Can you imagine what will happen if you add 20 types to the project? The project definitely wouldn't be easy to navigate and maintain with that many files littering the project's root directory.

To organize the project, create a new folder and name it *Models* to hold the type files. Place the type files into the *Models* folder:

```
/MyProject
|__Models
    |__AccountInformation.cs
    |__MonthlyReportRecords.cs
|__MyProject.csproj
|__Program.cs
```

Projects that logically group files into folders are easy to navigate and maintain. In the next section, you create a more complex sample with folders and unit testing.

Organizing and testing using the NewTypes Pets Sample

Prerequisites

- [.NET 5.0 SDK](#) or a later version.

Building the sample

For the following steps, you can either follow along using the [NewTypes Pets Sample](#) or create your own files and folders. The types are logically organized into a folder structure that permits the addition of more types

later, and tests are also logically placed in folders permitting the addition of more tests later.

The sample contains two types, `Dog` and `Cat`, and has them implement a common interface, `IPet`. For the `NewTypes` project, your goal is to organize the pet-related types into a `Pets` folder. If another set of types is added later, `WildAnimals` for example, they're placed in the `NewTypes` folder alongside the `Pets` folder. The `WildAnimals` folder may contain types for animals that aren't pets, such as `Squirrel` and `Rabbit` types. In this way as types are added, the project remains well organized.

Create the following folder structure with file content indicated:

```
/NewTypes
|__src
  |__NewTypes
    |__Pets
      |__Dog.cs
      |__Cat.cs
      |__IPet.cs
    |__Program.cs
  |__NewTypes.csproj
```

IPet.cs:

```
using System;

namespace Pets
{
    public interface IPet
    {
        string TalkToOwner();
    }
}
```

Dog.cs:

```
using System;

namespace Pets
{
    public class Dog : IPet
    {
        public string TalkToOwner() => "Woof!";
    }
}
```

Cat.cs:

```
using System;

namespace Pets
{
    public class Cat : IPet
    {
        public string TalkToOwner() => "Meow!";
    }
}
```

Program.cs:

```

using System;
using Pets;
using System.Collections.Generic;

namespace ConsoleApplication
{
    public class Program
    {
        public static void Main(string[] args)
        {
            List<IPet> pets = new List<IPet>
            {
                new Dog(),
                new Cat()
            };

            foreach (var pet in pets)
            {
                Console.WriteLine(pet.TalkToOwner());
            }
        }
    }
}

```

NewTypes.csproj:

```

<Project Sdk="Microsoft.NET.Sdk">

<PropertyGroup>
<OutputType>Exe</OutputType>
<TargetFramework>net6.0</TargetFramework>
</PropertyGroup>

</Project>

```

Execute the following command:

```
dotnet run
```

Obtain the following output:

```

Woof!
Meow!

```

Optional exercise: You can add a new pet type, such as a `Bird`, by extending this project. Make the bird's `TalkToOwner` method give a `Tweet!` to the owner. Run the app again. The output will include `Tweet!`

Testing the sample

The `NewTypes` project is in place, and you've organized it by keeping the pets-related types in a folder. Next, create your test project and start writing tests with the `xUnit` test framework. Unit testing allows you to automatically check the behavior of your pet types to confirm that they're operating properly.

Navigate back to the `src` folder and create a `test` folder with a `NewTypesTests` folder within it. At a command prompt from the `NewTypesTests` folder, execute `dotnet new xunit`. This command produces two files: `NewTypesTests.csproj` and `UnitTest1.cs`.

The test project can't currently test the types in `NewTypes` and requires a project reference to the `NewTypes` project. To add a project reference, use the `dotnet add reference` command:

```
dotnet add reference ../../src/NewTypes/NewTypes.csproj
```

Or, you also have the option of manually adding the project reference by adding an `<ItemGroup>` node to the `NewTypesTests.csproj` file:

```
<ItemGroup>
  <ProjectReference Include="../../src/NewTypes/NewTypes.csproj" />
</ItemGroup>
```

`NewTypesTests.csproj`:

```
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <TargetFramework>net6.0</TargetFramework>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.NET.Test.Sdk" Version="17.3.1" />
    <PackageReference Include="xunit" Version="2.4.2" />
    <PackageReference Include="xunit.runner.visualstudio" Version="2.4.5" />
  </ItemGroup>

  <ItemGroup>
    <ProjectReference Include="../../src/NewTypes/NewTypes.csproj"/>
  </ItemGroup>

</Project>
```

The `NewTypesTests.csproj` file contains the following package references:

- `Microsoft.NET.Test.Sdk`, the .NET testing infrastructure
- `xunit`, the xUnit testing framework
- `xunit.runner.visualstudio`, the test runner
- `NewTypes`, the code to test

Change the name of `UnitTest1.cs` to `PetTests.cs` and replace the code in the file with the following code:

```

using System;
using Xunit;
using Pets;

public class PetTests
{
    [Fact]
    public void DogTalkToOwnerReturnsWoof()
    {
        string expected = "Woof!";
        string actual = new Dog().TalkToOwner();

        Assert.NotEqual(expected, actual);
    }

    [Fact]
    public void CatTalkToOwnerReturnsMeow()
    {
        string expected = "Meow!";
        string actual = new Cat().TalkToOwner();

        Assert.NotEqual(expected, actual);
    }
}

```

Optional exercise: If you added a `Bird` type earlier that yields a `Tweet!` to the owner, add a test method to the `PetTests.cs` file, `BirdTalkToOwnerReturnsTweet`, to check that the `TalkToOwner` method works correctly for the `Bird` type.

NOTE

Although you expect that the `expected` and `actual` values are equal, an initial assertion with the `Assert.NotEqual` check specifies that these values are *not equal*. Always initially create a test to fail in order to check the logic of the test. After you confirm that the test fails, adjust the assertion to allow the test to pass.

The following shows the complete project structure:

```

/NewTypes
|__src
|__NewTypes
|__Pets
|__Dog.cs
|__Cat.cs
|__IPet.cs
|__Program.cs
|__NewTypes.csproj
|__test
|__NewTypesTests
|__PetTests.cs
|__NewTypesTests.csproj

```

Start in the `test/NewTypesTests` directory. Run the tests with the `dotnet test` command. This command starts the test runner specified in the project file.

As expected, testing fails, and the console displays the following output:

```
Test run for C:\Source\dotnet\docs\samples\snippets\core\tutorials\testing-with-
cli\csharp\test\NewTypesTests\bin\Debug\net5.0\NewTypesTests.dll (.NETCoreApp,Version=v5.0)
Microsoft (R) Test Execution Command Line Tool Version 16.8.1
Copyright (c) Microsoft Corporation. All rights reserved.

Starting test execution, please wait...
A total of 1 test files matched the specified pattern.
[xUnit.net 00:00:00.50]      PetTests.DogTalkToOwnerReturnsWoof [FAIL]
  Failed PetTests.DogTalkToOwnerReturnsWoof [6 ms]
    Error Message:
      Assert.NotEqual() Failure
    Expected: Not "Woof!"
    Actual:   "Woof!"
    Stack Trace:
      at PetTests.DogTalkToOwnerReturnsWoof() in
C:\Source\dotnet\docs\samples\snippets\core\tutorials\testing-with-
cli\csharp\test\NewTypesTests\PetTests.cs:line 13

Failed! - Failed: 1, Passed: 1, Skipped: 0, Total: 2, Duration: 8 ms - NewTypesTests.dll
(net5.0)
```

Change the assertions of your tests from `Assert.NotEqual` to `Assert.Equal`:

```
using System;
using Xunit;
using Pets;

public class PetTests
{
    [Fact]
    public void DogTalkToOwnerReturnsWoof()
    {
        string expected = "Woof!";
        string actual = new Dog().TalkToOwner();

        Assert.Equal(expected, actual);
    }

    [Fact]
    public void CatTalkToOwnerReturnsMeow()
    {
        string expected = "Meow!";
        string actual = new Cat().TalkToOwner();

        Assert.Equal(expected, actual);
    }
}
```

Rerun the tests with the `dotnet test` command and obtain the following output:

```
Test run for C:\Source\dotnet\docs\samples\snippets\core\tutorials\testing-with-
cli\csharp\test\NewTypesTests\bin\Debug\net5.0\NewTypesTests.dll (.NETCoreApp,Version=v5.0)
Microsoft (R) Test Execution Command Line Tool Version 16.8.1
Copyright (c) Microsoft Corporation. All rights reserved.

Starting test execution, please wait...
A total of 1 test files matched the specified pattern.

Passed! - Failed: 0, Passed: 2, Skipped: 0, Total: 2, Duration: 2 ms - NewTypesTests.dll
(net5.0)
```

Testing passes. The pet types' methods return the correct values when talking to the owner.

You've learned techniques for organizing and testing projects using xUnit. Go forward with these techniques applying them in your own projects. *Happy coding!*

Unit testing C# with NUnit and .NET Core

9/20/2022 • 4 minutes to read • [Edit Online](#)

This tutorial takes you through an interactive experience building a sample solution step-by-step to learn unit testing concepts. If you prefer to follow the tutorial using a pre-built solution, [view or download the sample code](#) before you begin. For download instructions, see [Samples and Tutorials](#).

This article is about testing a .NET Core project. If you're testing an ASP.NET Core project, see [Integration tests in ASP.NET Core](#).

Prerequisites

- [.NET Core 2.1 SDK](#) or later versions.
- A text editor or code editor of your choice.

Creating the source project

Open a shell window. Create a directory called *unit-testing-using-nunit* to hold the solution. Inside this new directory, run the following command to create a new solution file for the class library and the test project:

```
dotnet new sln
```

Next, create a *PrimeService* directory. The following outline shows the directory and file structure so far:

```
/unit-testing-using-nunit
  unit-testing-using-nunit.sln
  /PrimeService
```

Make *PrimeService* the current directory and run the following command to create the source project:

```
dotnet new classlib
```

Rename *Class1.cs* to *PrimeService.cs*. You create a failing implementation of the `PrimeService` class:

```
using System;

namespace Prime.Services
{
    public class PrimeService
    {
        public bool IsPrime(int candidate)
        {
            throw new NotImplementedException("Please create a test first.");
        }
    }
}
```

Change the directory back to the *unit-testing-using-nunit* directory. Run the following command to add the class library project to the solution:

```
dotnet sln add PrimeService/PrimeService.csproj
```

Creating the test project

Next, create the *PrimeService.Tests* directory. The following outline shows the directory structure:

```
/unit-testing-using-nunit
  unit-testing-using-nunit.sln
  /PrimeService
    Source Files
    PrimeService.csproj
  /PrimeService.Tests
```

Make the *PrimeService.Tests* directory the current directory and create a new project using the following command:

```
dotnet new nunit
```

The [dotnet new](#) command creates a test project that uses NUnit as the test library. The generated template configures the test runner in the *PrimeService.Tests.csproj* file:

```
<ItemGroup>
  <PackageReference Include="nunit" Version="3.13.3" />
  <PackageReference Include="NUnit3TestAdapter" Version="4.2.1" />
  <PackageReference Include="Microsoft.NET.Test.Sdk" Version="17.3.1" />
</ItemGroup>
```

The test project requires other packages to create and run unit tests. The [dotnet new](#) command in the previous step added the Microsoft test SDK, the NUnit test framework, and the NUnit test adapter. Now, add the *PrimeService* class library as another dependency to the project. Use the [dotnet add reference](#) command:

```
dotnet add reference ../PrimeService/PrimeService.csproj
```

You can see the entire file in the [samples repository](#) on GitHub.

The following outline shows the final solution layout:

```
/unit-testing-using-nunit
  unit-testing-using-nunit.sln
  /PrimeService
    Source Files
    PrimeService.csproj
  /PrimeService.Tests
    Test Source Files
    PrimeService.Tests.csproj
```

Execute the following command in the *unit-testing-using-nunit* directory:

```
dotnet sln add ./PrimeService.Tests/PrimeService.Tests.csproj
```

Creating the first test

You write one failing test, make it pass, and then repeat the process. In the `PrimeService.Tests` directory, rename the `UnitTest1.cs` file to `PrimeService_IsPrimeShould.cs` and replace its entire contents with the following code:

```
using NUnit.Framework;
using Prime.Services;

namespace Prime.UnitTests.Services
{
    [TestFixture]
    public class PrimeService_IsPrimeShould
    {
        private PrimeService _primeService;

        [SetUp]
        public void SetUp()
        {
            _primeService = new PrimeService();
        }

        [Test]
        public void IsPrime_InputIs1_ReturnFalse()
        {
            var result = _primeService.IsPrime(1);

            Assert.IsFalse(result, "1 should not be prime");
        }
    }
}
```

The `[TestFixture]` attribute denotes a class that contains unit tests. The `[Test]` attribute indicates a method is a test method.

Save this file and execute the `dotnet test` command to build the tests and the class library and run the tests. The NUnit test runner contains the program entry point to run your tests. `dotnet test` starts the test runner using the unit test project you've created.

Your test fails. You haven't created the implementation yet. Make the test pass by writing the simplest code in the `PrimeService` class that works:

```
public bool IsPrime(int candidate)
{
    if (candidate == 1)
    {
        return false;
    }
    throw new NotImplementedException("Please create a test first.");
}
```

In the `unit-testing-using-nunit` directory, run `dotnet test` again. The `dotnet test` command runs a build for the `PrimeService` project and then for the `PrimeService.Tests` project. After you build both projects, it runs this single test. It passes.

Adding more features

Now that you've made one test pass, it's time to write more. There are a few other simple cases for prime numbers: 0, -1. You could add new tests with the `[Test]` attribute, but that quickly becomes tedious. There are other NUnit attributes that enable you to write a suite of similar tests. A `[TestCase]` attribute is used to create a suite of tests that execute the same code but have different input arguments. You can use the `[TestCase]` attribute to specify values for those inputs.

Instead of creating new tests, apply this attribute to create a single data-driven test. The data driven test is a method that tests several values less than two, which is the lowest prime number:

```
[TestCase(-1)]
[TestCase(0)]
[TestCase(1)]
public void IsPrime_ValuesLessThan2_ReturnFalse(int value)
{
    var result = _primeService.IsPrime(value);

    Assert.IsFalse(result, $"{value} should not be prime");
}
```

Run `dotnet test`, and two of these tests fail. To make all of the tests pass, change the `if` clause at the beginning of the `Main` method in the `PrimeService.cs` file:

```
if (candidate < 2)
```

Continue to iterate by adding more tests, theories, and code in the main library. You have the [finished version of the tests](#) and the [complete implementation of the library](#).

You've built a small library and a set of unit tests for that library. You've also structured the solution so that adding new packages and tests is part of the standard workflow. You've concentrated most of your time and effort on solving the goals of the application.

Unit testing F# libraries in .NET Core using dotnet test and NUnit

9/20/2022 • 5 minutes to read • [Edit Online](#)

This tutorial takes you through an interactive experience building a sample solution step-by-step to learn unit testing concepts. If you prefer to follow the tutorial using a pre-built solution, [view or download the sample code](#) before you begin. For download instructions, see [Samples and Tutorials](#).

This article is about testing a .NET Core project. If you're testing an ASP.NET Core project, see [Integration tests in ASP.NET Core](#).

Prerequisites

- [.NET Core 2.1 SDK](#) or later versions.
- A text editor or code editor of your choice.

Creating the source project

Open a shell window. Create a directory called *unit-testing-with-fsharp* to hold the solution. Inside this new directory, run the following command to create a new solution file for the class library and the test project:

```
dotnet new sln
```

Next, create a *MathService* directory. The following outline shows the directory and file structure so far:

```
/unit-testing-with-fsharp
    unit-testing-with-fsharp.sln
    /MathService
```

Make *MathService* the current directory and run the following command to create the source project:

```
dotnet new classlib -lang "F#"
```

You create a failing implementation of the math service:

```
module MyMath =
    let squaresOfOdds xs = raise (System.NotImplementedException("You haven't written a test yet!"))
```

Change the directory back to the *unit-testing-with-fsharp* directory. Run the following command to add the class library project to the solution:

```
dotnet sln add .\MathService\MathService.fsproj
```

Creating the test project

Next, create the *MathService.Tests* directory. The following outline shows the directory structure:

```
/unit-testing-with-fsharp
  unit-testing-with-fsharp.sln
  /MathService
    Source Files
      MathService.fsproj
  /MathService.Tests
```

Make the *MathService.Tests* directory the current directory and create a new project using the following command:

```
dotnet new nunit -lang "F#"
```

This creates a test project that uses NUnit as the test framework. The generated template configures the test runner in the *MathServiceTests.fsproj*:

```
<ItemGroup>
  <PackageReference Include="Microsoft.NET.Test.Sdk" Version="15.5.0" />
  <PackageReference Include="NUnit" Version="3.9.0" />
  <PackageReference Include="NUnit3TestAdapter" Version="3.9.0" />
</ItemGroup>
```

The test project requires other packages to create and run unit tests. `dotnet new` in the previous step added NUnit and the NUnit test adapter. Now, add the `MathService` class library as another dependency to the project. Use the `dotnet add reference` command:

```
dotnet add reference ../MathService/MathService.fsproj
```

You can see the entire file in the [samples repository](#) on GitHub.

You have the following final solution layout:

```
/unit-testing-with-fsharp
  unit-testing-with-fsharp.sln
  /MathService
    Source Files
      MathService.fsproj
  /MathService.Tests
    Test Source Files
      MathService.Tests.fsproj
```

Execute the following command in the *unit-testing-with-fsharp* directory:

```
dotnet sln add .\MathService.Tests\MathService.Tests.fsproj
```

Creating the first test

You write one failing test, make it pass, then repeat the process. Open *UnitTest1.fs* and add the following code:

```

namespace MathService.Tests

open System
open NUnit.Framework
open MathService

[<TestFixture>]
type TestClass () =
    []
    member this.TestMethodPassing() =
        Assert.True(true)

    []
    member this.FailEveryTime() = Assert.True(false)

```

The `[<TestFixture>]` attribute denotes a class that contains tests. The `[<Test>]` attribute denotes a test method that is run by the test runner. From the *unit-testing-with-fsharp* directory, execute `dotnet test` to build the tests and the class library and then run the tests. The NUnit test runner contains the program entry point to run your tests. `dotnet test` starts the test runner using the unit test project you've created.

These two tests show the most basic passing and failing tests. `My test` passes, and `Fail every time` fails. Now, create a test for the `squaresOfOdds` method. The `squaresOfOdds` method returns a sequence of the squares of all odd integer values that are part of the input sequence. Rather than trying to write all of those functions at once, you can iteratively create tests that validate the functionality. Making each test pass means creating the necessary functionality for the method.

The simplest test we can write is to call `squaresOfOdds` with all even numbers, where the result should be an empty sequence of integers. Here's that test:

```

[<Test>]
member this.TestEvenSequence() =
    let expected = Seq.empty<int>
    let actual = MyMath.squaresOfOdds [2; 4; 6; 8; 10]
    Assert.That(actual, Is.EqualTo(expected))

```

Notice that the `expected` sequence has been converted to a list. The NUnit framework relies on many standard .NET types. That dependency means that your public interface and expected results support [ICollection](#) rather than [IEnumerable](#).

When you run the test, you see that your test fails. You haven't created the implementation yet. Make this test pass by writing the simplest code in the *Library.fs* class in your MathService project that works:

```

let squaresOfOdds xs =
    Seq.empty<int>

```

In the *unit-testing-with-fsharp* directory, run `dotnet test` again. The `dotnet test` command runs a build for the `MathService` project and then for the `MathService.Tests` project. After building both projects, it runs your tests. Two tests pass now.

Completing the requirements

Now that you've made one test pass, it's time to write more. The next simple case works with a sequence whose only odd number is `1`. The number 1 is easier because the square of 1 is 1. Here's that next test:

```
[<Test>]
member public this.TestOnesAndEvens() =
    let expected = [1; 1; 1; 1]
    let actual = MyMath.squaresOfOdds [2; 1; 4; 1; 6; 1; 8; 1; 10]
    Assert.That(actual, Is.EqualTo(expected))
```

Executing `dotnet test` fails the new test. You must update the `squaresOfOdds` method to handle this new test. You must filter all the even numbers out of the sequence to make this test pass. You can do that by writing a small filter function and using `Seq.filter`:

```
let private isOdd x = x % 2 <> 0

let squaresOfOdds xs =
    xs
    |> Seq.filter isOdd
```

There's one more step to go: square each of the odd numbers. Start by writing a new test:

```
[<Test>]
member public this.TestSquaresOfOdds() =
    let expected = [1; 9; 25; 49; 81]
    let actual = MyMath.squaresOfOdds [1; 2; 3; 4; 5; 6; 7; 8; 9; 10]
    Assert.That(actual, Is.EqualTo(expected))
```

You can fix the test by piping the filtered sequence through a map operation to compute the square of each odd number:

```
let private square x = x * x
let private isOdd x = x % 2 <> 0

let squaresOfOdds xs =
    xs
    |> Seq.filter isOdd
    |> Seq.map square
```

You've built a small library and a set of unit tests for that library. You've structured the solution so that adding new packages and tests is part of the normal workflow. You've concentrated most of your time and effort on solving the goals of the application.

See also

- [dotnet add reference](#)
- [dotnet test](#)

Unit testing Visual Basic .NET Core libraries using dotnet test and NUnit

9/20/2022 • 4 minutes to read • [Edit Online](#)

This tutorial takes you through an interactive experience building a sample solution step-by-step to learn unit testing concepts. If you prefer to follow the tutorial using a pre-built solution, [view or download the sample code](#) before you begin. For download instructions, see [Samples and Tutorials](#).

This article is about testing a .NET Core project. If you're testing an ASP.NET Core project, see [Integration tests in ASP.NET Core](#).

Prerequisites

- [.NET Core 2.1 SDK](#) or later versions.
- A text editor or code editor of your choice.

Creating the source project

Open a shell window. Create a directory called *unit-testing-vb-nunit* to hold the solution. Inside this new directory, run the following command to create a new solution file for the class library and the test project:

```
dotnet new sln
```

Next, create a *PrimeService* directory. The following outline shows the file structure so far:

```
/unit-testing-vb-nunit
    unit-testing-vb-nunit.sln
    /PrimeService
```

Make *PrimeService* the current directory and run the following command to create the source project:

```
dotnet new classlib -lang VB
```

Rename *Class1.VB* to *PrimeService.VB*. You create a failing implementation of the `PrimeService` class:

```
Namespace Prime.Services
    Public Class PrimeService
        Public Function IsPrime(candidate As Integer) As Boolean
            Throw New NotImplementedException("Please create a test first.")
        End Function
    End Class
End Namespace
```

Change the directory back to the *unit-testing-vb-using-mstest* directory. Run the following command to add the class library project to the solution:

```
dotnet sln add .\PrimeService\PrimeService.vbproj
```

Creating the test project

Next, create the `PrimeService.Tests` directory. The following outline shows the directory structure:

```
/unit-testing-vb-nunit
    unit-testing-vb-nunit.sln
    /PrimeService
        Source Files
        PrimeService.vbproj
    /PrimeService.Tests
```

Make the `PrimeService.Tests` directory the current directory and create a new project using the following command:

```
dotnet new nunit -lang VB
```

The `dotnet new` command creates a test project that uses NUnit as the test library. The generated template configures the test runner in the `PrimeServiceTests.vbproj` file:

```
<ItemGroup>
    <PackageReference Include="nunit" Version="3.13.3" />
    <PackageReference Include="NUnit3TestAdapter" Version="4.2.1" />
    <PackageReference Include="Microsoft.NET.Test.Sdk" Version="17.3.1" />
</ItemGroup>
```

The test project requires other packages to create and run unit tests. `dotnet new` in the previous step added NUnit and the NUnit test adapter. Now, add the `PrimeService` class library as another dependency to the project. Use the `dotnet add reference` command:

```
dotnet add reference ..\PrimeService\PrimeService.vbproj
```

You can see the entire file in the [samples repository](#) on GitHub.

You have the following final solution layout:

```
/unit-testing-vb-nunit
    unit-testing-vb-nunit.sln
    /PrimeService
        Source Files
        PrimeService.vbproj
    /PrimeService.Tests
        Test Source Files
        PrimeService.Tests.vbproj
```

Execute the following command in the `unit-testing-vb-nunit` directory:

```
dotnet sln add .\PrimeService.Tests\PrimeService.Tests.vbproj
```

Creating the first test

You write one failing test, make it pass, then repeat the process. In the `PrimeService.Tests` directory, rename the `UnitTest1.vb` file to `PrimeService_IsPrimeShould.VB` and replace its entire contents with the following code:

```

Imports NUnit.Framework

Namespace PrimeService.Tests
    <TestFixture>
    Public Class PrimeService_IsPrimeShould
        Private _primeService As Prime.Services.PrimeService = New Prime.Services.PrimeService()

        <Test>
        Sub IsPrime_InputIs1_ReturnFalse()
            Dim result As Boolean = _primeService.IsPrime(1)

            Assert.False(result, "1 should not be prime")
        End Sub

    End Class
End Namespace

```

The `<TestFixture>` attribute indicates a class that contains tests. The `<Test>` attribute denotes a method that is run by the test runner. From the *unit-testing-vb-nunit*, execute `dotnet test` to build the tests and the class library and then run the tests. The NUnit test runner contains the program entry point to run your tests. `dotnet test` starts the test runner using the unit test project you've created.

Your test fails. You haven't created the implementation yet. Make this test pass by writing the simplest code in the `PrimeService` class that works:

```

Public Function IsPrime(candidate As Integer) As Boolean
    If candidate = 1 Then
        Return False
    End If
    Throw New NotImplementedException("Please create a test first.")
End Function

```

In the *unit-testing-vb-nunit* directory, run `dotnet test` again. The `dotnet test` command runs a build for the `PrimeService` project and then for the `PrimeService.Tests` project. After building both projects, it runs this single test. It passes.

Adding more features

Now that you've made one test pass, it's time to write more. There are a few other simple cases for prime numbers: 0, -1. You could add those cases as new tests with the `<Test>` attribute, but that quickly becomes tedious. There are other xUnit attributes that enable you to write a suite of similar tests. A `<TestCase>` attribute represents a suite of tests that execute the same code but have different input arguments. You can use the `<TestCase>` attribute to specify values for those inputs.

Instead of creating new tests, apply these two attributes to create a series of tests that test several values less than two, which is the lowest prime number:

```

<TestFixture>
Public Class PrimeService_IsPrimeShould
    Private _primeService As Prime.Services.PrimeService = New Prime.Services.PrimeService()

    <TestCase(-1)>
    <TestCase(0)>
    <TestCase(1)>
    Sub IsPrime_ValuesLessThan2_ReturnFalse(value As Integer)
        Dim result As Boolean = _primeService.IsPrime(value)

        Assert.IsFalse(result, $"{value} should not be prime")
    End Sub

    <TestCase(2)>
    <TestCase(3)>
    <TestCase(5)>
    <TestCase(7)>
    Public Sub IsPrime_PrimesLessThan10_ReturnTrue(value As Integer)
        Dim result As Boolean = _primeService.IsPrime(value)

        Assert.IsTrue(result, $"{value} should be prime")
    End Sub

    <TestCase(4)>
    <TestCase(6)>
    <TestCase(8)>
    <TestCase(9)>
    Public Sub IsPrime_NonPrimesLessThan10_ReturnFalse(value As Integer)
        Dim result As Boolean = _primeService.IsPrime(value)

        Assert.IsFalse(result, $"{value} should not be prime")
    End Sub
End Class

```

Run `dotnet test`, and two of these tests fail. To make all of the tests pass, change the `if` clause at the beginning of the `Main` method in the `PrimeServices.cs` file:

```
if candidate < 2
```

Continue to iterate by adding more tests, more theories, and more code in the main library. You have the [finished version of the tests](#) and the [complete implementation of the library](#).

You've built a small library and a set of unit tests for that library. You've structured the solution so that adding new packages and tests is part of the normal workflow. You've concentrated most of your time and effort on solving the goals of the application.

Unit testing C# with MSTest and .NET

9/20/2022 • 4 minutes to read • [Edit Online](#)

This tutorial takes you through an interactive experience building a sample solution step-by-step to learn unit testing concepts. If you prefer to follow the tutorial using a pre-built solution, [view or download the sample code](#) before you begin. For download instructions, see [Samples and Tutorials](#).

This article is about testing a .NET Core project. If you're testing an ASP.NET Core project, see [Integration tests in ASP.NET Core](#).

Prerequisites

- The [.NET 6.0 SDK or later](#)

Create the source project

Open a shell window. Create a directory called *unit-testing-using-mstest* to hold the solution. Inside this new directory, run `dotnet new sln` to create a new solution file for the class library and the test project. Create a *PrimeService* directory. The following outline shows the directory and file structure thus far:

```
/unit-testing-using-mstest
    unit-testing-using-mstest.sln
    /PrimeService
```

Make *PrimeService* the current directory and run `dotnet new classlib` to create the source project. Rename *Class1.cs* to *PrimeService.cs*. Replace the code in the file with the following code to create a failing implementation of the `PrimeService` class:

```
using System;

namespace Prime.Services
{
    public class PrimeService
    {
        public bool IsPrime(int candidate)
        {
            throw new NotImplementedException("Please create a test first.");
        }
    }
}
```

Change the directory back to the *unit-testing-using-mstest* directory. Run `dotnet sln add` to add the class library project to the solution:

```
dotnet sln add PrimeService/PrimeService.csproj
```

Create the test project

Create the *PrimeService.Tests* directory. The following outline shows the directory structure:

```
/unit-testing-using-mstest
  unit-testing-using-mstest.sln
  /PrimeService
    Source Files
      PrimeService.csproj
  /PrimeService.Tests
```

Make the *PrimeService.Tests* directory the current directory and create a new project using `dotnet new mstest`.

The `dotnet new` command creates a test project that uses MSTest as the test library. The template configures the test runner in the *PrimeServiceTests.csproj* file:

```
<ItemGroup>
  <PackageReference Include="Microsoft.NET.Test.Sdk" Version="16.7.1" />
  <PackageReference Include="MSTest.TestAdapter" Version="2.1.1" />
  <PackageReference Include="MSTest.TestFramework" Version="2.1.1" />
  <PackageReference Include="coverlet.collector" Version="1.3.0" />
</ItemGroup>
```

The test project requires other packages to create and run unit tests. `dotnet new` in the previous step added the MSTest SDK, the MSTest test framework, the MSTest runner, and coverlet for code coverage reporting.

Add the `PrimeService` class library as another dependency to the project. Use the `dotnet add reference` command:

```
dotnet add reference ../PrimeService/PrimeService.csproj
```

You can see the entire file in the [samples repository](#) on GitHub.

The following outline shows the final solution layout:

```
/unit-testing-using-mstest
  unit-testing-using-mstest.sln
  /PrimeService
    Source Files
      PrimeService.csproj
  /PrimeService.Tests
    Test Source Files
      PrimeServiceTests.csproj
```

Change to the *unit-testing-using-mstest* directory, and run `dotnet sln add`:

```
dotnet sln add .\PrimeService.Tests\PrimeService.Tests.csproj
```

Create the first test

Write a failing test, make it pass, then repeat the process. Remove *UnitTest1.cs* from the *PrimeService.Tests* directory and create a new C# file named *PrimeService_IsPrimeShould.cs* with the following content:

```

using Microsoft.VisualStudio.TestTools.UnitTesting;
using Prime.Services;

namespace Prime.UnitTests.Services
{
    [TestClass]
    public class PrimeService_IsPrimeShould
    {
        private readonly PrimeService _primeService;

        public PrimeService_IsPrimeShould()
        {
            _primeService = new PrimeService();
        }

        [TestMethod]
        public void IsPrime_InputIs1_ReturnFalse()
        {
            bool result = _primeService.IsPrime(1);

            Assert.IsFalse(result, "1 should not be prime");
        }
    }
}

```

The [TestClass attribute](#) denotes a class that contains unit tests. The [TestMethod attribute](#) indicates a method is a test method.

Save this file and execute `dotnet test` to build the tests and the class library and then run the tests. The MSTest test runner contains the program entry point to run your tests. `dotnet test` starts the test runner using the unit test project you've created.

Your test fails. You haven't created the implementation yet. Make this test pass by writing the simplest code in the `PrimeService` class that works:

```

public bool IsPrime(int candidate)
{
    if (candidate == 1)
    {
        return false;
    }
    throw new NotImplementedException("Please create a test first.");
}

```

In the `unit-testing-using-mstest` directory, run `dotnet test` again. The `dotnet test` command runs a build for the `PrimeService` project and then for the `PrimeService.Tests` project. After building both projects, it runs this single test. It passes.

Add more features

Now that you've made one test pass, it's time to write more. There are a few other simple cases for prime numbers: 0, -1. You could add new tests with the [TestMethod attribute](#), but that quickly becomes tedious. There are other MSTest attributes that enable you to write a suite of similar tests. A test method can execute the same code but have different input arguments. You can use the [DataRow attribute](#) to specify values for those inputs.

Instead of creating new tests, apply these two attributes to create a single data driven test. The data driven test is a method that tests several values less than two, which is the lowest prime number. Add a new test method in `PrimeService_IsPrimeShould.cs`:

```
[TestMethod]
[DataRow(-1)]
[DataRow(0)]
[DataRow(1)]
public void IsPrime_ValuesLessThan2_ReturnFalse(int value)
{
    var result = _primeService.IsPrime(value);

    Assert.IsFalse(result, $"{value} should not be prime");
}
```

Run `dotnet test`, and two of these tests fail. To make all of the tests pass, change the `if` clause at the beginning of the `IsPrime` method in the *PrimeService.cs* file:

```
if (candidate < 2)
```

Continue to iterate by adding more tests, more theories, and more code in the main library. You have the [finished version of the tests](#) and the [complete implementation of the library](#).

You've built a small library and a set of unit tests for that library. You've structured the solution so that adding new packages and tests is part of the normal workflow. You've concentrated most of your time and effort on solving the goals of the application.

See also

- [Microsoft.VisualStudio.TestTools.UnitTesting](#)
- [Use the MSTest framework in unit tests](#)
- [MSTest V2 test framework docs](#)

Unit testing F# libraries in .NET Core using dotnet test and MSTest

9/20/2022 • 5 minutes to read • [Edit Online](#)

This tutorial takes you through an interactive experience building a sample solution step-by-step to learn unit testing concepts. If you prefer to follow the tutorial using a pre-built solution, [view or download the sample code](#) before you begin. For download instructions, see [Samples and Tutorials](#).

This article is about testing a .NET Core project. If you're testing an ASP.NET Core project, see [Integration tests in ASP.NET Core](#).

Creating the source project

Open a shell window. Create a directory called *unit-testing-with-fsharp* to hold the solution. Inside this new directory, run `dotnet new sln` to create a new solution. This makes it easier to manage both the class library and the unit test project. Inside the solution directory, create a *MathService* directory. The directory and file structure thus far is shown below:

```
/unit-testing-with-fsharp
  unit-testing-with-fsharp.sln
  /MathService
```

Make *MathService* the current directory and run `dotnet new classlib -lang "F#"` to create the source project. You'll create a failing implementation of the math service:

```
module MyMath =
    let squaresOfOdds xs = raise (System.NotImplementedException("You haven't written a test yet!"))
```

Change the directory back to the *unit-testing-with-fsharp* directory. Run `dotnet sln add .\MathService\MathService.fsproj` to add the class library project to the solution.

Creating the test project

Next, create the *MathService.Tests* directory. The following outline shows the directory structure:

```
/unit-testing-with-fsharp
  unit-testing-with-fsharp.sln
  /MathService
    Source Files
    MathService.fsproj
  /MathService.Tests
```

Make the *MathService.Tests* directory the current directory and create a new project using `dotnet new mstest -lang "F#"`. This creates a test project that uses MSTest as the test framework. The generated template configures the test runner in the *MathServiceTests.fsproj*.

```
<ItemGroup>
  <PackageReference Include="Microsoft.NET.Test.Sdk" Version="15.3.0-preview-20170628-02" />
  <PackageReference Include="MSTest.TestAdapter" Version="1.1.18" />
  <PackageReference Include="MSTest.TestFramework" Version="1.1.18" />
</ItemGroup>
```

The test project requires other packages to create and run unit tests. `dotnet new` in the previous step added MSTest and the MSTest runner. Now, add the `MathService` class library as another dependency to the project. Use the `dotnet add reference` command:

```
dotnet add reference ../MathService/MathService.fsproj
```

You can see the entire file in the [samples repository](#) on GitHub.

You have the following final solution layout:

```
/unit-testing-with-fsharp
  unit-testing-with-fsharp.sln
  /MathService
    Source Files
    MathService.fsproj
  /MathService.Tests
    Test Source Files
    MathServiceTests.fsproj
```

Execute `dotnet sln add .\MathService.Tests\MathService.Tests.fsproj` in the *unit-testing-with-fsharp* directory.

Creating the first test

You write one failing test, make it pass, then repeat the process. Open *Tests.fs* and add the following code:

```
namespace MathService.Tests

open System
open Microsoft.VisualStudio.TestTools.UnitTesting
open MathService

[<TestClass>]
type TestClass () =

  [<TestMethod>]
  member this.TestMethodPassing() =
    Assert.IsTrue(true)

  [<TestMethod>]
  member this.FailEveryTime() = Assert.IsTrue(false)
```

The `[<TestClass>]` attribute denotes a class that contains tests. The `[<TestMethod>]` attribute denotes a test method that is run by the test runner. From the *unit-testing-with-fsharp* directory, execute `dotnet test` to build the tests and the class library and then run the tests. The MSTest test runner contains the program entry point to run your tests. `dotnet test` starts the test runner using the unit test project you've created.

These two tests show the most basic passing and failing tests. `My test` passes, and `Fail every time` fails. Now, create a test for the `squaresOfOdds` method. The `squaresOfOdds` method returns a list of the squares of all odd integer values that are part of the input sequence. Rather than trying to write all of those functions at once, you can iteratively create tests that validate the functionality. Making each test pass means creating the necessary functionality for the method.

The simplest test we can write is to call `squaresOfOdds` with all even numbers, where the result should be an empty sequence of integers. Here's that test:

```
[<TestMethod>]
member this.TestEvenSequence() =
    let expected = Seq.empty<int> |> Seq.toList
    let actual = MyMath.squaresOfOdds [2; 4; 6; 8; 10]
    Assert.AreEqual(expected, actual)
```

Notice that the `expected` sequence has been converted to a list. The MSTest library relies on many standard .NET types. That dependency means that your public interface and expected results support [ICollection](#) rather than [IEnumerable](#).

When you run the test, you see that your test fails. You haven't created the implementation yet. Make this test pass by writing the simplest code in the `MathService` class that works:

```
let squaresOfOdds xs =
    Seq.empty<int> |> Seq.toList
```

In the `unit-testing-with-fsharp` directory, run `dotnet test` again. The `dotnet test` command runs a build for the `MathService` project and then for the `MathService.Tests` project. After building both projects, it runs this single test. It passes.

Completing the requirements

Now that you've made one test pass, it's time to write more. The next simple case works with a sequence whose only odd number is `1`. The number 1 is easier because the square of 1 is 1. Here's that next test:

```
[<TestMethod>]
member public this.TestOnesAndEvens() =
    let expected = [1; 1; 1; 1]
    let actual = MyMath.squaresOfOdds [2; 1; 4; 1; 6; 1; 8; 1; 10]
    Assert.AreEqual(expected, actual)
```

Executing `dotnet test` fails the new test. You must update the `squaresOfOdds` method to handle this new test. You must filter all the even numbers out of the sequence to make this test pass. You can do that by writing a small filter function and using `Seq.filter`:

```
let private isOdd x = x % 2 <> 0

let squaresOfOdds xs =
    xs
    |> Seq.filter isOdd |> Seq.toList
```

Notice the call to `Seq.toList`. That creates a list, which implements the [ICollection](#) interface.

There's one more step to go: square each of the odd numbers. Start by writing a new test:

```
[<TestMethod>]
member public this.TestSquaresOfOdds() =
    let expected = [1; 9; 25; 49; 81]
    let actual = MyMath.squaresOfOdds [1; 2; 3; 4; 5; 6; 7; 8; 9; 10]
    Assert.AreEqual(expected, actual)
```

You can fix the test by piping the filtered sequence through a map operation to compute the square of each odd

number:

```
let private square x = x * x
let private isOdd x = x % 2 <> 0

let squaresOfOdds xs =
    xs
    |> Seq.filter isOdd
    |> Seq.map square
    |> Seq.toList
```

You've built a small library and a set of unit tests for that library. You've structured the solution so that adding new packages and tests is part of the normal workflow. You've concentrated most of your time and effort on solving the goals of the application.

See also

- [dotnet new](#)
- [dotnet sln](#)
- [dotnet add reference](#)
- [dotnet test](#)

Unit testing Visual Basic .NET Core libraries using dotnet test and MSTest

9/20/2022 • 4 minutes to read • [Edit Online](#)

This tutorial takes you through an interactive experience building a sample solution step-by-step to learn unit testing concepts. If you prefer to follow the tutorial using a pre-built solution, [view or download the sample code](#) before you begin. For download instructions, see [Samples and Tutorials](#).

This article is about testing a .NET Core project. If you're testing an ASP.NET Core project, see [Integration tests in ASP.NET Core](#).

Creating the source project

Open a shell window. Create a directory called *unit-testing-vb-mstest* to hold the solution. Inside this new directory, run `dotnet new sln` to create a new solution. This practice makes it easier to manage both the class library and the unit test project. Inside the solution directory, create a *PrimeService* directory. You have the following directory and file structure thus far:

```
/unit-testing-vb-mstest
  unit-testing-vb-mstest.sln
  /PrimeService
```

Make *PrimeService* the current directory and run `dotnet new classlib -lang VB` to create the source project. Rename *Class1.VB* to *PrimeService.VB*. You create a failing implementation of the `PrimeService` class:

```
Namespace Prime.Services
    Public Class PrimeService
        Public Function IsPrime(candidate As Integer) As Boolean
            Throw New NotImplementedException("Please create a test first")
        End Function
    End Class
End Namespace
```

Change the directory back to the *unit-testing-vb-using-mstest* directory. Run `dotnet sln add .\PrimeService\PrimeService.vbproj` to add the class library project to the solution.

Creating the test project

Next, create the *PrimeService.Tests* directory. The following outline shows the directory structure:

```
/unit-testing-vb-mstest
  unit-testing-vb-mstest.sln
  /PrimeService
    Source Files
    PrimeService.vbproj
  /PrimeService.Tests
```

Make the *PrimeService.Tests* directory the current directory and create a new project using `dotnet new mstest -lang VB`. This command creates a test project that uses MSTest as the test library. The generated template configures the test runner in the *PrimeServiceTests.vbproj*.

```
<ItemGroup>
  <PackageReference Include="Microsoft.NET.Test.Sdk" Version="15.5.0" />
  <PackageReference Include="MSTest.TestAdapter" Version="1.1.18" />
  <PackageReference Include="MSTest.TestFramework" Version="1.1.18" />
</ItemGroup>
```

The test project requires other packages to create and run unit tests. `dotnet new` in the previous step added MSTest and the MSTest runner. Now, add the `PrimeService` class library as another dependency to the project. Use the `dotnet add reference` command:

```
dotnet add reference ../PrimeService/PrimeService.vbproj
```

You can see the entire file in the [samples repository](#) on GitHub.

You have the following final solution layout:

```
/unit-testing-vb-mstest
  unit-testing-vb-mstest.sln
  /PrimeService
    Source Files
    PrimeService.vbproj
  /PrimeService.Tests
    Test Source Files
    PrimeServiceTests.vbproj
```

Execute `dotnet sln add .\PrimeService.Tests\PrimeService.Tests.vbproj` in the *unit-testing-vb-mstest* directory.

Creating the first test

You write one failing test, make it pass, then repeat the process. Remove *UnitTest1.vb* from the *PrimeService.Tests* directory and create a new Visual Basic file named *PrimeService_IsPrimeShould.VB*. Add the following code:

```
Imports Microsoft.VisualStudio.TestTools.UnitTesting

Namespace PrimeService.Tests
  <TestClass>
  Public Class PrimeService_IsPrimeShould
    Private _primeService As Prime.Services.PrimeService = New Prime.Services.PrimeService()

    <TestMethod>
    Sub IsPrime_InputIs1_ReturnFalse()
      Dim result As Boolean = _primeService.IsPrime(1)

      Assert.IsFalse(result, "1 should not be prime")
    End Sub

  End Class
End Namespace
```

The `<TestClass>` attribute indicates a class that contains tests. The `<TestMethod>` attribute denotes a method that is run by the test runner. From the *unit-testing-vb-mstest*, execute `dotnet test` to build the tests and the class library and then run the tests. The MSTest test runner contains the program entry point to run your tests. `dotnet test` starts the test runner using the unit test project you've created.

Your test fails. You haven't created the implementation yet. Make this test pass by writing the simplest code in the `PrimeService` class that works:

```

Public Function IsPrime(candidate As Integer) As Boolean
    If candidate = 1 Then
        Return False
    End If
    Throw New NotImplementedException("Please create a test first.")
End Function

```

In the `unit-testing-vb-mstest` directory, run `dotnet test` again. The `dotnet test` command runs a build for the `PrimeService` project and then for the `PrimeService.Tests` project. After building both projects, it runs this single test. It passes.

Adding more features

Now that you've made one test pass, it's time to write more. There are a few other simple cases for prime numbers: 0, -1. You could add those cases as new tests with the `<TestMethod>` attribute, but that quickly becomes tedious. There are other MSTest attributes that enable you to write a suite of similar tests. A `<DataTestMethod>` attribute represents a suite of tests that execute the same code but have different input arguments. You can use the `<DataRow>` attribute to specify values for those inputs.

Instead of creating new tests, apply these two attributes to create a single theory. The theory is a method that tests several values less than two, which is the lowest prime number:

```

<TestClass>
Public Class PrimeService_IsPrimeShould
    Private _primeService As Prime.Services.PrimeService = New Prime.Services.PrimeService()

    <DataTestMethod>
    <DataRow(-1)>
    <DataRow(0)>
    <DataRow(1)>
    Sub IsPrime_ValuesLessThan2_ReturnFalse(value As Integer)
        Dim result As Boolean = _primeService.IsPrime(value)

        Assert.IsFalse(result, $"{value} should not be prime")
    End Sub

    <DataTestMethod>
    <DataRow(2)>
    <DataRow(3)>
    <DataRow(5)>
    <DataRow(7)>
    Public Sub IsPrime_PrimesLessThan10_ReturnTrue(value As Integer)
        Dim result As Boolean = _primeService.IsPrime(value)

        Assert.IsTrue(result, $"{value} should be prime")
    End Sub

    <DataTestMethod>
    <DataRow(4)>
    <DataRow(6)>
    <DataRow(8)>
    <DataRow(9)>
    Public Sub IsPrime_NonPrimesLessThan10_ReturnFalse(value As Integer)
        Dim result As Boolean = _primeService.IsPrime(value)

        Assert.IsFalse(result, $"{value} should not be prime")
    End Sub
End Class

```

Run `dotnet test`, and two of these tests fail. To make all of the tests pass, change the `if` clause at the beginning of the method:

```
if candidate < 2
```

Continue to iterate by adding more tests, more theories, and more code in the main library. You have the [finished version of the tests](#) and the [complete implementation of the library](#).

You've built a small library and a set of unit tests for that library. You've structured the solution so that adding new packages and tests is part of the normal workflow. You've concentrated most of your time and effort on solving the goals of the application.

Run selected unit tests

9/20/2022 • 4 minutes to read • [Edit Online](#)

With the `dotnet test` command in .NET Core, you can use a filter expression to run selected tests. This article demonstrates how to filter tests. The examples use `dotnet test`. If you're using `vstest.console.exe`, replace `--filter` with `--testcasefilter:`.

Syntax

```
dotnet test --filter <Expression>
```

- **Expression** is in the format `<Property><Operator><Value>[| & <Expression>]`.

Expressions can be joined with boolean operators: `|` for boolean **or**, `&` for boolean **and**.

Expressions can be enclosed in parentheses. For example: `(Name~MyClass) | (Name~MyClass2)`.

An expression without any **operator** is interpreted as a **contains** on the `FullyQualifiedName` property.

For example, `dotnet test --filter xyz` is the same as `dotnet test --filter FullyQualifiedName~xyz`.

- **Property** is an attribute of the `Test Case`. For example, the following properties are supported by popular unit test frameworks.

TEST FRAMEWORK	SUPPORTED PROPERTIES
MSTest	<code>FullyQualifiedName</code> <code>Name</code> <code>ClassName</code> <code>Priority</code> <code>TestCategory</code>
xUnit	<code>FullyQualifiedName</code> <code>DisplayName</code> <code>Traits</code>
Nunit	<code>FullyQualifiedName</code> <code>Name</code> <code>Priority</code> <code>TestCategory</code>

- **Operators**

- `=` exact match
- `!=` not exact match
- `~` contains
- `!~` doesn't contain

- **Value** is a string. All the lookups are case insensitive.

Character escaping

To use an exclamation mark (!) in a filter expression, you have to escape it in some Linux or macOS shells by putting a backslash in front of it (\!). For example, the following filter skips all tests in a namespace that contains `IntegrationTests`:

```
dotnet test --filter FullyQualifiedNamespace\!~IntegrationTests
```

For `FullyQualifiedNamespace` values that include a comma for generic type parameters, escape the comma with %2C. For example:

```
dotnet test --filter "FullyQualifiedNamespace=MyNamespace.MyTestsClass<ParameterType1%2CParameterType2>.MyTestMethod"
```

MSTest examples

```
using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace MSTestNamespace
{
    [TestClass]
    public class UnitTest1
    {
        [TestMethod, Priority(1), TestCategory("CategoryA")]
        public void TestMethod1()
        {
        }

        [TestMethod, Priority(2)]
        public void TestMethod2()
        {
        }
    }
}
```

EXPRESSION	RESULT
<code>dotnet test --filter Method</code>	Runs tests whose <code>FullyQualifiedNamespace</code> contains <code>Method</code> .
<code>dotnet test --filter Name~TestMethod1</code>	Runs tests whose name contains <code>TestMethod1</code> .
<code>dotnet test --filter ClassName=MSTestNamespace.UnitTest1</code>	Runs tests that are in class <code>MSTestNamespace.UnitTest1</code> . Note: The <code>ClassName</code> value should have a namespace, so <code>ClassName=UnitTest1</code> won't work.
<code>dotnet test --filter FullyQualifiedNamespace!=MSTestNamespace.UnitTest1.TestMethod1</code>	Runs all tests except <code>MSTestNamespace.UnitTest1.TestMethod1</code> .
<code>dotnet test --filter TestCategory=CategoryA</code>	Runs tests that are annotated with <code>[TestCategory("CategoryA")]</code> .
<code>dotnet test --filter Priority=2</code>	Runs tests that are annotated with <code>[Priority(2)]</code> .

Examples using the conditional operators | and &:

- To run tests that have `UnitTest1` in their `FullyQualifiedNamespace` or `TestCategoryAttribute` is `"CategoryA"`.

```
dotnet test --filter "FullyQualifiedName~UnitTest1|TestCategory=CategoryA"
```

- To run tests that have `UnitTest1` in their `FullyQualifiedName` and have a `TestCategoryAttribute` of `"CategoryA"`.

```
dotnet test --filter "FullyQualifiedName~UnitTest1&TestCategory=CategoryA"
```

- To run tests that have either `FullyQualifiedName` containing `UnitTest1` and have a `TestCategoryAttribute` of `"CategoryA"` or have a `PriorityAttribute` with a priority of `1`.

```
dotnet test --filter "(FullyQualifiedName~UnitTest1&TestCategory=CategoryA)|Priority=1"
```

xUnit examples

```
using Xunit;

namespace XUnitNamespace
{
    public class TestClass1
    {
        [Fact, Trait("Priority", "1"), Trait("Category", "CategoryA")]
        public void Test1()
        {
        }

        [Fact, Trait("Priority", "2")]
        public void Test2()
        {
        }
    }
}
```

EXPRESSION	RESULT
<code>dotnet test --filter DisplayName=XUnitNamespace.TestClass1.Test1</code>	Runs only one test, <code>XUnitNamespace.TestClass1.Test1</code> .
<code>dotnet test --filter FullyQualifiedName!=XUnitNamespace.TestClass1.Test1</code>	Runs all tests except <code>XUnitNamespace.TestClass1.Test1</code> .
<code>dotnet test --filter DisplayName~TestClass1</code>	Runs tests whose display name contains <code>TestClass1</code> .

In the code example, the defined traits with keys `"Category"` and `"Priority"` can be used for filtering.

EXPRESSION	RESULT
<code>dotnet test --filter XUnit</code>	Runs tests whose <code>FullyQualifiedName</code> contains <code>XUnit</code> .
<code>dotnet test --filter Category=CategoryA</code>	Runs tests that have <code>[Trait("Category", "CategoryA")]</code> .

Examples using the conditional operators `|` and `&`:

- To run tests that have `TestClass1` in their `FullyQualifiedName` or have a `Trait` with a key of `"Category"` and value of `"CategoryA"`.

```
dotnet test --filter "FullyQualifiedName~TestClass1|Category=CategoryA"
```

- To run tests that have `TestClass1` in their `FullyQualifiedName` and have a `Trait` with a key of `"Category"` and value of `"CategoryA"`.

```
dotnet test --filter "FullyQualifiedName~TestClass1&Category=CategoryA"
```

- To run tests that have either `FullyQualifiedName` containing `TestClass1` and have a `Trait` with a key of `"Category"` and value of `"CategoryA"` or have a `Trait` with a key of `"Priority"` and value of `1`.

```
dotnet test --filter "(FullyQualifiedName~TestClass1&Category=CategoryA)|Priority=1"
```

NUnit examples

```
using NUnit.Framework;

namespace NUnitNamespace
{
    public class UnitTest1
    {
        [Test, Property("Priority", 1), Category("CategoryA")]
        public void TestMethod1()
        {
        }

        [Test, Property("Priority", 2)]
        public void TestMethod2()
        {
        }
    }
}
```

EXPRESSION	RESULT
<code>dotnet test --filter Method</code>	Runs tests whose <code>FullyQualifiedName</code> contains <code>Method</code> .
<code>dotnet test --filter Name~TestMethod1</code>	Runs tests whose name contains <code>TestMethod1</code> .
<code>dotnet test --filter FullyQualifiedName~NUnitNamespace.UnitTest1</code>	Runs tests that are in class <code>NUnitNamespace.UnitTest1</code> .
<code>dotnet test --filter FullyQualifiedName!=NUnitNamespace.UnitTest1.TestMethod1</code>	Runs all tests except <code>NUnitNamespace.UnitTest1.TestMethod1</code> .
<code>dotnet test --filterTestCategory=CategoryA</code>	Runs tests that are annotated with <code>[Category("CategoryA")]</code> .
<code>dotnet test --filter Priority=2</code>	Runs tests that are annotated with <code>[Priority(2)]</code> .

Examples using the conditional operators `|` and `&`:

To run tests that have `UnitTest1` in their `FullyQualifiedName` or have a `Category` of `"CategoryA"`.

```
dotnet test --filter "FullyQualifiedNames~UnitTest1|TestCategory=CategoryA"
```

To run tests that have `UnitTest1` in their **FullyQualifiedNames** and have a `Category` of `"CategoryA"`.

```
dotnet test --filter "FullyQualifiedNames~UnitTest1&TestCategory=CategoryA"
```

To run tests that have either a **FullyQualifiedNames** containing `UnitTest1` and have a `Category` of `"CategoryA"` or have a `Property` with a `"Priority"` of `1`.

```
dotnet test --filter "(FullyQualifiedNames~UnitTest1&TestCategory=CategoryA)|Priority=1"
```

For more information, see [TestCase filter](#).

See also

- [dotnet test](#)
- [dotnet test --filter](#)

Next steps

[Order unit tests](#)

Order unit tests

9/20/2022 • 4 minutes to read • [Edit Online](#)

Occasionally, you may want to have unit tests run in a specific order. Ideally, the order in which unit tests run should *not* matter, and it is [best practice](#) to avoid ordering unit tests. Regardless, there may be a need to do so. In that case, this article demonstrates how to order test runs.

If you prefer to browse the source code, see the [order .NET Core unit tests](#) sample repository.

TIP

In addition to the ordering capabilities outlined in this article, consider [creating custom playlists with Visual Studio](#) as an alternative.

Order alphabetically

With MSTest, tests are automatically ordered by their test name.

NOTE

A test named `Test14` will run before `Test2` even though the number `2` is less than `14`. This is because, test name ordering uses the text name of the test.

```
using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace MSTest.Project
{
    [TestClass]
    public class ByAlphabeticalOrder
    {
        public static bool Test1Called;
        public static bool Test2Called;
        public static bool Test3Called;

        [TestMethod]
        public void Test2()
        {
            Test2Called = true;

            Assert.IsTrue(Test1Called);
            Assert.IsFalse(Test3Called);
        }

        [TestMethod]
        public void Test1()
        {
            Test1Called = true;

            Assert.IsFalse(Test2Called);
            Assert.IsFalse(Test3Called);
        }

        [TestMethod]
        public void Test3()
        {
            Test3Called = true;

            Assert.IsTrue(Test1Called);
            Assert.IsTrue(Test2Called);
        }
    }
}
```

The xUnit test framework allows for more granularity and control of test run order. You implement the `ITestCaseOrderer` and `ITestCollectionOrderer` interfaces to control the order of test cases for a class, or test collections.

Order by test case alphabetically

To order test cases by their method name, you implement the `ITestCaseOrderer` and provide an ordering mechanism.

```

using System.Collections.Generic;
using System.Linq;
using Xunit.Abstractions;
using Xunit.Sdk;

namespace XUnit.Project.Orderers
{
    public class AlphabeticalOrderer : ITestCaseOrderer
    {
        public IEnumerable<TTestCase> OrderTestCases<TTestCase>(
            IEnumerable<TTestCase> testCases) where TTestCase : ITestCase =>
            testCases.OrderBy(testCase => testCase.TestMethod.Method.Name);
    }
}

```

Then in a test class you set the test case order with the `TestCaseOrdererAttribute`.

```

using Xunit;

namespace XUnit.Project;

[TestCaseOrderer("XUnit.Project.Orderers.AlphabeticalOrderer", "XUnit.Project")]
public class ByAlphabeticalOrder
{
    public static bool Test1Called;
    public static bool Test2Called;
    public static bool Test3Called;

    [Fact]
    public void Test1()
    {
        Test1Called = true;

        Assert.False(Test2Called);
        Assert.False(Test3Called);
    }

    [Fact]
    public void Test2()
    {
        Test2Called = true;

        Assert.True(Test1Called);
        Assert.False(Test3Called);
    }

    [Fact]
    public void Test3()
    {
        Test3Called = true;

        Assert.True(Test1Called);
        Assert.True(Test2Called);
    }
}

```

Order by collection alphabetically

To order test collections by their display name, you implement the `ITestCollectionOrderer` and provide an ordering mechanism.

```
using System.Collections.Generic;
using System.Linq;
using Xunit;
using Xunit.Abstractions;

namespace XUnit.Project.Orderers
{
    public class DisplayNameOrderer : ITestCollectionOrderer
    {
        public IEnumerable<ITestCollection> OrderTestCollections(
            IEnumerable<ITestCollection> testCollections) =>
            testCollections.OrderBy(collection => collection.DisplayName);
    }
}
```

Since test collections potentially run in parallel, you must explicitly disable test parallelization of the collections with the `CollectionBehaviorAttribute`. Then specify the implementation to the `TestCollectionOrdererAttribute`.

```

using Xunit;

// Need to turn off test parallelization so we can validate the run order
[assembly: CollectionBehavior(DisableTestParallelization = true)]
[assembly: TestCollectionOrderer("XUnit.Project.Orderers.DisplayNameOrderer", "XUnit.Project")]

namespace XUnit.Project;

[Collection("Xzy Test Collection")]
public class TestsInCollection1
{
    public static bool Collection1Run;

    [Fact]
    public static void Test()
    {
        Assert.True(TestsInCollection2.Collection2Run); // Abc
        Assert.True(TestsInCollection3.Collection3Run); // Mno
        Assert.False(TestsInCollection1.Collection1Run); // Xyz

        Collection1Run = true;
    }
}

[Collection("Abc Test Collection")]
public class TestsInCollection2
{
    public static bool Collection2Run;

    [Fact]
    public static void Test()
    {
        Assert.False(TestsInCollection2.Collection2Run); // Abc
        Assert.False(TestsInCollection3.Collection3Run); // Mno
        Assert.False(TestsInCollection1.Collection1Run); // Xyz

        Collection2Run = true;
    }
}

[Collection("Mno Test Collection")]
public class TestsInCollection3
{
    public static bool Collection3Run;

    [Fact]
    public static void Test()
    {
        Assert.True(TestsInCollection2.Collection2Run); // Abc
        Assert.False(TestsInCollection3.Collection3Run); // Mno
        Assert.False(TestsInCollection1.Collection1Run); // Xyz

        Collection3Run = true;
    }
}

```

Order by custom attribute

To order xUnit tests with custom attributes, you first need an attribute to rely on. Define a `TestPriorityAttribute` as follows:

```

using System;

namespace XUnit.Project.Attributes
{
    [AttributeUsage(AttributeTargets.Method, AllowMultiple = false)]
    public class TestPriorityAttribute : Attribute
    {
        public int Priority { get; private set; }

        public TestPriorityAttribute(int priority) => Priority = priority;
    }
}

```

Next, consider the following `PriorityOrderer` implementation of the `ITestCaseOrderer` interface.

```

using System.Collections.Generic;
using System.Linq;
using Xunit.Abstractions;
using Xunit.Sdk;
using XUnit.Project.Attributes;

namespace XUnit.Project.Orderers
{
    public class PriorityOrderer : ITestCaseOrderer
    {
        public IEnumerable<TTestCase> OrderTestCases<TTestCase>(
            IEnumerable<TTestCase> testCases) where TTestCase : ITestCase
        {
            string assemblyName = typeof(TestPriorityAttribute).AssemblyQualifiedName!;
            var sortedMethods = new SortedDictionary<int, List<TTestCase>>();
            foreach (TTestCase testCase in testCases)
            {
                int priority = testCase.TestMethod.Method
                    .GetCustomAttributes(assemblyName)
                    .FirstOrDefault()
                    ?.GetNamedArgument<int>(nameof(TestPriorityAttribute.Priority)) ?? 0;

                GetOrCreate(sortedMethods, priority).Add(testCase);
            }

            foreach (TTestCase testCase in
                sortedMethods.Keys.SelectMany(
                    priority => sortedMethods[priority].OrderBy(
                        testCase => testCase.TestMethod.Method.Name)))
            {
                yield return testCase;
            }
        }

        private static TValue GetOrCreate<TKey, TValue>(
            IDictionary<TKey, TValue> dictionary, TKey key)
            where TKey : struct
            where TValue : new() =>
        {
            dictionary.TryGetValue(key, out TValue? result)
            ? result
            : (dictionary[key] = new TValue());
        }
    }
}

```

Then in a test class you set the test case order with the `TestCaseOrdererAttribute` to the `PriorityOrderer`.

```

using Xunit;
using XUnit.Project.Attributes;

namespace XUnit.Project;

[TestCaseOrderer("XUnit.Project.Orderers.PriorityOrderer", "XUnit.Project")]
public class ByPriorityOrder
{
    public static bool Test1Called;
    public static bool Test2ACalled;
    public static bool Test2BCalled;
    public static bool Test3Called;

    [Fact, TestPriority(5)]
    public void Test3()
    {
        Test3Called = true;

        Assert.True(Test1Called);
        Assert.True(Test2ACalled);
        Assert.True(Test2BCalled);
    }

    [Fact, TestPriority(0)]
    public void Test2B()
    {
        Test2BCalled = true;

        Assert.True(Test1Called);
        Assert.True(Test2ACalled);
        Assert.False(Test3Called);
    }

    [Fact]
    public void Test2A()
    {
        Test2ACalled = true;

        Assert.True(Test1Called);
        Assert.False(Test2BCalled);
        Assert.False(Test3Called);
    }

    [Fact, TestPriority(-5)]
    public void Test1()
    {
        Test1Called = true;

        Assert.False(Test2ACalled);
        Assert.False(Test2BCalled);
        Assert.False(Test3Called);
    }
}

```

Order by priority

To order tests explicitly, NUnit provides an [OrderAttribute](#). Tests with this attribute are started before tests without. The order value is used to determine the order to run the unit tests.

```
using NUnit.Framework;

namespace NUnit.Project
{
    public class ByOrder
    {
        public static bool Test1Called;
        public static bool Test2ACalled;
        public static bool Test2BCalled;
        public static bool Test3Called;

        [Test, Order(5)]
        public void Test1()
        {
            Test1Called = true;

            Assert.IsFalse(Test2ACalled);
            Assert.IsTrue(Test2BCalled);
            Assert.IsTrue(Test3Called);
        }

        [Test, Order(0)]
        public void Test2B()
        {
            Test2BCalled = true;

            Assert.IsFalse(Test1Called);
            Assert.IsFalse(Test2ACalled);
            Assert.IsTrue(Test3Called);
        }

        [Test]
        public void Test2A()
        {
            Test2ACalled = true;

            Assert.IsTrue(Test1Called);
            Assert.IsTrue(Test2BCalled);
            Assert.IsTrue(Test3Called);
        }

        [Test, Order(-5)]
        public void Test3()
        {
            Test3Called = true;

            Assert.IsFalse(Test1Called);
            Assert.IsFalse(Test2ACalled);
            Assert.IsFalse(Test2BCalled);
        }
    }
}
```

Next Steps

[Unit test code coverage](#)

Use code coverage for unit testing

9/20/2022 • 6 minutes to read • [Edit Online](#)

IMPORTANT

This article explains the creation of the example project. If you already have a project, you can skip ahead to the [Code coverage tooling](#) section.

Unit tests help to ensure functionality and provide a means of verification for refactoring efforts. Code coverage is a measurement of the amount of code that is run by unit tests - either lines, branches, or methods. As an example, if you have a simple application with only two conditional branches of code (*branch a*, and *branch b*), a unit test that verifies conditional *branch a* will report branch code coverage of 50%.

This article discusses the usage of code coverage for unit testing with Coverlet and report generation using ReportGenerator. While this article focuses on C# and xUnit as the test framework, both MSTest and NUnit would also work. Coverlet is an [open source project on GitHub](#) that provides a cross-platform code coverage framework for C#. [Coverlet](#) is part of the .NET foundation. Coverlet collects Cobertura coverage test run data, which is used for report generation.

Additionally, this article details how to use the code coverage information collected from a Coverlet test run to generate a report. The report generation is possible using another [open source project on GitHub](#) - [ReportGenerator](#). ReportGenerator converts coverage reports generated by Cobertura among many others, into human-readable reports in various formats.

This article is based on the [sample source code project](#), available on samples browser.

System under test

The "system under test" refers to the code that you're writing unit tests against, this could be an object, service, or anything else that exposes testable functionality. For this article, you'll create a class library that will be the system under test, and two corresponding unit test projects.

Create a class library

From a command prompt in a new directory named `UnitTestingCodeCoverage`, create a new .NET standard class library using the `dotnet new classlib` command:

```
dotnet new classlib -n Numbers
```

The snippet below defines a simple `PrimeService` class that provides functionality to check if a number is prime. Copy the snippet below and replace the contents of the *Class1.cs* file that was automatically created in the *Numbers* directory. Rename the *Class1.cs* file to *PrimeService.cs*.

```

namespace System.Numbers
{
    public class PrimeService
    {
        public bool IsPrime(int candidate)
        {
            if (candidate < 2)
            {
                return false;
            }

            for (int divisor = 2; divisor <= Math.Sqrt(candidate); ++divisor)
            {
                if (candidate % divisor == 0)
                {
                    return false;
                }
            }
            return true;
        }
    }
}

```

TIP

It is worth mentioning that the `Numbers` class library was intentionally added to the `System` namespace. This allows for `System.Math` to be accessible without a `using System;` namespace declaration. For more information, see [namespace \(C# Reference\)](#).

Create test projects

Create two new **xUnit Test Project (.NET Core)** templates from the same command prompt using the

`dotnet new xunit` command:

```
dotnet new xunit -n XUnit.Coverlet.Collector
```

```
dotnet new xunit -n XUnit.Coverlet.MSBuild
```

Both of the newly created xUnit test projects need to add a project reference of the *Numbers* class library. This is so that the test projects have access to the *PrimeService* for testing. From the command prompt, use the

`dotnet add` command:

```
dotnet add XUnit.Coverlet.Collector\XUnit.Coverlet.Collector.csproj reference Numbers\Numbers.csproj
```

```
dotnet add XUnit.Coverlet.MSBuild\XUnit.Coverlet.MSBuild.csproj reference Numbers\Numbers.csproj
```

The *MSBuild* project is named appropriately, as it will depend on the `coverlet.msbuild` NuGet package. Add this package dependency by running the `dotnet add package` command:

```
cd XUnit.Coverlet.MSBuild && dotnet add package coverlet.msbuild && cd ..
```

The previous command changed directories effectively scoping to the *MSBuild* test project, then added the NuGet package. When that was done, it then changed directories, stepping up one level.

Open both of the *UnitTest1.cs* files, and replace their contents with the following snippet. Rename the *UnitTest1.cs* files to *PrimeServiceTests.cs*.

```
using System.Numerics;
using Xunit;

namespace XUnit.Coverlet
{
    public class PrimeServiceTests
    {
        readonly PrimeService _primeService;

        public PrimeServiceTests() => _primeService = new PrimeService();

        [
            Theory,
            InlineData(-1), InlineData(0), InlineData(1)
        ]
        public void IsPrime_ValuesLessThan2_ReturnFalse(int value) =>
            Assert.False(_primeService.IsPrime(value), $"{value} should not be prime");

        [
            Theory,
            InlineData(2), InlineData(3), InlineData(5), InlineData(7)
        ]
        public void IsPrime_PrimesLessThan10_ReturnTrue(int value) =>
            Assert.True(_primeService.IsPrime(value), $"{value} should be prime");

        [
            Theory,
            InlineData(4), InlineData(6), InlineData(8), InlineData(9)
        ]
        public void IsPrime_NonPrimesLessThan10_ReturnFalse(int value) =>
            Assert.False(_primeService.IsPrime(value), $"{value} should not be prime");
    }
}
```

Create a solution

From the command prompt, create a new solution to encapsulate the class library and the two test projects.

Using the `dotnet sln` command:

```
dotnet new sln -n XUnit.Coverage
```

This will create a new solution file name `XUnit.Coverage.sln` in the *UnitTestingCodeCoverage* directory. Add the projects to the root of the solution.

- [Linux](#)
- [Windows](#)

```
dotnet sln XUnit.Coverage.sln add **/*.csproj --in-root
```

Build the solution using the `dotnet build` command:

```
dotnet build
```

If the build is successful, you've created the three projects, appropriately referenced projects and packages, and updated the source code correctly. Well done!

Code coverage tooling

There are two types of code coverage tools:

- **DataCollectors:** DataCollectors monitor test execution and collect information about test runs. They report the collected information in various output formats, such as XML and JSON. For more information, see [your first DataCollector](#).
- **Report generators:** Use data collected from test runs to generate reports, often as styled HTML.

In this section, the focus is on data collector tools. To use Coverlet for code coverage, an existing unit test project must have the appropriate package dependencies, or alternatively rely on [.NET global tooling](#) and the corresponding [coverlet.console](#) NuGet package.

Integrate with .NET test

The xUnit test project template already integrates with [coverlet.collector](#) by default. From the command prompt, change directories to the *XUnit.Coverlet.Collector* project, and run the `dotnet test` command:

```
cd XUnit.Coverlet.Collector && dotnet test --collect:"XPlat Code Coverage"
```

NOTE

The `"XPlat Code Coverage"` argument is a friendly name that corresponds to the data collectors from Coverlet. This name is required but is case insensitive.

As part of the `dotnet test` run, a resulting *coverage.cobertura.xml* file is output to the *TestResults* directory. The XML file contains the results. This is a cross-platform option that relies on the .NET CLI, and it is great for build systems where MSBuild is not available.

Below is the example *coverage.cobertura.xml* file.

```
<?xml version="1.0" encoding="utf-8"?>
<coverage line-rate="1" branch-rate="1" version="1.9" timestamp="1592248008"
    lines-covered="12" lines-valid="12" branches-covered="6" branches-valid="6">
    <sources>
        <source>C:\</source>
    </sources>
    <packages>
        <package name="Numbers" line-rate="1" branch-rate="1" complexity="6">
            <classes>
                <class name="Numbers.PrimeService" line-rate="1" branch-rate="1" complexity="6"
                    filename="Numbers\PrimeService.cs">
                    <methods>
                        <method name="IsPrime" signature="(System.Int32)" line-rate="1"
                            branch-rate="1" complexity="6">
                            <lines>
                                <line number="8" hits="11" branch="False" />
                                <line number="9" hits="11" branch="True" condition-coverage="100% (2/2)">
                                    <conditions>
                                        <condition number="7" type="jump" coverage="100%" />
                                    </conditions>
                                </line>
                                <line number="10" hits="3" branch="False" />
                                <line number="11" hits="3" branch="False" />
                                <line number="14" hits="22" branch="True" condition-coverage="100% (2/2)">
                                    <conditions>
                                        <condition number="57" type="jump" coverage="100%" />
                                    </conditions>
                                </line>
                            </lines>
                        </method>
                    </methods>
                </class>
            </classes>
        </package>
    </packages>
</coverage>
```

```

<line number="15" hits="7" branch="False" />
<line number="16" hits="7" branch="True" condition-coverage="100% (2/2)">
  <conditions>
    <condition number="27" type="jump" coverage="100%" />
  </conditions>
</line>
<line number="17" hits="4" branch="False" />
<line number="18" hits="4" branch="False" />
<line number="20" hits="3" branch="False" />
<line number="21" hits="4" branch="False" />
<line number="23" hits="11" branch="False" />
</lines>
</method>
</methods>
<lines>
  <line number="8" hits="11" branch="False" />
  <line number="9" hits="11" branch="True" condition-coverage="100% (2/2)">
    <conditions>
      <condition number="7" type="jump" coverage="100%" />
    </conditions>
  </line>
  <line number="10" hits="3" branch="False" />
  <line number="11" hits="3" branch="False" />
  <line number="14" hits="22" branch="True" condition-coverage="100% (2/2)">
    <conditions>
      <condition number="57" type="jump" coverage="100%" />
    </conditions>
  </line>
  <line number="15" hits="7" branch="False" />
  <line number="16" hits="7" branch="True" condition-coverage="100% (2/2)">
    <conditions>
      <condition number="27" type="jump" coverage="100%" />
    </conditions>
  </line>
  <line number="17" hits="4" branch="False" />
  <line number="18" hits="4" branch="False" />
  <line number="20" hits="3" branch="False" />
  <line number="21" hits="4" branch="False" />
  <line number="23" hits="11" branch="False" />
</lines>
</class>
</classes>
</package>
</packages>
</coverage>

```

TIP

As an alternative, you could use the MSBuild package if your build system already makes use of MSBuild. From the command prompt, change directories to the *XUnit.Coverlet.MSBuild* project, and run the `dotnet test` command:

```
dotnet test /p:CollectCoverage=true /p:CoverletOutputFormat=cobertura
```

The resulting *coverage.cobertura.xml* file is output. You can follow MSBuild integration guide [here](#)

Generate reports

Now that you're able to collect data from unit test runs, you can generate reports using [ReportGenerator](#). To install the [ReportGenerator](#) NuGet package as a [.NET global tool](#), use the `dotnet tool install` command:

```
dotnet tool install -g dotnet-reportgenerator-globaltool
```

Run the tool and provide the desired options, given the output `coverage.cobertura.xml` file from the previous test run.

```
reportgenerator
-reports:"Path\To\TestProject\TestResults\{guid}\coverage.cobertura.xml"
-targetdir:"coverageReport"
-reporttypes:Html
```

After running this command, an HTML file represents the generated report.

The screenshot shows a browser window displaying an HTML-based code coverage report. The title bar says "System.Numbers.PrimeService -". The address bar shows the URL: "/UnitTestingCodeCoverage/coverageReport/Numbers_Prime...". The report is organized into several sections:

- Summary:** Shows metrics for the class: System.Numbers.PrimeService, Assembly: Numbers, File(s): \UnitTestingCodeCoverage\Numbers\PrimeService.cs. It includes:
 - Covered lines: 12
 - Uncovered lines: 0
 - Coverable lines: 12
 - Total lines: 22
 - Line coverage: 100% (12 of 12)
 - Covered branches: 6
 - Total branches: 6
 - Branch coverage: 100% (6 of 6)
- Metrics:** A table showing method coverage for IsPrime(...).

Method	Line coverage	Branch coverage
IsPrime(...)	100%	100%
- File(s):** Shows the PrimeService.cs file with line numbers and coverage status.

#	Line	Line coverage
1	namespace System.Numbers	
2	{	
3	public class PrimeService	
4	{	
5	public bool IsPrime(int candidate)	
6	{	
7	if (candidate < 2)	
8	{	
9	return false;	
10	}	
11	}	
12	for (int divisor = 2; divisor <= Math.Sqrt(candidate); ++divisor)	
13	{	
14	if (candidate % divisor == 0)	
15	{	
16	return false;	
17	}	
18	}	
19	return true;	
20	}	
21	}	

See also

- [Visual Studio unit test code coverage](#)
- [GitHub - Coverlet repository](#)
- [GitHub - ReportGenerator repository](#)
- [ReportGenerator project site](#)
- [Azure: Publish code coverage results](#)
- [Azure: Review code coverage results](#)
- [.NET CLI test command](#)
- [dotnet-coverage](#)
- [Sample source code](#)

Next Steps

[Unit testing best practices](#)

Test published output with dotnet vstest

9/20/2022 • 2 minutes to read • [Edit Online](#)

You can run tests on already published output by using the `dotnet vstest` command. This will work on xUnit, MSTest, and NUnit tests. Simply locate the DLL file that was part of your published output and run:

```
dotnet vstest <MyPublishedTests>.dll
```

Where `<MyPublishedTests>` is the name of your published test project.

Example

The commands below demonstrate running tests on a published DLL.

```
dotnet new mstest -o MyProject.Tests
cd MyProject.Tests
dotnet publish -o out
dotnet vstest out/MyProject.Tests.dll
```

NOTE

Note: If your app targets a framework other than `netcoreapp`, you can still run the `dotnet vstest` command by passing in the targeted framework with a framework flag. For example,

```
dotnet vstest <MyPublishedTests>.dll --Framework:".NETFramework,Version=v4.6" . In Visual Studio 2017 Update 5 and later, the desired framework is automatically detected.
```

See also

- [Unit Testing with dotnet test and xUnit](#)
- [Unit Testing with dotnet test and NUnit](#)
- [Unit Testing with dotnet test and MSTest](#)

Security in .NET

9/20/2022 • 2 minutes to read • [Edit Online](#)

The common language runtime and .NET provide many useful classes and services that enable developers to write secure code, use cryptography, and implement role-based security.

In this section

- [Key Security Concepts](#)
Provides an overview of common language runtime security features.
- [Role-Based Security](#)
Describes how to interact with role-based security in your code.
- [Cryptography Model](#)
Provides an overview of cryptographic services provided by .NET.
- [Secure Coding Guidelines](#)
Describes some of the best practices for creating reliable .NET applications.

Related sections

[Development Guide](#)

Provides a guide to all key technology areas and tasks for application development, including creating, configuring, debugging, securing, and deploying your application, and information about dynamic programming, interoperability, extensibility, memory management, and threading.

What is "managed code"?

9/20/2022 • 2 minutes to read • [Edit Online](#)

When working with .NET, you will often encounter the term "managed code". This document will explain what this term means and additional information around it.

To put it very simply, managed code is just that: code whose execution is managed by a runtime. In this case, the runtime in question is called the **Common Language Runtime** or CLR, regardless of the implementation (for example, [Mono](#), .NET Framework, or .NET Core/.NET 5+). CLR is in charge of taking the managed code, compiling it into machine code and then executing it. On top of that, runtime provides several important services such as automatic memory management, security boundaries, type safety etc.

Contrast this to the way you would run a C/C++ program, also called "unmanaged code". In the unmanaged world, the programmer is in charge of pretty much everything. The actual program is, essentially, a binary that the operating system (OS) loads into memory and starts. Everything else, from memory management to security considerations are a burden of the programmer.

Managed code is written in one of the high-level languages that can be run on top of .NET, such as C#, Visual Basic, F# and others. When you compile code written in those languages with their respective compiler, you don't get machine code. You get **Intermediate Language** code which the runtime then compiles and executes. C++ is the one exception to this rule, as it can also produce native, unmanaged binaries that run on Windows.

Intermediate Language & execution

What is "Intermediate Language" (or IL for short)? It is a product of compilation of code written in high-level .NET languages. Once you compile your code written in one of these languages, you will get a binary that is made out of IL. It is important to note that the IL is independent from any specific language that runs on top of the runtime; there is even a separate specification for it that you can read if you're so inclined.

Once you produce IL from your high-level code, you will most likely want to run it. This is where the CLR takes over and starts the process of **Just-In-Time** compiling, or **JIT-ing** your code from IL to machine code that can actually be run on a CPU. In this way, the CLR knows exactly what your code is doing and can effectively *manage* it.

Intermediate Language is sometimes also called Common Intermediate Language (CIL) or Microsoft Intermediate Language (MSIL).

Unmanaged code interoperability

Of course, the CLR allows passing the boundaries between managed and unmanaged world, and there is a lot of code that does that, even in the [Base Class Libraries](#). This is called **interoperability** or just **interop** for short. These provisions would allow you to, for example, wrap up an unmanaged library and call into it. However, it is important to note that once you do this, when the code passes the boundaries of the runtime, the actual management of the execution is again in the hand of unmanaged code, and thus falls under the same restrictions.

Similar to this, C# is one language that allows you to use unmanaged constructs such as pointers directly in code by utilizing what is known as **unsafe context** which designates a piece of code for which the execution is not managed by the CLR.

More resources

- Overview of .NET Framework
- Unsafe Code and Pointers
- Native interoperability

Automatic Memory Management

9/20/2022 • 7 minutes to read • [Edit Online](#)

Automatic memory management is one of the services that the Common Language Runtime provides during [Managed Execution](#). The Common Language Runtime's garbage collector manages the allocation and release of memory for an application. For developers, this means that you do not have to write code to perform memory management tasks when you develop managed applications. Automatic memory management can eliminate common problems, such as forgetting to free an object and causing a memory leak, or attempting to access memory for an object that has already been freed. This section describes how the garbage collector allocates and releases memory.

Allocating Memory

When you initialize a new process, the runtime reserves a contiguous region of address space for the process. This reserved address space is called the managed heap. The managed heap maintains a pointer to the address where the next object in the heap will be allocated. Initially, this pointer is set to the managed heap's base address. All [reference types](#) are allocated on the managed heap. When an application creates the first reference type, memory is allocated for the type at the base address of the managed heap. When the application creates the next object, the garbage collector allocates memory for it in the address space immediately following the first object. As long as address space is available, the garbage collector continues to allocate space for new objects in this manner.

Allocating memory from the managed heap is faster than unmanaged memory allocation. Because the runtime allocates memory for an object by adding a value to a pointer, it is almost as fast as allocating memory from the stack. In addition, because new objects that are allocated consecutively are stored contiguously in the managed heap, an application can access the objects very quickly.

Releasing Memory

The garbage collector's optimizing engine determines the best time to perform a collection based on the allocations being made. When the garbage collector performs a collection, it releases the memory for objects that are no longer being used by the application. It determines which objects are no longer being used by examining the application's roots. Every application has a set of roots. Each root either refers to an object on the managed heap or is set to null. An application's roots include static fields, local variables and parameters on a thread's stack, and CPU registers. The garbage collector has access to the list of active roots that the [just-in-time \(JIT\) compiler](#) and the runtime maintain. Using this list, it examines an application's roots, and in the process creates a graph that contains all the objects that are reachable from the roots.

Objects that are not in the graph are unreachable from the application's roots. The garbage collector considers unreachable objects garbage and will release the memory allocated for them. During a collection, the garbage collector examines the managed heap, looking for the blocks of address space occupied by unreachable objects. As it discovers each unreachable object, it uses a memory-copying function to compact the reachable objects in memory, freeing up the blocks of address spaces allocated to unreachable objects. Once the memory for the reachable objects has been compacted, the garbage collector makes the necessary pointer corrections so that the application's roots point to the objects in their new locations. It also positions the managed heap's pointer after the last reachable object. Note that memory is compacted only if a collection discovers a significant number of unreachable objects. If all the objects in the managed heap survive a collection, then there is no need for memory compaction.

To improve performance, the runtime allocates memory for large objects in a separate heap. The garbage

collector automatically releases the memory for large objects. However, to avoid moving large objects in memory, this memory is not compacted.

Generations and Performance

To optimize the performance of the garbage collector, the managed heap is divided into three generations: 0, 1, and 2. The runtime's garbage collection algorithm is based on several generalizations that the computer software industry has discovered to be true by experimenting with garbage collection schemes. First, it is faster to compact the memory for a portion of the managed heap than for the entire managed heap. Secondly, newer objects will have shorter lifetimes and older objects will have longer lifetimes. Lastly, newer objects tend to be related to each other and accessed by the application around the same time.

The runtime's garbage collector stores new objects in generation 0. Objects created early in the application's lifetime that survive collections are promoted and stored in generations 1 and 2. The process of object promotion is described later in this topic. Because it is faster to compact a portion of the managed heap than the entire heap, this scheme allows the garbage collector to release the memory in a specific generation rather than release the memory for the entire managed heap each time it performs a collection.

In reality, the garbage collector performs a collection when generation 0 is full. If an application attempts to create a new object when generation 0 is full, the garbage collector discovers that there is no address space remaining in generation 0 to allocate for the object. The garbage collector performs a collection in an attempt to free address space in generation 0 for the object. The garbage collector starts by examining the objects in generation 0 rather than all objects in the managed heap. This is the most efficient approach, because new objects tend to have short lifetimes, and it is expected that many of the objects in generation 0 will no longer be in use by the application when a collection is performed. In addition, a collection of generation 0 alone often reclaims enough memory to allow the application to continue creating new objects.

After the garbage collector performs a collection of generation 0, it compacts the memory for the reachable objects as explained in [Releasing Memory](#) earlier in this topic. The garbage collector then promotes these objects and considers this portion of the managed heap generation 1. Because objects that survive collections tend to have longer lifetimes, it makes sense to promote them to a higher generation. As a result, the garbage collector does not have to reexamine the objects in generations 1 and 2 each time it performs a collection of generation 0.

After the garbage collector performs its first collection of generation 0 and promotes the reachable objects to generation 1, it considers the remainder of the managed heap generation 0. It continues to allocate memory for new objects in generation 0 until generation 0 is full and it is necessary to perform another collection. At this point, the garbage collector's optimizing engine determines whether it is necessary to examine the objects in older generations. For example, if a collection of generation 0 does not reclaim enough memory for the application to successfully complete its attempt to create a new object, the garbage collector can perform a collection of generation 1, then generation 2. If this does not reclaim enough memory, the garbage collector can perform a collection of generations 2, 1, and 0. After each collection, the garbage collector compacts the reachable objects in generation 0 and promotes them to generation 1. Objects in generation 1 that survive collections are promoted to generation 2. Because the garbage collector supports only three generations, objects in generation 2 that survive a collection remain in generation 2 until they are determined to be unreachable in a future collection.

Releasing Memory for Unmanaged Resources

For the majority of the objects that your application creates, you can rely on the garbage collector to automatically perform the necessary memory management tasks. However, unmanaged resources require explicit cleanup. The most common type of unmanaged resource is an object that wraps an operating system resource, such as a file handle, window handle, or network connection. Although the garbage collector is able to track the lifetime of a managed object that encapsulates an unmanaged resource, it does not have specific

knowledge about how to clean up the resource. When you create an object that encapsulates an unmanaged resource, it is recommended that you provide the necessary code to clean up the unmanaged resource in a public **Dispose** method. By providing a **Dispose** method, you enable users of your object to explicitly free its memory when they are finished with the object. When you use an object that encapsulates an unmanaged resource, you should be aware of **Dispose** and call it as necessary. For more information about cleaning up unmanaged resources and an example of a design pattern for implementing **Dispose**, see [Garbage Collection](#).

See also

- [GC](#)
- [Garbage Collection](#)
- [Managed Execution Process](#)

Cleaning up unmanaged resources

9/20/2022 • 2 minutes to read • [Edit Online](#)

For a majority of the objects that your app creates, you can rely on the [.NET garbage collector](#) to handle memory management. However, when you create objects that include unmanaged resources, you **must** explicitly release those resources when you finish using them. The most common types of unmanaged resources are objects that wrap operating system resources, such as files, windows, network connections, or database connections. Although the garbage collector is able to track the lifetime of an object that encapsulates an unmanaged resource, it doesn't know how to release and clean up the unmanaged resource.

If your types use unmanaged resources, you should do the following:

- Implement the [dispose pattern](#). This requires that you provide an [IDisposable.Dispose](#) implementation to enable the deterministic release of unmanaged resources. A consumer of your type calls [Dispose](#) when the object (and the resources it uses) are no longer needed. The [Dispose](#) method immediately releases the unmanaged resources.
- In the event that a consumer of your type forgets to call [Dispose](#), provide a way for your unmanaged resources to be released. There are two ways to do this:
 - Use a safe handle to wrap your unmanaged resource. This is the recommended technique. Safe handles are derived from the [System.Runtime.InteropServices.SafeHandle](#) abstract class and include a robust [Finalize](#) method. When you use a safe handle, you simply implement the [IDisposable](#) interface and call your safe handle's [Dispose](#) method in your [IDisposable.Dispose](#) implementation. The safe handle's finalizer is called automatically by the garbage collector if its [Dispose](#) method is not called.

—or—

- Define a [finalizer](#). Finalization enables the non-deterministic release of unmanaged resources when the consumer of a type fails to call [IDisposable.Dispose](#) to dispose of them deterministically.

WARNING

Object finalization can be a complex and error-prone operation, we recommend that you use a safe handle instead of providing your own finalizer.

Consumers of your type can then call your [IDisposable.Dispose](#) implementation directly to free memory used by unmanaged resources. When you properly implement a [Dispose](#) method, either your safe handle's [Finalize](#) method or your own override of the [Object.Finalize](#) method becomes a safeguard to clean up resources in the event that the [Dispose](#) method is not called.

In this section

[Implementing a Dispose method](#) describes how to implement the dispose pattern for releasing unmanaged resources.

[Using objects that implement IDisposable](#) describes how consumers of a type ensure that its [Dispose](#) implementation is called. We strongly recommend using the C# `using` (or the Visual Basic `Using`) statement to do this.

Reference

Type / Member	Description
System.IDisposable	Defines the Dispose method for releasing unmanaged resources.
Object.Finalize	Provides for object finalization if unmanaged resources are not released by the Dispose method.
GC.SuppressFinalize	Suppresses finalization. This method is customarily called from a Dispose method to prevent a finalizer from executing.

Implement a Dispose method

9/20/2022 • 12 minutes to read • [Edit Online](#)

Implementing the `Dispose` method is primarily for releasing unmanaged resources. When working with instance members that are `IDisposable` implementations, it's common to cascade `Dispose` calls. There are additional reasons for implementing `Dispose`, for example, to free memory that was allocated, remove an item that was added to a collection, or signal the release of a lock that was acquired.

The .NET garbage collector does not allocate or release unmanaged memory. The pattern for disposing an object, referred to as the dispose pattern, imposes order on the lifetime of an object. The dispose pattern is used for objects that implement the `IDisposable` interface, and is common when interacting with file and pipe handles, registry handles, wait handles, or pointers to blocks of unmanaged memory. This is because the garbage collector is unable to reclaim unmanaged objects.

To help ensure that resources are always cleaned up appropriately, a `Dispose` method should be idempotent, such that it is callable multiple times without throwing an exception. Furthermore, subsequent invocations of `Dispose` should do nothing.

The code example provided for the `GC.KeepAlive` method shows how garbage collection can cause a finalizer to run, while an unmanaged reference to the object or its members is still in use. It may make sense to utilize `GC.KeepAlive` to make the object ineligible for garbage collection from the start of the current routine to the point where this method is called.

TIP

With regard to dependency injection, when registering services in an `IServiceCollection`, the service lifetime is managed implicitly on your behalf. The `IServiceProvider` and corresponding `IHost` orchestrate resource cleanup. Specifically, implementations of `IDisposable` and `IAsyncDisposable` are properly disposed at the end of their specified lifetime.

For more information, see [Dependency injection in .NET](#).

Safe handles

Writing code for an object's finalizer is a complex task that can cause problems if not done correctly. Therefore, we recommend that you construct `System.Runtime.InteropServices.SafeHandle` objects instead of implementing a finalizer.

A `System.Runtime.InteropServices.SafeHandle` is an abstract managed type that wraps an `System.IntPtr` that identifies an unmanaged resource. On Windows it might identify a handle while on Unix, a file descriptor. It provides all of the logic necessary to ensure that this resource is released once and only once, when the `SafeHandle` is disposed of or when all references to the `SafeHandle` have been dropped and the `SafeHandle` instance is finalized.

The `System.Runtime.InteropServices.SafeHandle` is an abstract base class. Derived classes provide specific instances for different kinds of handle. These derived classes validate what values for the `System.IntPtr` are considered invalid and how to actually free the handle. For example, `SafeFileHandle` derives from `SafeHandle` to wrap `IntPtrs` that identify open file handles/descriptors, and overrides its `SafeHandle.ReleaseHandle()` method to close it (via the `close` function on Unix or `CloseHandle` function on Windows). Most APIs in .NET libraries that create an unmanaged resource will wrap it in a `SafeHandle` and return that `SafeHandle` to you as needed, rather than handing back the raw pointer. In situations where you interact with an unmanaged component and get an `IntPtr` for an unmanaged resource, you can create your own `SafeHandle` type to wrap it. As a result, few

non-`SafeHandle` types need to implement finalizers; most disposable pattern implementations only end up wrapping other managed resources, some of which may be `SafeHandle`s.

The following derived classes in the `Microsoft.Win32.SafeHandles` namespace provide safe handles:

- The `SafeFileHandle`, `SafeMemoryMappedFileHandle`, and `SafePipeHandle` class, for files, memory mapped files, and pipes.
- The `SafeMemoryMappedViewHandle` class, for memory views.
- The `SafeNCryptKeyHandle`, `SafeNCryptProviderHandle`, and `SafeNCryptSecretHandle` classes, for cryptography constructs.
- The `SafeRegistryHandle` class, for registry keys.
- The `SafeWaitHandle` class, for wait handles.

Dispose() and Dispose(bool)

The `IDisposable` interface requires the implementation of a single parameterless method, `Dispose`. Also, any non-sealed class should have an additional `Dispose(bool)` overload method.

Method signatures are:

- `public non-virtual (NotOverridable in Visual Basic) (IDisposable.Dispose implementation).`
- `protected virtual (Overridable in Visual Basic) Dispose(bool).`

The `Dispose()` method

Because the `public`, non-virtual (`NotOverridable` in Visual Basic), parameterless `Dispose` method is called when it is no longer needed (by a consumer of the type), its purpose is to free unmanaged resources, perform general cleanup, and to indicate that the finalizer, if one is present, doesn't have to run. Freeing the actual memory associated with a managed object is always the domain of the `garbage collector`. Because of this, it has a standard implementation:

```
public void Dispose()
{
    // Dispose of unmanaged resources.
    Dispose(true);
    // Suppress finalization.
    GC.SuppressFinalize(this);
}
```

```
Public Sub Dispose() _
    Implements IDisposable.Dispose
    ' Dispose of unmanaged resources.
    Dispose(True)
    ' Suppress finalization.
    GC.SuppressFinalize(Me)
End Sub
```

The `Dispose` method performs all object cleanup, so the garbage collector no longer needs to call the objects' `Object.Finalize` override. Therefore, the call to the `SuppressFinalize` method prevents the garbage collector from running the finalizer. If the type has no finalizer, the call to `GC.SuppressFinalize` has no effect. Note that the actual cleanup is performed by the `Dispose(bool)` method overload.

The `Dispose(bool)` method overload

In the overload, the `disposing` parameter is a `Boolean` that indicates whether the method call comes from a `Dispose` method (its value is `true`) or from a finalizer (its value is `false`).

```

protected virtual void Dispose(bool disposing)
{
    if (_disposed)
    {
        return;
    }

    if (disposing)
    {
        // TODO: dispose managed state (managed objects).
    }

    // TODO: free unmanaged resources (unmanaged objects) and override a finalizer below.
    // TODO: set large fields to null.

    _disposed = true;
}

```

```

Protected Overridable Sub Dispose(disposing As Boolean)
    If disposed Then Exit Sub

    ' A block that frees unmanaged resources.

    If disposing Then
        ' Deterministic call...
        ' A conditional block that frees managed resources.
    End If

    disposed = True
End Sub

```

IMPORTANT

The `disposing` parameter should be `false` when called from a finalizer, and `true` when called from the `IDisposable.Dispose` method. In other words, it is `true` when deterministically called and `false` when non-deterministically called.

The body of the method consists of three blocks of code:

- A block for conditional return if object is already disposed.
- A block that frees unmanaged resources. This block executes regardless of the value of the `disposing` parameter.
- A conditional block that frees managed resources. This block executes if the value of `disposing` is `true`.

The managed resources that it frees can include:

- **Managed objects that implement `IDisposable`.** The conditional block can be used to call their `Dispose` implementation (cascade dispose). If you have used a derived class of `System.Runtime.InteropServices.SafeHandle` to wrap your unmanaged resource, you should call the `SafeHandle.Dispose()` implementation here.
- **Managed objects that consume large amounts of memory or consume scarce resources.** Assign large managed object references to `null` to make them more likely to be unreachable. This releases them faster than if they were reclaimed non-deterministically.

If the method call comes from a finalizer, only the code that frees unmanaged resources should execute. The implementer is responsible for ensuring that the false path doesn't interact with managed objects that may have been disposed. This is important because the order in which the garbage collector disposes managed objects

during finalization is non-deterministic.

Cascade dispose calls

If your class owns a field or property, and its type implements [IDisposable](#), the containing class itself should also implement [IDisposable](#). A class that instantiates an [IDisposable](#) implementation and storing it as an instance member, is also responsible for its cleanup. This is to help ensure that the referenced disposable types are given the opportunity to deterministically perform cleanup through the [Dispose](#) method. In this example, the class is `sealed` (or `NotInheritable` in Visual Basic).

```
using System;

public sealed class Foo : IDisposable
{
    private readonly IDisposable _bar;

    public Foo()
    {
        _bar = new Bar();
    }

    public void Dispose() => _bar.Dispose();
}
```

```
Public NotInheritable Class Foo
    Implements IDisposable

    Private ReadOnly _bar As IDisposable

    Public Sub New()
        _bar = New Bar()
    End Sub

    Public Sub Dispose() Implements IDisposable.Dispose
        _bar.Dispose()
    End Sub
End Class
```

TIP

There are cases when you may want to perform `null`-checking in a finalizer (which includes the `Dispose(false)` method invoked by a finalizer), one of the primary reasons is if you're unsure whether the instance got fully initialized (for example, an exception may be thrown in a constructor).

Implement the dispose pattern

All non-sealed classes (or Visual Basic classes not modified as `NotInheritable`) should be considered a potential base class, because they could be inherited. If you implement the dispose pattern for any potential base class, you must provide the following:

- A [Dispose](#) implementation that calls the `Dispose(bool)` method.
- A `Dispose(bool)` method that performs the actual cleanup.
- Either a class derived from [SafeHandle](#) that wraps your unmanaged resource (recommended), or an override to the [Object.Finalize](#) method. The [SafeHandle](#) class provides a finalizer, so you do not have to write one yourself.

IMPORTANT

It is possible for a base class to only reference managed objects, and implement the dispose pattern. In these cases, a finalizer is unnecessary. A finalizer is only required if you directly reference unmanaged resources.

Here's an example of the general pattern for implementing the dispose pattern for a base class that uses a safe handle.

```
using Microsoft.Win32.SafeHandles;
using System;
using System.Runtime.InteropServices;

class BaseClassWithSafeHandle : IDisposable
{
    // To detect redundant calls
    private bool _disposedValue;

    // Instantiate a SafeHandle instance.
    private SafeHandle _safeHandle = new SafeFileHandle(IntPtr.Zero, true);

    // Public implementation of Dispose pattern callable by consumers.
    public void Dispose() => Dispose(true);

    // Protected implementation of Dispose pattern.
    protected virtual void Dispose(bool disposing)
    {
        if (!_disposedValue)
        {
            if (disposing)
            {
                _safeHandle.Dispose();
            }

            _disposedValue = true;
        }
    }
}
```

```

Imports Microsoft.Win32.SafeHandles
Imports System.Runtime.InteropServices

Class BaseClassWithSafeHandle : Implements IDisposable
    ' Flag: Has Dispose already been called?
    Dim disposed As Boolean = False
    ' Instantiate a SafeHandle instance.
    Dim handle As SafeHandle = New SafeFileHandle(IntPtr.Zero, True)

    ' Public implementation of Dispose pattern callable by consumers.
    Public Sub Dispose() _
        Implements IDisposable.Dispose
        Dispose(True)
        GC.SuppressFinalize(Me)
    End Sub

    ' Protected implementation of Dispose pattern.
    Protected Overridable Sub Dispose(disposing As Boolean)
        If disposed Then Return

        If disposing Then
            handle.Dispose()
        End If

        disposed = True
    End Sub
End Class

```

NOTE

The previous example uses a [SafeFileHandle](#) object to illustrate the pattern; any object derived from [SafeHandle](#) could be used instead. Note that the example does not properly instantiate its [SafeFileHandle](#) object.

Here's the general pattern for implementing the dispose pattern for a base class that overrides [Object.Finalize](#).

```

using System;

class BaseClassWithFinalizer : IDisposable
{
    // To detect redundant calls
    private bool _disposedValue;

    ~BaseClassWithFinalizer() => Dispose(false);

    // Public implementation of Dispose pattern callable by consumers.
    public void Dispose()
    {
        Dispose(true);
        GC.SuppressFinalize(this);
    }

    // Protected implementation of Dispose pattern.
    protected virtual void Dispose(bool disposing)
    {
        if (!_disposedValue)
        {
            if (disposing)
            {
                // TODO: dispose managed state (managed objects)
            }

            // TODO: free unmanaged resources (unmanaged objects) and override finalizer
            // TODO: set large fields to null
            _disposedValue = true;
        }
    }
}

```

```

Class BaseClassWithFinalizer : Implements IDisposable
    ' Flag: Has Dispose already been called?
    Dim disposed As Boolean = False

    ' Public implementation of Dispose pattern callable by consumers.
    Public Sub Dispose() _
        Implements IDisposable.Dispose
        Dispose(True)
        GC.SuppressFinalize(Me)
    End Sub

    ' Protected implementation of Dispose pattern.
    Protected Overridable Sub Dispose(disposing As Boolean)
        If disposed Then Return

        If disposing Then
            ' Dispose managed objects that implement IDisposable.
            ' Assign null to managed objects that consume large amounts of memory or consume scarce
            resources.
            End If

            ' Free any unmanaged objects here.
            '

            disposed = True
    End Sub

    Protected Overrides Sub Finalize()
        Dispose(False)
    End Sub
End Class

```

TIP

In C#, you implement a finalization by providing a [finalizer](#), not by overriding [Object.Finalize](#). In Visual Basic, you create a finalizer with `Protected Overrides Sub Finalize()`.

Implement the dispose pattern for a derived class

A class derived from a class that implements the [IDisposable](#) interface shouldn't implement [IDisposable](#), because the base class implementation of [IDisposable.Dispose](#) is inherited by its derived classes. Instead, to clean up a derived class, you provide the following:

- A `protected override void Dispose(bool)` method that overrides the base class method and performs the actual cleanup of the derived class. This method must also call the `base.Dispose(bool)` (`MyBase.Dispose(bool)` in Visual Basic) method passing it the disposing status (`bool disposing` parameter) as an argument.
- Either a class derived from [SafeHandle](#) that wraps your unmanaged resource (recommended), or an override to the [Object.Finalize](#) method. The [SafeHandle](#) class provides a finalizer that frees you from having to code one. If you do provide a finalizer, it must call the `Dispose(bool)` overload with `false` argument.

Here's an example of the general pattern for implementing the dispose pattern for a derived class that uses a safe handle:

```
using Microsoft.Win32.SafeHandles;
using System;
using System.Runtime.InteropServices;

class DerivedClassWithSafeHandle : BaseClassWithSafeHandle
{
    // To detect redundant calls
    private bool _disposedValue;

    // Instantiate a SafeHandle instance.
    private SafeHandle _safeHandle = new SafeFileHandle(IntPtr.Zero, true);

    // Protected implementation of Dispose pattern.
    protected override void Dispose(bool disposing)
    {
        if (!_disposedValue)
        {
            if (disposing)
            {
                _safeHandle.Dispose();
            }

            _disposedValue = true;
        }
    }

    // Call base class implementation.
    base.Dispose(disposing);
}
```

```

Imports Microsoft.Win32.SafeHandles
Imports System.Runtime.InteropServices

Class DerivedClassWithSafeHandle : Inherits BaseClassWithSafeHandle
    ' Flag: Has Dispose already been called?
    Dim disposed As Boolean = False
    ' Instantiate a SafeHandle instance.
    Dim handle As SafeHandle = New SafeFileHandle(IntPtr.Zero, True)

    ' Protected implementation of Dispose pattern.
    Protected Overrides Sub Dispose(disposing As Boolean)
        If disposed Then Return

        If disposing Then
            handle.Dispose()
            ' Free any other managed objects here.
            '
        End If

        ' Free any unmanaged objects here.
        '

        disposed = True

        ' Call base class implementation.
        MyBase.Dispose(disposing)
    End Sub
End Class

```

NOTE

The previous example uses a [SafeFileHandle](#) object to illustrate the pattern; any object derived from [SafeHandle](#) could be used instead. Note that the example does not properly instantiate its [SafeFileHandle](#) object.

Here's the general pattern for implementing the dispose pattern for a derived class that overrides [Object.Finalize](#):

```

class DerivedClassWithFinalizer : BaseClassWithFinalizer
{
    // To detect redundant calls
    private bool _disposedValue;

    ~DerivedClassWithFinalizer() => this.Dispose(false);

    // Protected implementation of Dispose pattern.
    protected override void Dispose(bool disposing)
    {
        if (!_disposedValue)
        {
            if (disposing)
            {
                // TODO: dispose managed state (managed objects).
            }

            // TODO: free unmanaged resources (unmanaged objects) and override a finalizer below.
            // TODO: set large fields to null.
            _disposedValue = true;
        }
    }

    // Call the base class implementation.
    base.Dispose(disposing);
}

```

```

Class DerivedClassWithFinalizer : Inherits BaseClassWithFinalizer
    ' Flag: Has Dispose already been called?
    Dim disposed As Boolean = False

    ' Protected implementation of Dispose pattern.
    Protected Overrides Sub Dispose(disposing As Boolean)
        If disposed Then Return

        If disposing Then
            ' Dispose managed objects that implement IDisposable.
            ' Assign null to managed objects that consume large amounts of memory or consume scarce
resources.
        End If

        ' Free any unmanaged objects here.
        '

        disposed = True

        ' Call the base class implementation.
        MyBase.Dispose(disposing)
    End Sub

    Protected Overrides Sub Finalize()
        Dispose(False)
    End Sub
End Class

```

See also

- [Disposal of services](#)
- [SuppressFinalize](#)
- [IDisposable](#)
- [IDisposable.Dispose](#)
- [Microsoft.Win32.SafeHandles](#)
- [System.Runtime.InteropServices.SafeHandle](#)
- [Object.Finalize](#)
- [Define and consume classes and structs \(C++/CLI\)](#)

Implement a DisposeAsync method

9/20/2022 • 8 minutes to read • [Edit Online](#)

The [System.IAsyncDisposable](#) interface was introduced as part of C# 8.0. You implement the [IAsyncDisposable.DisposeAsync\(\)](#) method when you need to perform resource cleanup, just as you would when [implementing a Dispose method](#). One of the key differences, however, is that this implementation allows for asynchronous cleanup operations. The [DisposeAsync\(\)](#) returns a [ValueTask](#) that represents the asynchronous disposal operation.

It is typical when implementing the [IAsyncDisposable](#) interface that classes will also implement the [IDisposable](#) interface. A good implementation pattern of the [IAsyncDisposable](#) interface is to be prepared for either synchronous or asynchronous disposal. If no synchronous disposable of your class is possible, having only [IAsyncDisposable](#) is acceptable. All of the guidance for implementing the dispose pattern also applies to the asynchronous implementation. This article assumes that you're already familiar with how to [implement a Dispose method](#).

Caution

If you implement the [IAsyncDisposable](#) interface but not the [IDisposable](#) interface, your app can potentially leak resources. If a class implements [IAsyncDisposable](#), but not [IDisposable](#), and a consumer only calls `Dispose`, your implementation would never call `DisposeAsync`. This would result in a resource leak.

TIP

With regard to dependency injection, when registering services in an [IServiceCollection](#), the [service lifetime](#) is managed implicitly on your behalf. The [IServiceProvider](#) and corresponding [IHost](#) orchestrate resource cleanup. Specifically, implementations of [IDisposable](#) and [IAsyncDisposable](#) are properly disposed at the end of their specified lifetime.

For more information, see [Dependency injection in .NET](#).

DisposeAsync() and DisposeAsyncCore()

The [IAsyncDisposable](#) interface declares a single parameterless method, [DisposeAsync\(\)](#). Any non-sealed class should have an additional `DisposeAsyncCore()` method that also returns a [ValueTask](#).

- A `public IAsyncDisposable.DisposeAsync()` implementation that has no parameters.
- A `protected virtual ValueTask DisposeAsyncCore()` method whose signature is:

```
protected virtual ValueTask DisposeAsyncCore()
{
}
```

The `DisposeAsync()` method

The `public` parameterless `DisposeAsync()` method is called implicitly in an `await using` statement, and its purpose is to free unmanaged resources, perform general cleanup, and to indicate that the finalizer, if one is present, need not run. Freeing the memory associated with a managed object is always the domain of the [garbage collector](#). Because of this, it has a standard implementation:

```

public async ValueTask DisposeAsync()
{
    // Perform async cleanup.
    await DisposeAsyncCore();

    // Dispose of unmanaged resources.
    Dispose(false);

    #pragma warning disable CA1816 // Dispose methods should call SuppressFinalize
        // Suppress finalization.
        GC.SuppressFinalize(this);
    #pragma warning restore CA1816 // Dispose methods should call SuppressFinalize
}

```

NOTE

One primary difference in the async dispose pattern compared to the dispose pattern, is that the call from `DisposeAsync()` to the `Dispose(bool)` overload method is given `false` as an argument. When implementing the `IDisposable.Dispose()` method, however, `true` is passed instead. This helps ensure functional equivalence with the synchronous dispose pattern, and further ensures that finalizer code paths still get invoked. In other words, the `DisposeAsyncCore()` method will dispose of managed resources asynchronously, so you don't want to dispose of them synchronously as well. Therefore, call `Dispose(false)` instead of `Dispose(true)`.

The `DisposeAsyncCore()` method

The `DisposeAsyncCore()` method is intended to perform the asynchronous cleanup of managed resources or for cascading calls to `DisposeAsync()`. It encapsulates the common asynchronous cleanup operations when a subclass inherits a base class that is an implementation of `IAsyncDisposable`. The `DisposeAsyncCore()` method is `virtual` so that derived classes can define additional cleanup in their overrides.

TIP

If an implementation of `IAsyncDisposable` is `sealed`, the `DisposeAsyncCore()` method is not needed, and the asynchronous cleanup can be performed directly in the `IAsyncDisposable.DisposeAsync()` method.

Implement the async dispose pattern

All non-sealed classes should be considered a potential base class, because they could be inherited. If you implement the async dispose pattern for any potential base class, you must provide the `protected virtual ValueTask DisposeAsyncCore()` method. Here is an example implementation of the async dispose pattern that uses a `System.Text.Json.Utf8JsonWriter`.

```

using System;
using System.IO;
using System.Text.Json;
using System.Threading.Tasks;

public class ExampleAsyncDisposable : IAsyncDisposable, IDisposable
{
    private Utf8JsonWriter? _jsonWriter = new(new MemoryStream());

    public void Dispose()
    {
        Dispose(disposing: true);
        GC.SuppressFinalize(this);
    }

    public async ValueTask DisposeAsync()
    {
        await DisposeAsyncCore().ConfigureAwait(false);

        Dispose(disposing: false);
#pragma warning disable CA1816 // Dispose methods should call SuppressFinalize
        GC.SuppressFinalize(this);
#pragma warning restore CA1816 // Dispose methods should call SuppressFinalize
    }

    protected virtual void Dispose(bool disposing)
    {
        if (disposing)
        {
            _jsonWriter?.Dispose();
            _jsonWriter = null;
        }
    }

    protected virtual async ValueTask DisposeAsyncCore()
    {
        if (_jsonWriter is not null)
        {
            await _jsonWriter.DisposeAsync().ConfigureAwait(false);
        }

        _jsonWriter = null;
    }
}

```

The preceding example uses the [Utf8JsonWriter](#). For more information about [System.Text.Json](#), see [How to migrate from Newtonsoft.Json to System.Text.Json](#).

Implement both dispose and async dispose patterns

You may need to implement both the [IDisposable](#) and [IAsyncDisposable](#) interfaces, especially when your class scope contains instances of these implementations. Doing so ensures that you can properly cascade clean up calls. Here is an example class that implements both interfaces and demonstrates the proper guidance for cleanup.

```

using System;
using System.IO;
using System.Threading.Tasks;

class ExampleConjunctiveDisposable : IDisposable, IAsyncDisposable
{
    IDisposable? _disposableResource = new MemoryStream();
    IAsyncDisposable? _asyncDisposableResource = new MemoryStream();

    public void Dispose()
    {
        Dispose(disposing: true);
        GC.SuppressFinalize(this);
    }

    public async ValueTask DisposeAsync()
    {
        await DisposeAsyncCore().ConfigureAwait(false);

        Dispose(disposing: false);
#pragma warning disable CA1816 // Dispose methods should call SuppressFinalize
        GC.SuppressFinalize(this);
#pragma warning restore CA1816 // Dispose methods should call SuppressFinalize
    }

    protected virtual void Dispose(bool disposing)
    {
        if (disposing)
        {
            _disposableResource?.Dispose();
            (_asyncDisposableResource as IDisposable)??.Dispose();
            _disposableResource = null;
            _asyncDisposableResource = null;
        }
    }

    protected virtual async ValueTask DisposeAsyncCore()
    {
        if (_asyncDisposableResource is not null)
        {
            await _asyncDisposableResource.DisposeAsync().ConfigureAwait(false);
        }

        if (_disposableResource is IAsyncDisposable disposable)
        {
            await disposable.DisposeAsync().ConfigureAwait(false);
        }
        else
        {
            _disposableResource?.Dispose();
        }

        _asyncDisposableResource = null;
        _disposableResource = null;
    }
}

```

The [IDisposable.Dispose\(\)](#) and [IAsyncDisposable.DisposeAsync\(\)](#) implementations are both simple boilerplate code.

In the `Dispose(bool)` overload method, the [IDisposable](#) instance is conditionally disposed of if it is not `null`. The [IAsyncDisposable](#) instance is cast as [IDisposable](#), and if it is also not `null`, it is disposed of as well. Both instances are then assigned to `null`.

With the `DisposeAsyncCore()` method, the same logical approach is followed. If the [IAsyncDisposable](#) instance is

not `null`, its call to `DisposeAsync().ConfigureAwait(false)` is awaited. If the `IDisposable` instance is also an implementation of `IAsyncDisposable`, it's also disposed of asynchronously. Both instances are then assigned to `null`.

Using async disposable

To properly consume an object that implements the `IAsyncDisposable` interface, you use the `await` and `using` keywords together. Consider the following example, where the `ExampleAsyncDisposable` class is instantiated and then wrapped in an `await using` statement.

```
using System;
using System.Threading.Tasks;

class ExampleConfigureAwaitProgram
{
    static async Task Main()
    {
        var exampleAsyncDisposable = new ExampleAsyncDisposable();
        await using (exampleAsyncDisposable.ConfigureAwait(false))
        {
            // Interact with the exampleAsyncDisposable instance.
        }

        Console.ReadLine();
    }
}
```

IMPORTANT

Use the `ConfigureAwait(IAsyncDisposable, Boolean)` extension method of the `IAsyncDisposable` interface to configure how the continuation of the task is marshalled on its original context or scheduler. For more information on `ConfigureAwait`, see [ConfigureAwait FAQ](#).

For situations where the usage of `ConfigureAwait` is not needed, the `await using` statement could be simplified as follows:

```
using System;
using System.Threading.Tasks;

class ExampleUsingStatementProgram
{
    static async Task Main()
    {
        await using (var exampleAsyncDisposable = new ExampleAsyncDisposable())
        {
            // Interact with the exampleAsyncDisposable instance.
        }

        Console.ReadLine();
    }
}
```

Furthermore, it could be written to use the implicit scoping of a [using declaration](#).

```
using System;
using System.Threading.Tasks;

class ExampleUsingDeclarationProgram
{
    static async Task Main()
    {
        await using var exampleAsyncDisposable = new ExampleAsyncDisposable();

        // Interact with the exampleAsyncDisposable instance.

        Console.ReadLine();
    }
}
```

Multiple await keywords in a single line

Sometimes the `await` keyword may appear multiple times within a single line. For example, consider the following code:

```
await using var transaction = await context.Database.BeginTransactionAsync(token);
```

In the preceding example:

- The `BeginTransactionAsync` method is awaited.
- The return type is `DbTransaction`, which implements `IAsyncDisposable`.
- The `transaction` is used asynchronously, and also awaited.

Stacked usings

In situations where you create and use multiple objects that implement `IAsyncDisposable`, it's possible that stacking `await using` statements with `ConfigureAwait` could prevent calls to `DisposeAsync()` in errant conditions. To ensure that `DisposeAsync()` is always called, you should avoid stacking. The following three code examples show acceptable patterns to use instead.

Acceptable pattern one

```

using System;
using System.Threading.Tasks;

class ExampleOneProgram
{
    static async Task Main()
    {
        var objOne = new ExampleAsyncDisposable();
        await using (objOne.ConfigureAwait(false))
        {
            // Interact with the objOne instance.

            var objTwo = new ExampleAsyncDisposable();
            await using (objTwo.ConfigureAwait(false))
            {
                // Interact with the objOne and/or objTwo instance(s).
            }
        }

        Console.ReadLine();
    }
}

```

In the preceding example, each asynchronous clean up operation is explicitly scoped under the `await using` block. The outer scope is defined by how `objOne` sets its braces, enclosing `objTwo`, as such `objTwo` is disposed first, followed by `objOne`. Both `IAsyncDisposable` instances have their `DisposeAsync()` method awaited, so each instance performs its asynchronous clean up operation. The calls are nested, not stacked.

Acceptable pattern two

```

class ExampleTwoProgram
{
    static async Task Main()
    {
        var objOne = new ExampleAsyncDisposable();
        await using (objOne.ConfigureAwait(false))
        {
            // Interact with the objOne instance.
        }

        var objTwo = new ExampleAsyncDisposable();
        await using (objTwo.ConfigureAwait(false))
        {
            // Interact with the objTwo instance.
        }

        Console.ReadLine();
    }
}

```

In the preceding example, each asynchronous clean up operation is explicitly scoped under the `await using` block. At the end of each block, the corresponding `IAsyncDisposable` instance has its `DisposeAsync()` method awaited, thus performing its asynchronous clean up operation. The calls are sequential, not stacked. In this scenario `objOne` is disposed first, then `objTwo` is disposed.

Acceptable pattern three

```

class ExampleThreeProgram
{
    static async Task Main()
    {
        var objOne = new ExampleAsyncDisposable();
        await using var ignored1 = objOne.ConfigureAwait(false);

        var objTwo = new ExampleAsyncDisposable();
        await using var ignored2 = objTwo.ConfigureAwait(false);

        // Interact with objOne and/or objTwo instance(s).

        Console.ReadLine();
    }
}

```

In the preceding example, each asynchronous clean up operation is implicitly scoped with the containing method body. At the end of the enclosing block, the `IAsyncDisposable` instances perform their asynchronous clean up operations. This runs in reverse order from which they were declared, meaning that `objTwo` is disposed before `objOne`.

Unacceptable pattern

The highlighted lines in the following code show what it means to have "stacked usings". If an exception is thrown from the `AnotherAsyncDisposable` constructor, neither object is properly disposed of. The variable `objTwo` is never assigned because the constructor did not complete successfully. As a result, the constructor for `AnotherAsyncDisposable` is responsible for disposing any resources allocated before it throws an exception. If the `ExampleAsyncDisposable` type has a finalizer, it's eligible for finalization.

```

class DoNotDoThisProgram
{
    static async Task Main()
    {
        var objOne = new ExampleAsyncDisposable();
        // Exception thrown on .ctor
        var objTwo = new AnotherAsyncDisposable();

        await using (objOne.ConfigureAwait(false))
        await using (objTwo.ConfigureAwait(false))
        {
            // Neither object has its DisposeAsync called.
        }

        Console.ReadLine();
    }
}

```

TIP

Avoid this pattern as it could lead to unexpected behavior. If you use one of the acceptable patterns, the problem of undisposed objects is non-existent. The clean-up operations are correctly performed when `using` statements aren't stacked.

See also

For a dual implementation example of `IDisposable` and `IAsyncDisposable`, see the [Utf8JsonWriter](#) source code on [GitHub](#).

- [Disposal of services](#)

- [IAsyncDisposable](#)
- [IAsyncDisposable.DisposeAsync\(\)](#)
- [ConfigureAwait\(IAsyncDisposable, Boolean\)](#)

Using objects that implement IDisposable

9/20/2022 • 5 minutes to read • [Edit Online](#)

The common language runtime's garbage collector (GC) reclaims the memory used by managed objects.

Typically, types that use unmanaged resources implement the [IDisposable](#) or [IAsyncDisposable](#) interface to allow the unmanaged resources to be reclaimed. When you finish using an object that implements [IDisposable](#), you call the object's [Dispose](#) or [DisposeAsync](#) implementation to explicitly perform cleanup. You can do this in one of two ways:

- With the C# `using` statement or declaration (`Using` in Visual Basic).
- By implementing a `try/finally` block, and calling the [Dispose](#) or [DisposeAsync](#) method in the `finally`.

IMPORTANT

The GC does *not* dispose your objects, as it has no knowledge of [IDisposable.Dispose\(\)](#) or [IAsyncDisposable.DisposeAsync\(\)](#). The GC only knows whether an object is finalizable (that is, it defines an [Object.Finalize\(\)](#) method), and when the object's finalizer needs to be called. For more information, see [How finalization works](#). For additional details on implementing [Dispose](#) and [DisposeAsync](#), see:

- [Implement a Dispose method](#)
- [Implement a DisposeAsync method](#)

Objects that implement [System.IDisposable](#) or [System.IAsyncDisposable](#) should always be properly disposed of, regardless of variable scoping, unless otherwise explicitly stated. Types that define a finalizer to release unmanaged resources usually call [GC.SuppressFinalize](#) from either their [Dispose](#) or [DisposeAsync](#) implementation. Calling [SuppressFinalize](#) indicates to the GC that the finalizer has already been run and the object shouldn't be promoted for finalization.

The using statement

The `using` statement in C# and the `Using` statement in Visual Basic simplify the code that you must write to cleanup an object. The `using` statement obtains one or more resources, executes the statements that you specify, and automatically disposes of the object. However, the `using` statement is useful only for objects that are used within the scope of the method in which they are constructed.

The following example uses the `using` statement to create and release a [System.IO.StreamReader](#) object.

```

using System.IO;

class UsingStatement
{
    static void Main()
    {
        var buffer = new char[50];
        using (StreamReader streamReader = new("file1.txt"))
        {
            int charsRead = 0;
            while (streamReader.Peek() != -1)
            {
                charsRead = streamReader.Read(buffer, 0, buffer.Length);
                //
                // Process characters read.
                //
            }
        }
    }
}

```

```

Imports System.IO

Module UsingStatement
    Public Sub Main()
        Dim buffer(49) As Char
        Using streamReader As New StreamReader("File1.txt")
            Dim charsRead As Integer
            Do While streamReader.Peek() <> -1
                charsRead = streamReader.Read(buffer, 0, buffer.Length)
                '
                ' Process characters read.
                '

            Loop
        End Using
    End Sub
End Module

```

With C# 8, a **using declaration** is an alternative syntax available where the braces are removed, and scoping is implicit.

```

using System.IO;

class UsingDeclaration
{
    static void Main()
    {
        var buffer = new char[50];
        using StreamReader streamReader = new("file1.txt");

        int charsRead = 0;
        while (streamReader.Peek() != -1)
        {
            charsRead = streamReader.Read(buffer, 0, buffer.Length);
            //
            // Process characters read.
            //
        }
    }
}

```

Although the **StreamReader** class implements the **IDisposable** interface, which indicates that it uses an

unmanaged resource, the example doesn't explicitly call the `StreamReader.Dispose` method. When the C# or Visual Basic compiler encounters the `using` statement, it emits intermediate language (IL) that is equivalent to the following code that explicitly contains a `try/finally` block.

```
using System.IO;

class TryFinallyGenerated
{
    static void Main()
    {
        var buffer = new char[50];
        StreamReader? streamReader = null;
        try
        {
            streamReader = new StreamReader("file1.txt");
            int charsRead = 0;
            while (streamReader.Peek() != -1)
            {
                charsRead = streamReader.Read(buffer, 0, buffer.Length);
                //
                // Process characters read.
                //
            }
        }
        finally
        {
            // If non-null, call the object's Dispose method.
            streamReader?.Dispose();
        }
    }
}
```

```
Imports System.IO

Module TryFinallyGenerated
    Public Sub Main()
        Dim buffer(49) As Char
        Dim streamReader As New StreamReader("File1.txt")
        Try
            Dim charsRead As Integer
            Do While streamReader.Peek() <> -1
                charsRead = streamReader.Read(buffer, 0, buffer.Length)
                '
                ' Process characters read.
                '

            Loop
        Finally
            If streamReader IsNot Nothing Then DirectCast(streamReader, IDisposable).Dispose()
        End Try
    End Sub
End Module
```

The C# `using` statement also allows you to acquire multiple resources in a single statement, which is internally equivalent to nested `using` statements. The following example instantiates two `StreamReader` objects to read the contents of two different files.

```

using System.IO;

class SingleStatementMultiple
{
    static void Main()
    {
        var buffer1 = new char[50];
        var buffer2 = new char[50];

        using StreamReader version1 = new("file1.txt"),
            version2 = new("file2.txt");

        int charsRead1, charsRead2 = 0;
        while (version1.Peek() != -1 && version2.Peek() != -1)
        {
            charsRead1 = version1.Read(buffer1, 0, buffer1.Length);
            charsRead2 = version2.Read(buffer2, 0, buffer2.Length);
            //
            // Process characters read.
            //
        }
    }
}

```

Try/finally block

Instead of wrapping a `try/finally` block in a `using` statement, you may choose to implement the `try/finally` block directly. It may be your personal coding style, or you might want to do this for one of the following reasons:

- To include a `catch` block to handle exceptions thrown in the `try` block. Otherwise, any exceptions thrown within the `using` statement are unhandled.
- To instantiate an object that implements [IDisposable](#) whose scope is not local to the block within which it is declared.

The following example is similar to the previous example, except that it uses a `try/catch/finally` block to instantiate, use, and dispose of a `StreamReader` object, and to handle any exceptions thrown by the `StreamReader` constructor and its `ReadToEnd` method. The code in the `finally` block checks that the object that implements [IDisposable](#) isn't `null` before it calls the `Dispose` method. Failure to do this can result in a [NullReferenceException](#) exception at run time.

```

using System;
using System.Globalization;
using System.IO;

class TryExplicitCatchFinally
{
    static void Main()
    {
        StreamReader? streamReader = null;
        try
        {
            streamReader = new StreamReader("file1.txt");
            string contents = streamReader.ReadToEnd();
            var info = new StringInfo(contents);
            Console.WriteLine($"The file has {info.LengthInTextElements} text elements.");
        }
        catch (FileNotFoundException)
        {
            Console.WriteLine("The file cannot be found.");
        }
        catch (IOException)
        {
            Console.WriteLine("An I/O error has occurred.");
        }
        catch (OutOfMemoryException)
        {
            Console.WriteLine("There is insufficient memory to read the file.");
        }
        finally
        {
            streamReader?.Dispose();
        }
    }
}

```

```

Imports System.Globalization
Imports System.IO

Module TryExplicitCatchFinally
    Sub Main()
        Dim streamReader As StreamReader = Nothing
        Try
            streamReader = New StreamReader("file1.txt")
            Dim contents As String = streamReader.ReadToEnd()
            Dim info As StringInfo = New StringInfo(contents)
            Console.WriteLine($"The file has {info.LengthInTextElements} text elements.")
        Catch e As FileNotFoundException
            Console.WriteLine("The file cannot be found.")
        Catch e As IOException
            Console.WriteLine("An I/O error has occurred.")
        Catch e As OutOfMemoryException
            Console.WriteLine("There is insufficient memory to read the file.")
        Finally
            If streamReader IsNot Nothing Then streamReader.Dispose()
        End Try
    End Sub
End Module

```

You can follow this basic pattern if you choose to implement or must implement a `try/finally` block, because your programming language doesn't support a `using` statement but does allow direct calls to the `Dispose` method.

IDisposable instance members

If a class holds an [IDisposable](#) implementation as an instance member, either a field or a property, the class should also implement [IDisposable](#). For more information, see [implement a cascade dispose](#).

See also

- [Cleaning up unmanaged resources](#)
- [using Statement \(C# Reference\)](#)
- [Using Statement \(Visual Basic\)](#)

Garbage collection

9/20/2022 • 2 minutes to read • [Edit Online](#)

.NET's garbage collector manages the allocation and release of memory for your application. Each time you create a new object, the common language runtime allocates memory for the object from the managed heap. As long as address space is available in the managed heap, the runtime continues to allocate space for new objects. However, memory is not infinite. Eventually the garbage collector must perform a collection in order to free some memory. The garbage collector's optimizing engine determines the best time to perform a collection, based upon the allocations being made. When the garbage collector performs a collection, it checks for objects in the managed heap that are no longer being used by the application and performs the necessary operations to reclaim their memory.

In this section

TITLE	DESCRIPTION
Fundamentals of garbage collection	Describes how garbage collection works, how objects are allocated on the managed heap, and other core concepts.
Workstation and server garbage collection	Describes the differences between workstation garbage collection for client apps and server garbage collection for server apps.
Background garbage collection	Describes background garbage collection, which is the collection of generation 0 and 1 objects while generation 2 collection is in progress.
The large object heap	Describes the large object heap (LOH) and how large objects are garbage-collected.
Garbage collection and performance	Describes the performance checks you can use to diagnose garbage collection and performance issues.
Induced collections	Describes how to make a garbage collection occur.
Latency modes	Describes the modes that determine the intrusiveness of garbage collection.
Optimization for shared web hosting	Describes how to optimize garbage collection on servers shared by several small Web sites.
Garbage collection notifications	Describes how to determine when a full garbage collection is approaching and when it has completed.
Application domain resource monitoring	Describes how to monitor CPU and memory usage by an application domain.
Weak references	Describes features that permit the garbage collector to collect an object while still allowing the application to access that object.

Reference

- [System.GC](#)
- [System.GCCollectionMode](#)
- [System.GCNotificationStatus](#)
- [System.Runtime.GCLatencyMode](#)
- [System.Runtime.GCSettings](#)
- [GCSettings.LargeObjectHeapCompactionMode](#)
- [Object.Finalize](#)
- [System.IDisposable](#)

See also

- [Clean up unmanaged resources](#)

Fundamentals of garbage collection

9/20/2022 • 14 minutes to read • [Edit Online](#)

In the common language runtime (CLR), the garbage collector (GC) serves as an automatic memory manager. The garbage collector manages the allocation and release of memory for an application. For developers working with managed code, this means that you don't have to write code to perform memory management tasks. Automatic memory management can eliminate common problems, such as forgetting to free an object and causing a memory leak or attempting to access memory for an object that's already been freed.

This article describes the core concepts of garbage collection.

Benefits

The garbage collector provides the following benefits:

- Freed developers from having to manually release memory.
- Allocates objects on the managed heap efficiently.
- Reclaims objects that are no longer being used, clears their memory, and keeps the memory available for future allocations. Managed objects automatically get clean content to start with, so their constructors don't have to initialize every data field.
- Provides memory safety by making sure that an object cannot use for itself the memory allocated for another object.

Fundamentals of memory

The following list summarizes important CLR memory concepts.

- Each process has its own, separate virtual address space. All processes on the same computer share the same physical memory and the page file, if there is one.
- By default, on 32-bit computers, each process has a 2-GB user-mode virtual address space.
- As an application developer, you work only with virtual address space and never manipulate physical memory directly. The garbage collector allocates and frees virtual memory for you on the managed heap.

If you're writing native code, you use Windows functions to work with the virtual address space. These functions allocate and free virtual memory for you on native heaps.

- Virtual memory can be in three states:

STATE	DESCRIPTION
Free	The block of memory has no references to it and is available for allocation.
Reserved	The block of memory is available for your use and cannot be used for any other allocation request. However, you cannot store data to this memory block until it is committed.
Committed	The block of memory is assigned to physical storage.

- Virtual address space can get fragmented. This means that there are free blocks, also known as holes, in the address space. When a virtual memory allocation is requested, the virtual memory manager has to find a single free block that is large enough to satisfy that allocation request. Even if you have 2 GB of free space, an allocation that requires 2 GB will be unsuccessful unless all of that free space is in a single address block.
- You can run out of memory if there isn't enough virtual address space to reserve or physical space to commit.

The page file is used even if physical memory pressure (that is, demand for physical memory) is low. The first time that physical memory pressure is high, the operating system must make room in physical memory to store data, and it backs up some of the data that is in physical memory to the page file. That data is not paged until it's needed, so it's possible to encounter paging in situations where the physical memory pressure is low.

Memory allocation

When you initialize a new process, the runtime reserves a contiguous region of address space for the process. This reserved address space is called the managed heap. The managed heap maintains a pointer to the address where the next object in the heap will be allocated. Initially, this pointer is set to the managed heap's base address. All reference types are allocated on the managed heap. When an application creates the first reference type, memory is allocated for the type at the base address of the managed heap. When the application creates the next object, the garbage collector allocates memory for it in the address space immediately following the first object. As long as address space is available, the garbage collector continues to allocate space for new objects in this manner.

Allocating memory from the managed heap is faster than unmanaged memory allocation. Because the runtime allocates memory for an object by adding a value to a pointer, it's almost as fast as allocating memory from the stack. In addition, because new objects that are allocated consecutively are stored contiguously in the managed heap, an application can access the objects quickly.

Memory release

The garbage collector's optimizing engine determines the best time to perform a collection based on the allocations being made. When the garbage collector performs a collection, it releases the memory for objects that are no longer being used by the application. It determines which objects are no longer being used by examining the application's *roots*. An application's roots include static fields, local variables on a thread's stack, CPU registers, GC handles, and the finalize queue. Each root either refers to an object on the managed heap or is set to null. The garbage collector can ask the rest of the runtime for these roots. Using this list, the garbage collector creates a graph that contains all the objects that are reachable from the roots.

Objects that are not in the graph are unreachable from the application's roots. The garbage collector considers unreachable objects garbage and releases the memory allocated for them. During a collection, the garbage collector examines the managed heap, looking for the blocks of address space occupied by unreachable objects. As it discovers each unreachable object, it uses a memory-copying function to compact the reachable objects in memory, freeing up the blocks of address spaces allocated to unreachable objects. Once the memory for the reachable objects has been compacted, the garbage collector makes the necessary pointer corrections so that the application's roots point to the objects in their new locations. It also positions the managed heap's pointer after the last reachable object.

Memory is compacted only if a collection discovers a significant number of unreachable objects. If all the objects in the managed heap survive a collection, then there is no need for memory compaction.

To improve performance, the runtime allocates memory for large objects in a separate heap. The garbage collector automatically releases the memory for large objects. However, to avoid moving large objects in memory, this memory is usually not compacted.

Conditions for a garbage collection

Garbage collection occurs when one of the following conditions is true:

- The system has low physical memory. This is detected by either the low memory notification from the OS or low memory as indicated by the host.
- The memory that's used by allocated objects on the managed heap surpasses an acceptable threshold. This threshold is continuously adjusted as the process runs.
- The [GC.Collect](#) method is called. In almost all cases, you don't have to call this method, because the garbage collector runs continuously. This method is primarily used for unique situations and testing.

The managed heap

After the garbage collector is initialized by the CLR, it allocates a segment of memory to store and manage objects. This memory is called the managed heap, as opposed to a native heap in the operating system.

There is a managed heap for each managed process. All threads in the process allocate memory for objects on the same heap.

To reserve memory, the garbage collector calls the Windows [VirtualAlloc](#) function and reserves one segment of memory at a time for managed applications. The garbage collector also reserves segments, as needed, and releases segments back to the operating system (after clearing them of any objects) by calling the Windows [VirtualFree](#) function.

IMPORTANT

The size of segments allocated by the garbage collector is implementation-specific and is subject to change at any time, including in periodic updates. Your app should never make assumptions about or depend on a particular segment size, nor should it attempt to configure the amount of memory available for segment allocations.

The fewer objects allocated on the heap, the less work the garbage collector has to do. When you allocate objects, don't use rounded-up values that exceed your needs, such as allocating an array of 32 bytes when you need only 15 bytes.

When a garbage collection is triggered, the garbage collector reclaims the memory that's occupied by dead objects. The reclaiming process compacts live objects so that they are moved together, and the dead space is removed, thereby making the heap smaller. This ensures that objects that are allocated together stay together on the managed heap to preserve their locality.

The intrusiveness (frequency and duration) of garbage collections is the result of the volume of allocations and the amount of survived memory on the managed heap.

The heap can be considered as the accumulation of two heaps: the [large object heap](#) and the small object heap. The large object heap contains objects that are 85,000 bytes and larger, which are usually arrays. It's rare for an instance object to be extremely large.

TIP

You can [configure the threshold size](#) for objects to go on the large object heap.

Generations

The GC algorithm is based on several considerations:

- It's faster to compact the memory for a portion of the managed heap than for the entire managed heap.
- Newer objects have shorter lifetimes and older objects have longer lifetimes.
- Newer objects tend to be related to each other and accessed by the application around the same time.

Garbage collection primarily occurs with the reclamation of short-lived objects. To optimize the performance of the garbage collector, the managed heap is divided into three generations, 0, 1, and 2, so it can handle long-lived and short-lived objects separately. The garbage collector stores new objects in generation 0. Objects created early in the application's lifetime that survive collections are promoted and stored in generations 1 and 2. Because it's faster to compact a portion of the managed heap than the entire heap, this scheme allows the garbage collector to release the memory in a specific generation rather than release the memory for the entire managed heap each time it performs a collection.

- **Generation 0.** This is the youngest generation and contains short-lived objects. An example of a short-lived object is a temporary variable. Garbage collection occurs most frequently in this generation.

Newly allocated objects form a new generation of objects and are implicitly generation 0 collections. However, if they are large objects, they go on the large object heap (LOH), which is sometimes referred to as *generation 3*. Generation 3 is a physical generation that's logically collected as part of generation 2.

Most objects are reclaimed for garbage collection in generation 0 and don't survive to the next generation.

If an application attempts to create a new object when generation 0 is full, the garbage collector performs a collection in an attempt to free address space for the object. The garbage collector starts by examining the objects in generation 0 rather than all objects in the managed heap. A collection of generation 0 alone often reclaims enough memory to enable the application to continue creating new objects.

- **Generation 1.** This generation contains short-lived objects and serves as a buffer between short-lived objects and long-lived objects.

After the garbage collector performs a collection of generation 0, it compacts the memory for the reachable objects and promotes them to generation 1. Because objects that survive collections tend to have longer lifetimes, it makes sense to promote them to a higher generation. The garbage collector doesn't have to reexamine the objects in generations 1 and 2 each time it performs a collection of generation 0.

If a collection of generation 0 does not reclaim enough memory for the application to create a new object, the garbage collector can perform a collection of generation 1, then generation 2. Objects in generation 1 that survive collections are promoted to generation 2.

- **Generation 2.** This generation contains long-lived objects. An example of a long-lived object is an object in a server application that contains static data that's live for the duration of the process.

Objects in generation 2 that survive a collection remain in generation 2 until they are determined to be unreachable in a future collection.

Objects on the large object heap (which is sometimes referred to as *generation 3*) are also collected in generation 2.

Garbage collections occur on specific generations as conditions warrant. Collecting a generation means collecting objects in that generation and all its younger generations. A generation 2 garbage collection is also known as a full garbage collection, because it reclaims objects in all generations (that is, all objects in the managed heap).

Survival and promotions

Objects that are not reclaimed in a garbage collection are known as survivors and are promoted to the next generation:

- Objects that survive a generation 0 garbage collection are promoted to generation 1.
- Objects that survive a generation 1 garbage collection are promoted to generation 2.
- Objects that survive a generation 2 garbage collection remain in generation 2.

When the garbage collector detects that the survival rate is high in a generation, it increases the threshold of allocations for that generation. The next collection gets a substantial size of reclaimed memory. The CLR continually balances two priorities: not letting an application's working set get too large by delaying garbage collection and not letting the garbage collection run too frequently.

Ephemeral generations and segments

Because objects in generations 0 and 1 are short-lived, these generations are known as the *ephemeral generations*.

Ephemeral generations are allocated in the memory segment that's known as the ephemeral segment. Each new segment acquired by the garbage collector becomes the new ephemeral segment and contains the objects that survived a generation 0 garbage collection. The old ephemeral segment becomes the new generation 2 segment.

The size of the ephemeral segment varies depending on whether a system is 32-bit or 64-bit and on the type of garbage collector it is running ([workstation or server GC](#)). The following table shows the default sizes of the ephemeral segment.

WORKSTATION/SERVER GC	32-BIT	64-BIT
Workstation GC	16 MB	256 MB
Server GC	64 MB	4 GB
Server GC with > 4 logical CPUs	32 MB	2 GB
Server GC with > 8 logical CPUs	16 MB	1 GB

The ephemeral segment can include generation 2 objects. Generation 2 objects can use multiple segments (as many as your process requires and memory allows for).

The amount of freed memory from an ephemeral garbage collection is limited to the size of the ephemeral segment. The amount of memory that is freed is proportional to the space that was occupied by the dead objects.

What happens during a garbage collection

A garbage collection has the following phases:

- A marking phase that finds and creates a list of all live objects.
- A relocating phase that updates the references to the objects that will be compacted.
- A compacting phase that reclaims the space occupied by the dead objects and compacts the surviving objects. The compacting phase moves objects that have survived a garbage collection toward the older end of the segment.

Because generation 2 collections can occupy multiple segments, objects that are promoted into generation 2 can be moved into an older segment. Both generation 1 and generation 2 survivors can be moved to a different segment, because they are promoted to generation 2.

Ordinarily, the large object heap (LOH) is not compacted, because copying large objects imposes a

performance penalty. However, in .NET Core and in .NET Framework 4.5.1 and later, you can use the [GCSettings.LargeObjectHeapCompactionMode](#) property to compact the large object heap on demand. In addition, the LOH is automatically compacted when a hard limit is set by specifying either:

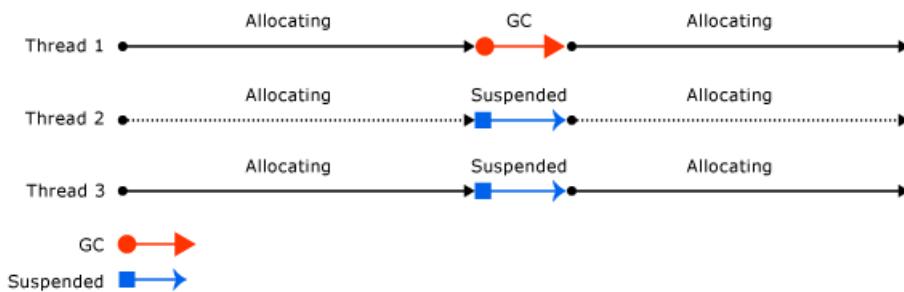
- A memory limit on a container.
- The [GCHeapHardLimit](#) or [GCHeapHardLimitPercent](#) runtime configuration options.

The garbage collector uses the following information to determine whether objects are live:

- **Stack roots.** Stack variables provided by the just-in-time (JIT) compiler and stack walker. JIT optimizations can lengthen or shorten regions of code within which stack variables are reported to the garbage collector.
- **Garbage collection handles.** Handles that point to managed objects and that can be allocated by user code or by the common language runtime.
- **Static data.** Static objects in application domains that could be referencing other objects. Each application domain keeps track of its static objects.

Before a garbage collection starts, all managed threads are suspended except for the thread that triggered the garbage collection.

The following illustration shows a thread that triggers a garbage collection and causes the other threads to be suspended.



Unmanaged resources

For most of the objects that your application creates, you can rely on garbage collection to automatically perform the necessary memory management tasks. However, unmanaged resources require explicit cleanup. The most common type of unmanaged resource is an object that wraps an operating system resource, such as a file handle, window handle, or network connection. Although the garbage collector is able to track the lifetime of a managed object that encapsulates an unmanaged resource, it doesn't have specific knowledge about how to clean up the resource.

When you define an object that encapsulates an unmanaged resource, it's recommended that you provide the necessary code to clean up the unmanaged resource in a public `Dispose` method. By providing a `Dispose` method, you enable users of your object to explicitly release the resource when they're finished with the object. When you use an object that encapsulates an unmanaged resource, make sure to call `Dispose` as necessary.

You must also provide a way for your unmanaged resources to be released in case a consumer of your type forgets to call `Dispose`. You can either use a safe handle to wrap the unmanaged resource, or override the `Object.Finalize()` method.

For more information about cleaning up unmanaged resources, see [Clean up unmanaged resources](#).

See also

- [Workstation and server garbage collection](#)

- Background garbage collection
- Configuration options for GC
- Garbage collection

Workstation and server garbage collection

9/20/2022 • 2 minutes to read • [Edit Online](#)

The garbage collector is self-tuning and can work in a wide variety of scenarios. However, you can [set the type of garbage collection](#) based on the characteristics of the workload. The CLR provides the following types of garbage collection:

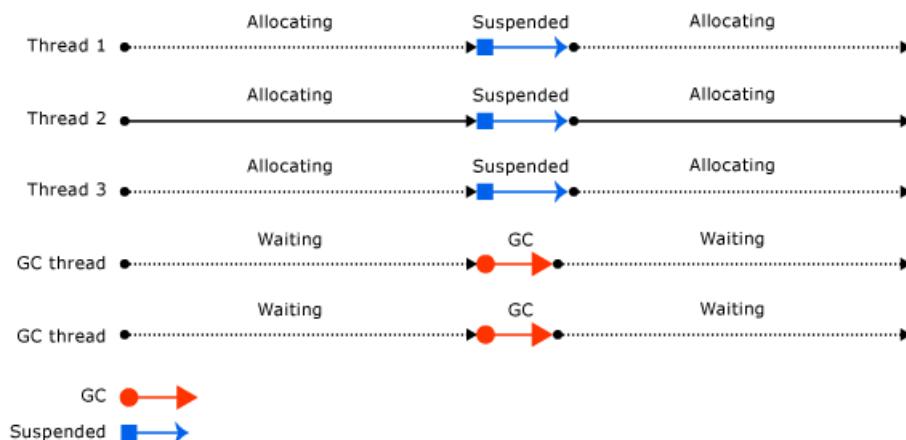
- Workstation garbage collection (GC), which is designed for client apps. It's the default GC flavor for standalone apps. For hosted apps, for example, those hosted by ASP.NET, the host determines the default GC flavor.

Workstation garbage collection can be concurrent or non-concurrent. Concurrent (or *background*) garbage collection enables managed threads to continue operations during a garbage collection.

[Background garbage collection](#) replaces [concurrent garbage collection](#) in .NET Framework 4 and later versions.

- Server garbage collection, which is intended for server applications that need high throughput and scalability.
 - In .NET Core, server garbage collection can be non-concurrent or background.
 - In .NET Framework 4.5 and later versions, server garbage collection can be non-concurrent or background. In .NET Framework 4 and previous versions, server garbage collection is non-concurrent.

The following illustration shows the dedicated threads that perform the garbage collection on a server:



Performance considerations

Workstation GC

The following are threading and performance considerations for workstation garbage collection:

- The collection occurs on the user thread that triggered the garbage collection and remains at the same priority. Because user threads typically run at normal priority, the garbage collector (which runs on a normal priority thread) must compete with other threads for CPU time. (Threads that run native code are not suspended on either server or workstation garbage collection.)
- Workstation garbage collection is always used on a computer that has only one logical CPU, regardless of the [configuration setting](#).

Server GC

The following are threading and performance considerations for server garbage collection:

- The collection occurs on multiple dedicated threads that are running at `THREAD_PRIORITY_HIGHEST` priority level.
- A heap and a dedicated thread to perform garbage collection are provided for each logical CPU, and the heaps are collected at the same time. Each heap contains a small object heap and a large object heap, and all heaps can be accessed by user code. Objects on different heaps can refer to each other.
- Because multiple garbage collection threads work together, server garbage collection is faster than workstation garbage collection on the same size heap.
- Server garbage collection often has larger size segments. However, this is only a generalization: segment size is implementation-specific and is subject to change. Don't make assumptions about the size of segments allocated by the garbage collector when tuning your app.
- Server garbage collection can be resource-intensive. For example, imagine that there are 12 processes that use server GC running on a computer that has four logical CPUs. If all the processes happen to collect garbage at the same time, they would interfere with each other, as there would be 12 threads scheduled on the same logical CPU. If the processes are active, it's not a good idea to have them all use server GC.

If you're running hundreds of instances of an application, consider using workstation garbage collection with concurrent garbage collection disabled. This will result in less context switching, which can improve performance.

See also

- [Background garbage collection](#)
- [Runtime configuration options for garbage collection](#)

Background garbage collection

9/20/2022 • 2 minutes to read • [Edit Online](#)

In background garbage collection (GC), ephemeral generations (0 and 1) are collected as needed while the collection of generation 2 is in progress. Background garbage collection is performed on one or more dedicated threads, depending on whether it's workstation or server GC, and applies only to generation 2 collections.

Background garbage collection is enabled by default. It can be enabled or disabled with the `gcConcurrent` configuration setting in .NET Framework apps or the `System.GC.Concurrent` setting in .NET Core and .NET 5 and later apps.

NOTE

Background garbage collection replaces [concurrent garbage collection](#) and is available in .NET Framework 4 and later versions. In .NET Framework 4, it's supported only for *workstation* garbage collection. Starting with .NET Framework 4.5, background garbage collection is available for both *workstation* and *server* garbage collection.

A collection on ephemeral generations during background garbage collection is known as *foreground* garbage collection. When foreground garbage collections occur, all managed threads are suspended.

When background garbage collection is in progress and you've allocated enough objects in generation 0, the CLR performs a generation 0 or generation 1 foreground garbage collection. The dedicated background garbage collection thread checks at frequent safe points to determine whether there is a request for foreground garbage collection. If there is, the background collection suspends itself so that foreground garbage collection can occur. After the foreground garbage collection is completed, the dedicated background garbage collection threads and user threads resume.

Background garbage collection removes allocation restrictions imposed by concurrent garbage collection, because ephemeral garbage collections can occur during background garbage collection. Background garbage collection can remove dead objects in ephemeral generations. It can also expand the heap if needed during a generation 1 garbage collection.

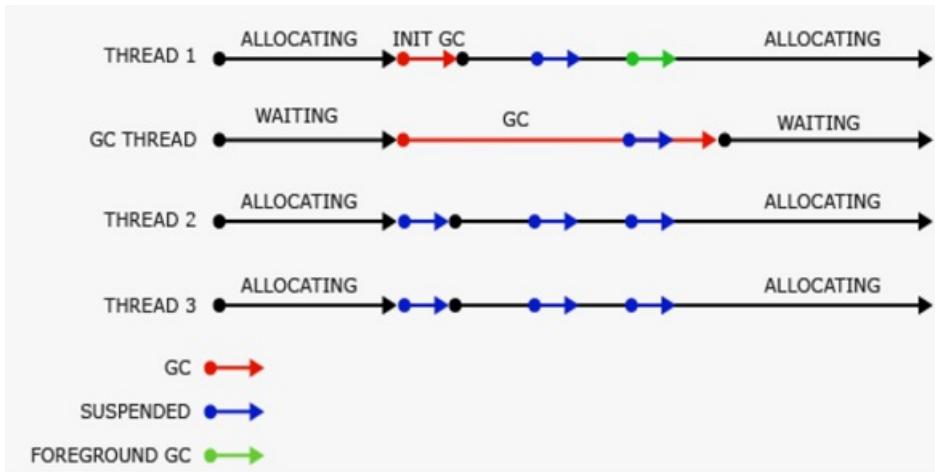
Background workstation vs. server GC

Starting with .NET Framework 4.5, background garbage collection is available for server GC. Background GC is the default mode for server garbage collection.

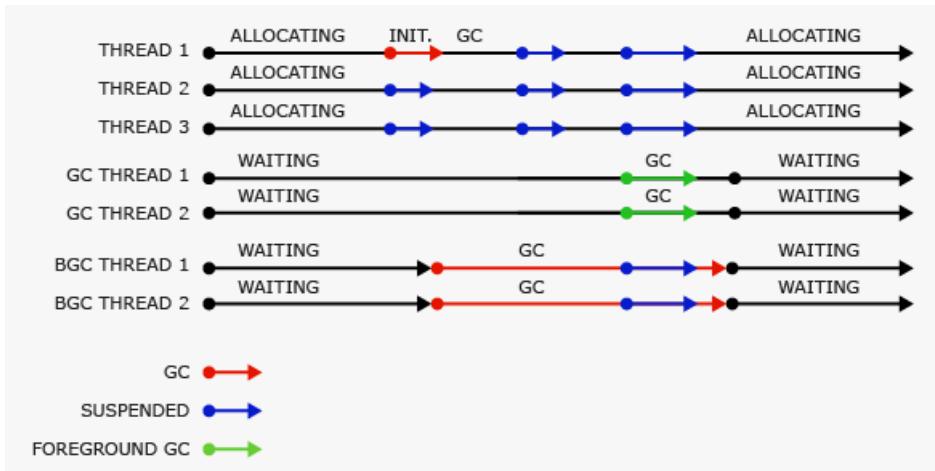
Background server garbage collection functions similarly to background workstation garbage collection, but there are a few differences:

- Background workstation garbage collection uses one dedicated background garbage collection thread, whereas background server garbage collection uses multiple threads. Typically, there's a dedicated thread for each logical processor.
- Unlike the workstation background garbage collection thread, the background server GC threads do not time out.

The following illustration shows background *workstation* garbage collection performed on a separate, dedicated thread:



The following illustration shows background *server* garbage collection performed on separate, dedicated threads:



Concurrent garbage collection

TIP

This section applies to:

- .NET Framework 3.5 and earlier for workstation garbage collection
- .NET Framework 4 and earlier for server garbage collection

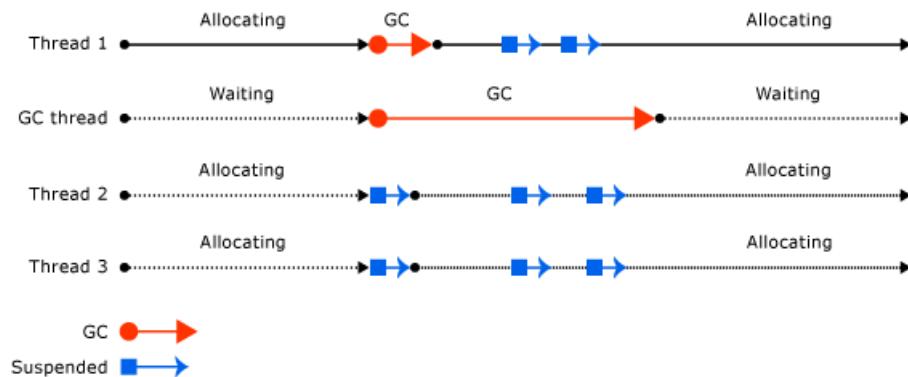
Concurrent garbage is replaced by background garbage collection in later versions.

In workstation or server garbage collection, you can [enable concurrent garbage collection](#), which enables threads to run concurrently with a dedicated thread that performs the garbage collection for most of the duration of the collection. This option affects only garbage collections in generation 2; generations 0 and 1 are always non-concurrent because they finish fast.

Concurrent garbage collection enables interactive applications to be more responsive by minimizing pauses for a collection. Managed threads can continue to run most of the time while the concurrent garbage collection thread is running. This design results in shorter pauses while a garbage collection is occurring.

Concurrent garbage collection is performed on a dedicated thread. By default, the CLR runs workstation garbage collection with concurrent garbage collection enabled on both single-processor and multi-processor computers.

The following illustration shows concurrent garbage collection performed on a separate dedicated thread.



See also

- [Workstation and server garbage collection](#)
- [Runtime configuration options for garbage collection](#)

The large object heap on Windows systems

9/20/2022 • 15 minutes to read • [Edit Online](#)

The .NET garbage collector (GC) divides objects up into small and large objects. When an object is large, some of its attributes become more significant than if the object is small. For instance, compacting it—that is, copying it in memory elsewhere on the heap—can be expensive. Because of this, the garbage collector places large objects on the large object heap (LOH). This article discusses what qualifies an object as a large object, how large objects are collected, and what kind of performance implications large objects impose.

IMPORTANT

This article discusses the large object heap in .NET Framework and .NET Core running on Windows systems only. It does not cover the LOH running on .NET implementations on other platforms.

How an object ends up on the LOH

If an object is greater than or equal to 85,000 bytes in size, it's considered a large object. This number was determined by performance tuning. When an object allocation request is for 85,000 or more bytes, the runtime allocates it on the large object heap.

To understand what this means, it's useful to examine some fundamentals about the garbage collector.

The garbage collector is a generational collector. It has three generations: generation 0, generation 1, and generation 2. The reason for having three generations is that, in a well-tuned app, most objects die in gen0. For example, in a server app, the allocations associated with each request should die after the request is finished. The in-flight allocation requests will make it into gen1 and die there. Essentially, gen1 acts as a buffer between young object areas and long-lived object areas.

Newly allocated objects form a new generation of objects and are implicitly generation 0 collections. However, if they are large objects, they go on the large object heap (LOH), which is sometimes referred to as generation 3. Generation 3 is a physical generation that's logically collected as part of generation 2.

Large objects belong to generation 2 because they are collected only during a generation 2 collection. When a generation is collected, all its younger generation(s) are also collected. For example, when a generation 1 GC happens, both generation 1 and 0 are collected. And when a generation 2 GC happens, the whole heap is collected. For this reason, a generation 2 GC is also called a *full GC*. This article refers to generation 2 GC instead of full GC, but the terms are interchangeable.

Generations provide a logical view of the GC heap. Physically, objects live in managed heap segments. A *managed heap segment* is a chunk of memory that the GC reserves from the OS by calling the [VirtualAlloc function](#) on behalf of managed code. When the CLR is loaded, the GC allocates two initial heap segments: one for small objects (the small object heap, or SOH), and one for large objects (the large object heap).

The allocation requests are then satisfied by putting managed objects on these managed heap segments. If the object is less than 85,000 bytes, it is put on the segment for the SOH; otherwise, it is put on an LOH segment. Segments are committed (in smaller chunks) as more and more objects are allocated onto them. For the SOH, objects that survive a GC are promoted to the next generation. Objects that survive a generation 0 collection are now considered generation 1 objects, and so on. However, objects that survive the oldest generation are still considered to be in the oldest generation. In other words, survivors from generation 2 are generation 2 objects; and survivors from the LOH are LOH objects (which are collected with gen2).

User code can only allocate in generation 0 (small objects) or the LOH (large objects). Only the GC can "allocate" objects in generation 1 (by promoting survivors from generation 0) and generation 2 (by promoting survivors from generation 1).

When a garbage collection is triggered, the GC traces through the live objects and compacts them. But because compaction is expensive, the GC *sweeps* the LOH; it makes a free list out of dead objects that can be reused later to satisfy large object allocation requests. Adjacent dead objects are made into one free object.

.NET Core and .NET Framework (starting with .NET Framework 4.5.1) include the [GCSettings.LargeObjectHeapCompactionMode](#) property that allows users to specify that the LOH should be compacted during the next full blocking GC. And in the future, .NET may decide to compact the LOH automatically. This means that, if you allocate large objects and want to make sure that they don't move, you should still pin them.

Figure 1 illustrates a scenario where the GC forms generation 1 after the first generation 0 GC where `Obj1` and `Obj3` are dead, and it forms generation 2 after the first generation 1 GC where `Obj2` and `Obj5` are dead. Note that this and the following figures are only for illustration purposes; they contain very few objects to better show what happens on the heap. In reality, many more objects are typically involved in a GC.

Fig. 1 – SOH Allocations And GCs

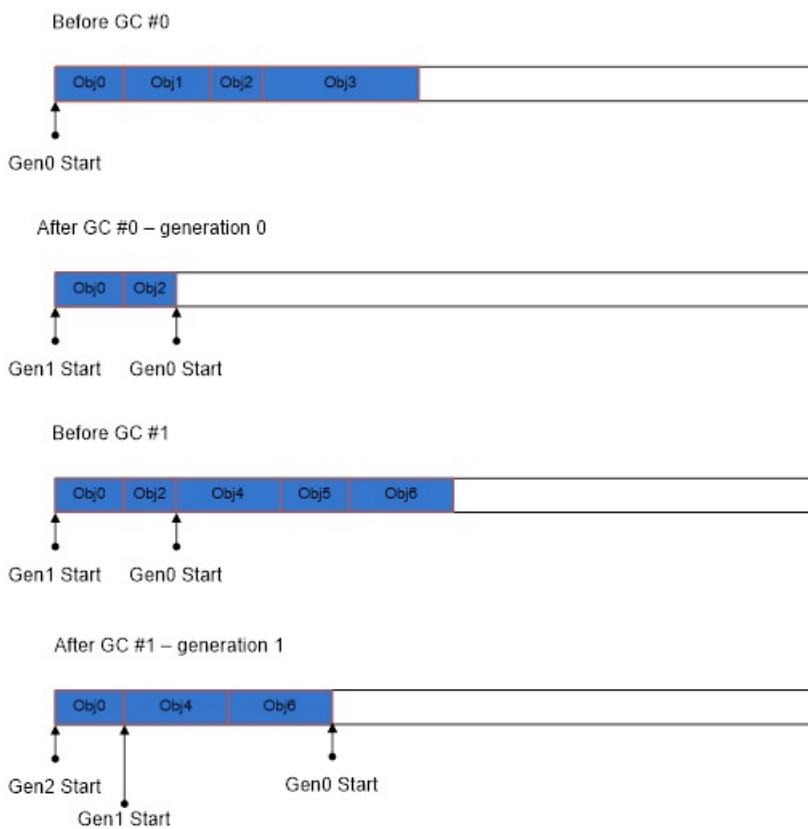


Figure 1: A generation 0 and a generation 1 GC.

Figure 2 shows that after a generation 2 GC that saw that `Obj1` and `Obj2` are dead, the GC forms contiguous free space out of memory that used to be occupied by `Obj1` and `Obj2`, which then was used to satisfy an allocation request for `Obj4`. The space after the last object, `Obj3`, to end of the segment can also be used to satisfy allocation requests.

Fig. 2 – LOH Allocations And GCs

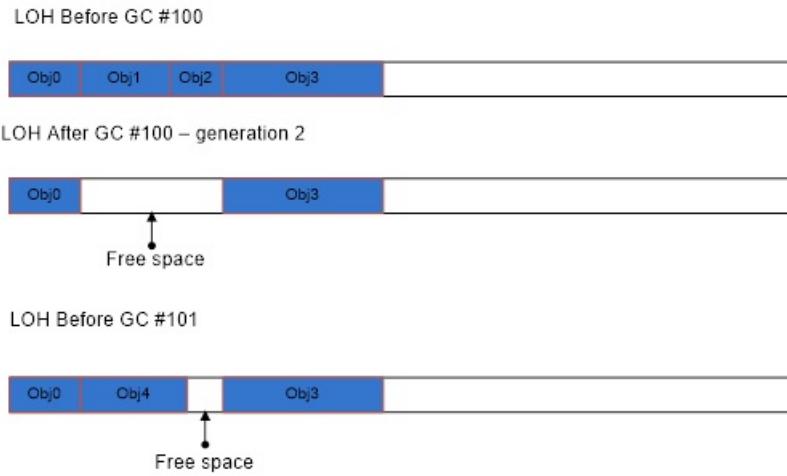


Figure 2: After a generation 2 GC

If there isn't enough free space to accommodate the large object allocation requests, the GC first attempts to acquire more segments from the OS. If that fails, it triggers a generation 2 GC in the hope of freeing up some space.

During a generation 1 or generation 2 GC, the garbage collector releases segments that have no live objects on them back to the OS by calling the [VirtualFree function](#). Space after the last live object to the end of the segment is decommitted (except on the ephemeral segment where gen0/gen1 live, where the garbage collector does keep some committed because your application will be allocating in it right away). And the free spaces remain committed though they are reset, meaning that the OS doesn't need to write data in them back to disk.

Since the LOH is only collected during generation 2 GCs, the LOH segment can only be freed during such a GC. Figure 3 illustrates a scenario where the garbage collector releases one segment (segment 2) back to the OS and decommits more space on the remaining segments. If it needs to use the decommitted space at the end of the segment to satisfy large object allocation requests, it commits the memory again. (For an explanation of commit/decommit, see the documentation for [VirtualAlloc](#).

Fig. 3 – Dead segments released on LOH during GC

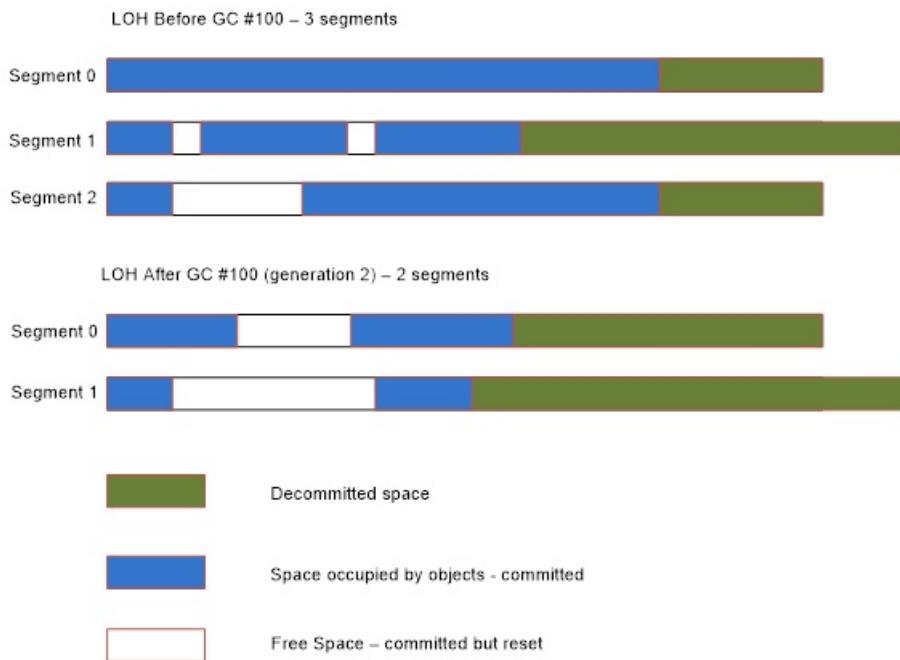


Figure 3: The LOH after a generation 2 GC

When is a large object collected?

In general, a GC occurs under one of the following three conditions:

- Allocation exceeds the generation 0 or large object threshold.

The threshold is a property of a generation. A threshold for a generation is set when the garbage collector allocates objects into it. When the threshold is exceeded, a GC is triggered on that generation. When you allocate small or large objects, you consume generation 0 and the LOH's thresholds, respectively. When the garbage collector allocates into generation 1 and 2, it consumes their thresholds. These thresholds are dynamically tuned as the program runs.

This is the typical case; most GCs happen because of allocations on the managed heap.

- The [GC.Collect](#) method is called.

If the parameterless [GC.Collect\(\)](#) method is called or another overload is passed [GC.MaxGeneration](#) as an argument, the LOH is collected along with the rest of the managed heap.

- The system is in low memory situation.

This occurs when the garbage collector receives a high memory notification from the OS. If the garbage collector thinks that doing a generation 2 GC will be productive, it triggers one.

LOH performance implications

Allocations on the large object heap impact performance in the following ways.

- Allocation cost.

The CLR makes the guarantee that the memory for every new object it gives out is cleared. This means the allocation cost of a large object is dominated by memory clearing (unless it triggers a GC). If it takes two cycles to clear one byte, it takes 170,000 cycles to clear the smallest large object. Clearing the memory of a 16-MB object on a 2-GHz machine takes approximately 16 ms. That's a rather large cost.

- Collection cost.

Because the LOH and generation 2 are collected together, if either one's threshold is exceeded, a generation 2 collection is triggered. If a generation 2 collection is triggered because of the LOH, generation 2 won't necessarily be much smaller after the GC. If there's not much data on generation 2, this has minimal impact. But if generation 2 is large, it can cause performance problems if many generation 2 GCs are triggered. If many large objects are allocated on a temporary basis and you have a large SOH, you could be spending too much time doing GCs. In addition, the allocation cost can really add up if you keep allocating and letting go of really large objects.

- Array elements with reference types.

Very large objects on the LOH are usually arrays (it's very rare to have an instance object that's really large). If the elements of an array are reference-rich, it incurs a cost that is not present if the elements are not reference-rich. If the element doesn't contain any references, the garbage collector doesn't need to go through the array at all. For example, if you use an array to store nodes in a binary tree, one way to implement it is to refer to a node's right and left node by the actual nodes:

```
class Node
{
    Data d;
    Node left;
    Node right;
};

Node[] binary_tr = new Node [num_nodes];
```

If `num_nodes` is large, the garbage collector needs to go through at least two references per element. An alternative approach is to store the index of the right and the left nodes:

```
class Node
{
    Data d;
    uint left_index;
    uint right_index;
}
```

Instead of referring the left node's data as `left.d`, you refer to it as `binary_tr[left_index].d`. And the garbage collector doesn't need to look at any references for the left and right node.

Out of the three factors, the first two are usually more significant than the third. Because of this, we recommend that you allocate a pool of large objects that you reuse instead of allocating temporary ones.

Collect performance data for the LOH

Before you collect performance data for a specific area, you should already have done the following:

1. Found evidence that you should be looking at this area.
2. Exhausted other areas that you know of without finding anything that could explain the performance problem you saw.

For more information on the fundamentals of memory and the CPU, see the blog [Understand the problem before you try to find a solution](#).

You can use the following tools to collect data on LOH performance:

- [.NET CLR memory performance counters](#)
- [ETW events](#)
- [A debugger](#)

.NET CLR Memory Performance counters

These performance counters are usually a good first step in investigating performance issues (although we recommend that you use [ETW events](#)). You configure Performance Monitor by adding the counters that you want, as Figure 4 shows. The ones that are relevant for the LOH are:

- **Gen 2 Collections**

Displays the number of times generation 2 GCs have occurred since the process started. The counter is incremented at the end of a generation 2 collection (also called a full garbage collection). This counter displays the last observed value.

- **Large Object Heap size**

Displays the current size, in bytes, including free space, of the LOH. This counter is updated at the end of a garbage collection, not at each allocation.

A common way to look at performance counters is with Performance Monitor (perfmon.exe). Use "Add Counters" to add the interesting counter for processes that you care about. You can save the performance counter data to a log file, as Figure 4 shows:

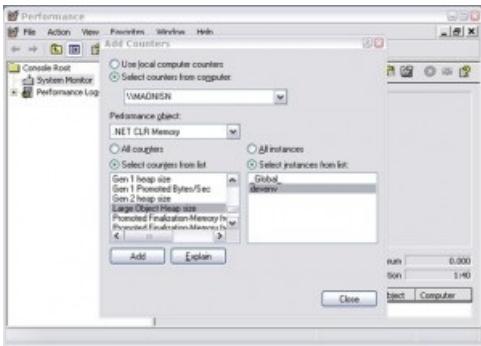


Figure 4: The LOH after a generation 2 GC

Performance counters can also be queried programmatically. Many people collect them this way as part of their routine testing process. When they spot counters with values that are out of the ordinary, they use other means to get more detailed data to help with the investigation.

NOTE

We recommend that you to use ETW events instead of performance counters, because ETW provides much richer information.

ETW events

The garbage collector provides a rich set of ETW events to help you understand what the heap is doing and why. The following blog posts show how to collect and understand GC events with ETW:

- [GC ETW Events - 1](#)
- [GC ETW Events - 2](#)
- [GC ETW Events - 3](#)
- [GC ETW Events - 4](#)

To identify excessive generation 2 GCs caused by temporary LOH allocations, look at the Trigger Reason column for GCs. For a simple test that only allocates temporary large objects, you can collect information on ETW events with the following [PerfView](#) command line:

```
perfview /GCCollectOnly /AcceptEULA /nogui collect
```

The result is something like this:

Gen 2 for Process 10720: testlarge

GC Index	Pause Start	Trigger Reason	Gen	Suspend MSec	Pause MSec	% Pause	% GC Alloc MB	Gen0 Alloc Rate MB/sec	Peak MB	After MB	Ratio	Promoted MB	Gen0 Survival Rate %	Gen0 Frag %	Gen1 MB	Gen1 Survival Rate %	Gen1 Frag %	Gen2 MB	Gen2 Survival Rate %	Gen2 Frag %	LOH MB	LOH Survival Rate %	LOH Frag %	Finalizable Surv MB	Pinned Obj	
									GC Events by Time		All times are in msec. Hover over columns for help.															
1	2,248,409	AllocLarge	2N	0.007	0.422	8.0	Kn0 0.000	0.00	3,244	0.043	76.19	0.043	0.000	85	0.000	0.007	0	15.25	0.000	0	0.00	0.035	1	0.36	0.00	5
2	2,248,501	AllocLarge	2N	0.001	0.056	43.3	Kn0 0.000	0.00	3,244	0.043	76.14	0.043	0.000	0	0.000	0.000	100	0.000	0.007	0	15.53	0.035	1	0.36	0.00	5
3	2,249,001	AllocLarge	2N	0.001	0.052	52.9	Kn0 0.000	0.00	3,244	0.043	76.15	0.043	0.000	0	0.000	0.000	0	0.000	0.007	100	15.53	0.035	1	0.36	0.00	5
4	2,249,101	AllocLarge	2N	0.001	0.051	51.8	Kn0 0.000	0.00	3,244	0.043	76.15	0.043	0.000	0	0.000	0.000	0	0.000	0.007	100	15.53	0.035	1	0.36	0.00	5
5	2,249,191	AllocLarge	2N	0.001	0.051	51.4	Kn0 0.000	0.00	3,244	0.043	76.15	0.043	0.000	0	0.000	0.000	0	0.000	0.007	100	15.53	0.035	1	0.36	0.00	5
6	2,249,201	AllocLarge	2N	0.001	0.051	52.5	Kn0 0.000	0.00	3,244	0.043	76.15	0.043	0.000	0	0.000	0.000	0	0.000	0.007	100	15.53	0.035	1	0.36	0.00	5
7	2,249,301	AllocLarge	2N	0.001	0.052	53.1	Kn0 0.000	0.00	3,244	0.043	76.15	0.043	0.000	0	0.000	0.000	0	0.000	0.007	100	15.53	0.035	1	0.36	0.00	5
8	2,249,401	AllocLarge	2N	0.001	0.051	52.7	Kn0 0.000	0.00	3,244	0.043	76.15	0.043	0.000	0	0.000	0.000	0	0.000	0.007	100	15.53	0.035	1	0.36	0.00	5
9	2,249,501	AllocLarge	2N	0.001	0.043	53.4	Kn0 0.000	0.00	3,244	0.043	76.15	0.043	0.000	0	0.000	0.000	0	0.000	0.007	100	15.53	0.035	1	0.36	0.00	5
10	2,249,601	AllocLarge	2N	0.001	0.043	53.2	Kn0 0.000	0.00	3,244	0.043	76.15	0.043	0.000	0	0.000	0.000	0	0.000	0.007	100	15.53	0.035	1	0.36	0.00	5
11	2,249,701	AllocLarge	2N	0.001	0.048	46.3	Kn0 0.000	0.00	3,244	0.043	76.15	0.043	0.000	0	0.000	0.000	0	0.000	0.007	100	15.53	0.035	1	0.36	0.00	5
12	2,249,801	AllocLarge	2N	0.001	0.061	63.7	Kn0 0.000	0.00	3,244	0.043	76.15	0.043	0.000	0	0.000	0.000	0	0.000	0.007	100	15.53	0.035	1	0.36	0.00	5
13	2,249,901	AllocLarge	2N	0.001	0.061	64.9	Kn0 0.000	0.00	3,244	0.043	76.15	0.043	0.000	0	0.000	0.000	0	0.000	0.007	100	15.53	0.035	1	0.36	0.00	5
14	2,250,001	AllocLarge	2N	0.001	0.037	39.5	Kn0 0.000	0.00	3,244	0.043	76.15	0.043	0.000	0	0.000	0.000	0	0.000	0.007	100	15.53	0.035	1	0.36	0.00	5
15	2,250,101	AllocLarge	2N	0.001	0.051	52.2	Kn0 0.000	0.00	3,244	0.043	76.15	0.043	0.000	0	0.000	0.000	0	0.000	0.007	100	15.53	0.035	1	0.36	0.00	5
16	2,250,201	AllocLarge	2N	0.001	0.042	51.2	Kn0 0.005	121.73	3,249	0.047	68.47	0.046	0.000	99	0.000	0.005	0	0.49	0.007	100	15.53	0.035	1	0.36	0.00	5
17	2,250,301	AllocLarge	2N	0.001	0.040	52.0	Kn0 0.000	0.00	3,249	0.047	68.43	0.046	0.000	0	0.000	0.000	0	0.000	0.012	100	9.69	0.035	1	0.36	0.00	5
18	2,250,381	AllocLarge	2N	0.001	0.039	51.7	Kn0 0.000	0.00	3,249	0.047	68.43	0.046	0.000	0	0.000	0.000	0	0.000	0.012	100	9.69	0.035	1	0.36	0.00	5
19	2,250,451	AllocLarge	2N	0.001	0.038	56.6	Kn0 0.000	0.00	3,249	0.047	68.43	0.046	0.000	0	0.000	0.000	0	0.000	0.012	100	9.69	0.035	1	0.36	0.00	5
20	2,250,531	AllocLarge	2N	0.001	0.039	56.9	Kn0 0.000	0.00	3,249	0.047	68.43	0.046	0.000	0	0.000	0.000	0	0.000	0.012	100	9.69	0.035	1	0.36	0.00	5
21	2,250,601	AllocLarge	2N	0.001	0.042	53.1	Kn0 0.000	0.00	3,249	0.047	68.43	0.046	0.000	0	0.000	0.000	0	0.000	0.012	100	9.69	0.035	1	0.36	0.00	5
22	2,250,681	AllocLarge	2N	0.001	0.042	52.7	Kn0 0.000	0.00	3,249	0.047	68.43	0.046	0.000	0	0.000	0.000	0	0.000	0.012	100	9.69	0.035	1	0.36	0.00	5
23	2,250,761	AllocLarge	2N	0.001	0.044	54.6	Kn0 0.000	0.00	3,249	0.047	68.43	0.046	0.000	0	0.000	0.000	0	0.000	0.012	100	9.69	0.035	1	0.36	0.00	5
24	2,250,841	AllocLarge	2N	0.001	0.039	51.3	Kn0 0.000	0.00	3,249	0.047	68.43	0.046	0.000	0	0.000	0.000	0	0.000	0.012	100	9.69	0.035	1	0.36	0.00	5

Figure 5: ETW events shown using PerfView

As you can see, all GCs are generation 2 GCs, and they are all triggered by AllocLarge, which means that allocating a large object triggered this GC. We know that these allocations are temporary because the LOH

Survival Rate % column says 1%.

You can collect additional ETW events that tell you who allocated these large objects. The following command line:

```
perfview /GOnly /AcceptEULA /nogui collect
```

collects an AllocationTick event, which is fired approximately every 100k worth of allocations. In other words, an event is fired each time a large object is allocated. You can then look at one of the GC Heap Alloc views, which show you the callstacks that allocated large objects:

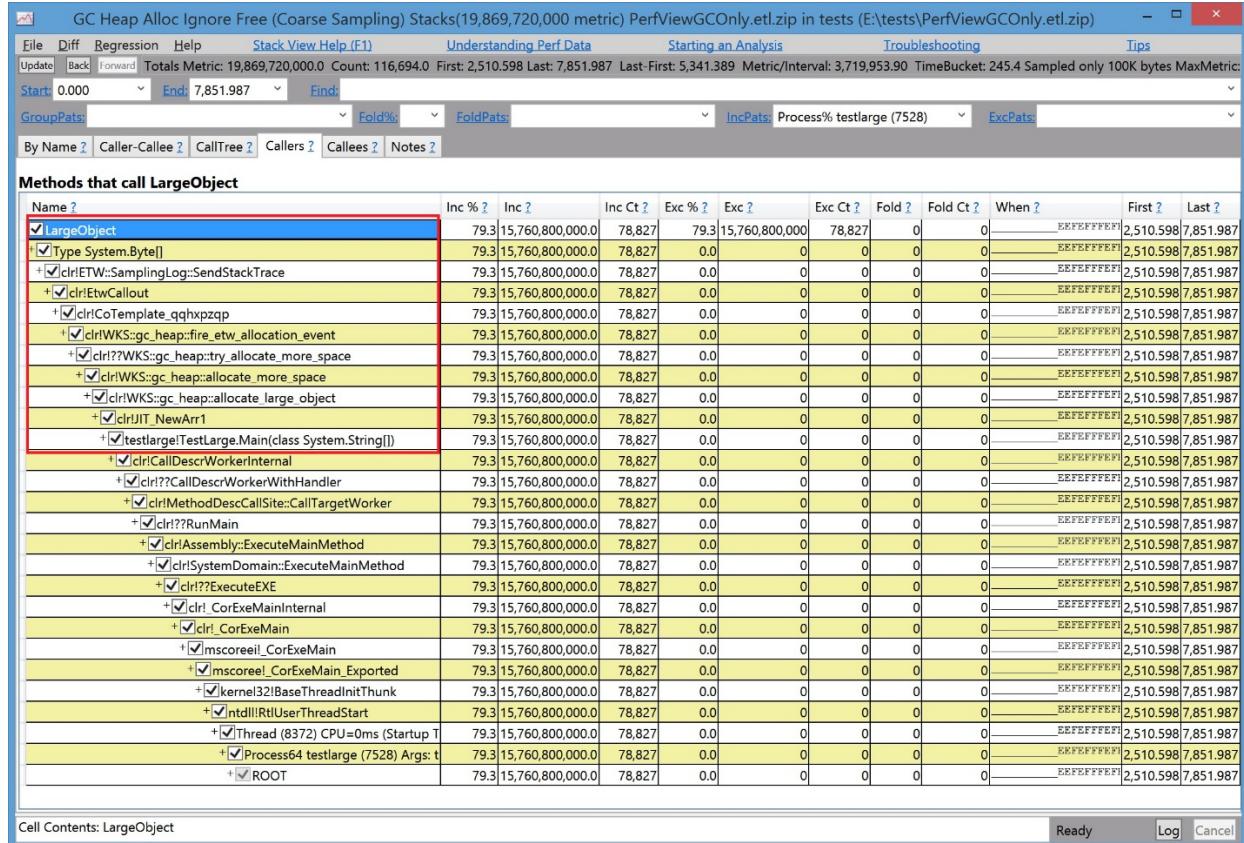


Figure 6: A GC Heap Alloc view

As you can see, this is a very simple test that just allocates large objects from its `Main` method.

A debugger

If all you have is a memory dump and you need to look at what objects are actually on the LOH, you can use the [SoS debugger extension](#) provided by .NET.

NOTE

The debugging commands mentioned in this section are applicable to the [Windows Debuggers](#).

The following shows sample output from analyzing the LOH:

```

0:003> .loadby sos mscorewks
0:003> !eeheap -gc
Number of GC Heaps: 1
generation 0 starts at 0x013e35ec
sgeneration 1 starts at 0x013e1b6c
generation 2 starts at 0x013e1000
ephemeral segment allocation context: none
segment begin allocated size
0018f2d0 790d5588 790f4b38 0x0001f5b0(128432)
013e0000 013e1000 013e35f8 0x000025f8(9720)
Large object heap starts at 0x023e1000
segment begin allocated size
023e0000 023e1000 033db630 0x00ffa630(16754224)
033e0000 033e1000 043cdf98 0x00fecf98(16699288)
043e0000 043e1000 05368b58 0x00f87b58(16284504)
Total Size 0x2f90cc8(49876168)
-----
GC Heap Size 0x2f90cc8(49876168)
0:003> !dumpheap -stat 023e1000 033db630
total 133 objects
Statistics:
MT      Count    TotalSize Class Name
001521d0       66     2081792     Free
7912273c       63     6663696 System.Byte[]
7912254c        4     8008736 System.Object[]
Total 133 objects

```

The LOH heap size is $(16,754,224 + 16,699,288 + 16,284,504) = 49,738,016$ bytes. Between addresses 023e1000 and 033db630, 8,008,736 bytes are occupied by an array of [System.Object](#) objects, 6,663,696 bytes are occupied by an array of [System.Byte](#) objects, and 2,081,792 bytes are occupied by free space.

Sometimes, the debugger shows that the total size of the LOH is less than 85,000 bytes. This happens because the runtime itself uses the LOH to allocate some objects that are smaller than a large object.

Because the LOH is not compacted, sometimes the LOH is thought to be the source of fragmentation.

Fragmentation means:

- Fragmentation of the managed heap, which is indicated by the amount of free space between managed objects. In SoS, the `!dumpheap -type Free` command displays the amount of free space between managed objects.
- Fragmentation of the virtual memory (VM) address space, which is the memory marked as `MEM_FREE`. You can get it by using various debugger commands in windbg.

The following example shows fragmentation in the VM space:

```

0:000> !address
00000000 : 00000000 - 00010000
Type      00000000
Protect  00000001 PAGE_NOACCESS
State     00010000 MEM_FREE
Usage    RegionUsageFree
00010000 : 00010000 - 00002000
Type      00020000 MEM_PRIVATE
Protect  00000004 PAGE_READWRITE
State     00001000 MEM_COMMIT
Usage    RegionUsageEnvironmentBlock
00012000 : 00012000 - 0000e000
Type      00000000
Protect  00000001 PAGE_NOACCESS
State     00010000 MEM_FREE
Usage    RegionUsageFree
... [omitted]

----- Usage SUMMARY -----
TotSize (   KB) Pct(Tots) Pct(Busy)  Usage
701000 (  7172) : 00.34%  20.69%  : RegionUsageIsVAD
7de15000 ( 2062420) : 98.35%  00.00%  : RegionUsageFree
1452000 (  20808) : 00.99%  60.02%  : RegionUsageImage
300000 (  3072) : 00.15%  08.86%  : RegionUsageStack
3000 (    12) : 00.00%  00.03%  : RegionUsageTeb
381000 (  3588) : 00.17%  10.35%  : RegionUsageHeap
0 (      0) : 00.00%  00.00%  : RegionUsagePageHeap
1000 (     4) : 00.00%  00.01%  : RegionUsagePeb
1000 (     4) : 00.00%  00.01%  : RegionUsageProcessParametrs
2000 (     8) : 00.00%  00.02%  : RegionUsageEnvironmentBlock
Tot: 7fff0000 (2097088 KB) Busy: 021db000 (34668 KB)

----- Type SUMMARY -----
TotSize (   KB) Pct(Tots) Usage
7de15000 ( 2062420) : 98.35%  : <free>
1452000 (  20808) : 00.99%  : MEM_IMAGE
69f000 (  6780) : 00.32%  : MEM_MAPPED
6ea000 (  7080) : 00.34%  : MEM_PRIVATE

----- State SUMMARY -----
TotSize (   KB) Pct(Tots) Usage
1a58000 ( 26976) : 01.29%  : MEM_COMMIT
7de15000 ( 2062420) : 98.35%  : MEM_FREE
783000 (  7692) : 00.37%  : MEM_RESERVE

Largest free region: Base 01432000 - Size 707ee000 (1843128 KB)

```

It's more common to see VM fragmentation caused by temporary large objects that require the garbage collector to frequently acquire new managed heap segments from the OS and to release empty ones back to the OS.

To verify whether the LOH is causing VM fragmentation, you can set a breakpoint on [VirtualAlloc](#) and [VirtualFree](#) to see who called them. For example, to see who tried to allocate virtual memory chunks larger than 8 MB from the OS, you can set a breakpoint like this:

```
bp kernel32!virtualalloc "j (dwo(@esp+8)>800000) 'kb';'g'"
```

This command breaks into the debugger and shows the call stack only if [VirtualAlloc](#) is called with an allocation size greater than 8 MB (0x800000).

CLR 2.0 added a feature called *VM Hoarding* that can be useful for scenarios where segments (including on the large and small object heaps) are frequently acquired and released. To specify VM Hoarding, you specify a startup flag called `STARTUP_HOARD_GC_VM` via the hosting API. Instead of releasing empty segments back to the OS,

the CLR decommits the memory on these segments and puts them on a standby list. (Note that the CLR doesn't do this for segments that are too large.) The CLR later uses those segments to satisfy new segment requests. The next time that your app needs a new segment, the CLR uses one from this standby list if it can find one that's large enough.

VM hoarding is also useful for applications that want to hold onto the segments that they already acquired, such as some server apps that are the dominant apps running on the system, to avoid out-of-memory exceptions.

We strongly recommend that you carefully test your application when you use this feature to ensure your application has fairly stable memory usage.

Garbage collection and performance

9/20/2022 • 25 minutes to read • [Edit Online](#)

This article describes issues related to garbage collection and memory usage. It addresses issues that pertain to the managed heap and explains how to minimize the effect of garbage collection on your applications. Each issue has links to procedures that you can use to investigate problems.

Performance analysis tools

The following sections describe the tools that are available for investigating memory usage and garbage collection issues. The [procedures](#) provided later in this article refer to these tools.

Memory Performance Counters

You can use performance counters to gather performance data. For instructions, see [Runtime Profiling](#). The .NET CLR Memory category of performance counters, as described in [Performance Counters in .NET](#), provides information about the garbage collector.

Debugging with SOS

You can use the [Windows Debugger \(WinDbg\)](#) to inspect objects on the managed heap.

To install WinDbg, install Debugging Tools for Windows from the [Download Debugging Tools for Windows](#) page.

Garbage Collection ETW Events

Event tracing for Windows (ETW) is a tracing system that supplements the profiling and debugging support provided by .NET. Starting with .NET Framework 4, [garbage collection ETW events](#) capture useful information for analyzing the managed heap from a statistical point of view. For example, the `GCStart_V1` event, which is raised when a garbage collection is about to occur, provides the following information:

- Which generation of objects is being collected.
- What triggered the garbage collection.
- Type of garbage collection (concurrent or not concurrent).

ETW event logging is efficient and will not mask any performance problems associated with garbage collection. A process can provide its own events in conjunction with ETW events. When logged, both the application's events and the garbage collection events can be correlated to determine how and when heap problems occur. For example, a server application could provide events at the start and end of a client request.

The Profiling API

The common language runtime (CLR) profiling interfaces provide detailed information about the objects that were affected during garbage collection. A profiler can be notified when a garbage collection starts and ends. It can provide reports about the objects on the managed heap, including an identification of objects in each generation. For more information, see [Profiling Overview](#).

Profilers can provide comprehensive information. However, complex profilers can potentially modify an application's behavior.

Application Domain Resource Monitoring

Starting with .NET Framework 4, Application domain resource monitoring (ARM) enables hosts to monitor CPU and memory usage by application domain. For more information, see [Application Domain Resource Monitoring](#).

Troubleshoot performance issues

The first step is to [determine whether the issue is actually garbage collection](#). If you determine that it is, select from the following list to troubleshoot the problem.

- [An out-of-memory exception is thrown](#)
- [The process uses too much memory](#)
- [The garbage collector does not reclaim objects fast enough](#)
- [The managed heap is too fragmented](#)
- [Garbage collection pauses are too long](#)
- [Generation 0 is too big](#)
- [CPU usage during a garbage collection is too high](#)

Issue: An Out-of-Memory Exception Is Thrown

There are two legitimate cases for a managed [OutOfMemoryException](#) to be thrown:

- Running out of virtual memory.

The garbage collector allocates memory from the system in segments of a pre-determined size. If an allocation requires an additional segment, but there is no contiguous free block left in the process's virtual memory space, the allocation for the managed heap will fail.

- Not having enough physical memory to allocate.

PERFORMANCE CHECKS

[Determine whether the out-of-memory exception is managed.](#)

[Determine how much virtual memory can be reserved.](#)

[Determine whether there is enough physical memory.](#)

If you determine that the exception is not legitimate, contact Microsoft Customer Service and Support with the following information:

- The stack with the managed out-of-memory exception.
- Full memory dump.
- Data that proves that it is not a legitimate out-of-memory exception, including data that shows that virtual or physical memory is not an issue.

Issue: The Process Uses Too Much Memory

A common assumption is that the memory usage display on the **Performance** tab of Windows Task Manager can indicate when too much memory is being used. However, that display pertains to the working set; it does not provide information about virtual memory usage.

If you determine that the issue is caused by the managed heap, you must measure the managed heap over time to determine any patterns.

If you determine that the problem is not caused by the managed heap, you must use native debugging.

PERFORMANCE CHECKS

[Determine how much virtual memory can be reserved.](#)

[Determine how much memory the managed heap is committing.](#)

[Determine how much memory the managed heap reserves.](#)

[Determine large objects in generation 2.](#)

[Determine references to objects.](#)

Issue: The Garbage Collector Does Not Reclaim Objects Fast Enough

When it appears as if objects are not being reclaimed as expected for garbage collection, you must determine if

there are any strong references to those objects.

You may also encounter this issue if there has been no garbage collection for the generation that contains a dead object, which indicates that the finalizer for the dead object has not been run. For example, this is possible when you are running a single-threaded apartment (STA) application and the thread that services the finalizer queue cannot call into it.

PERFORMANCE CHECKS

[Check references to objects.](#)

[Determine whether a finalizer has been run.](#)

[Determine whether there are objects waiting to be finalized.](#)

Issue: The Managed Heap Is Too Fragmented

The fragmentation level is calculated as the ratio of free space over the total allocated memory for the generation. For generation 2, an acceptable level of fragmentation is no more than 20%. Because generation 2 can get very big, the ratio of fragmentation is more important than the absolute value.

Having lots of free space in generation 0 is not a problem because this is the generation where new objects are allocated.

Fragmentation always occurs in the large object heap because it is not compacted. Free objects that are adjacent are naturally collapsed into a single space to satisfy large object allocation requests.

Fragmentation can become a problem in generation 1 and generation 2. If these generations have a large amount of free space after a garbage collection, an application's object usage may need modification, and you should consider re-evaluating the lifetime of long-term objects.

Excessive pinning of objects can increase fragmentation. If fragmentation is high, too many objects could have been pinned.

If fragmentation of virtual memory is preventing the garbage collector from adding segments, the causes could be one of the following:

- Frequent loading and unloading of many small assemblies.
- Holding too many references to COM objects when interoperating with unmanaged code.
- Creation of large transient objects, which causes the large object heap to allocate and free heap segments frequently.

When hosting the CLR, an application can request that the garbage collector retain its segments. This reduces the frequency of segment allocations. This is accomplished by using the STARTUP_HOARD_GC_VM flag in the [STARTUP_FLAGS Enumeration](#).

PERFORMANCE CHECKS

[Determine the amount of free space in the managed heap.](#)

[Determine the number of pinned objects.](#)

If you think that there is no legitimate cause for the fragmentation, contact Microsoft Customer Service and Support.

Issue: Garbage Collection Pauses Are Too Long

Garbage collection operates in soft real time, so an application must be able to tolerate some pauses. A criterion for soft real time is that 95% of the operations must finish on time.

In concurrent garbage collection, managed threads are allowed to run during a collection, which means that

pauses are very minimal.

Ephemeral garbage collections (generations 0 and 1) last only a few milliseconds, so decreasing pauses is usually not feasible. However, you can decrease the pauses in generation 2 collections by changing the pattern of allocation requests by an application.

Another, more accurate, method is to use [garbage collection ETW events](#). You can find the timings for collections by adding the time stamp differences for a sequence of events. The whole collection sequence includes suspension of the execution engine, the garbage collection itself, and the resumption of the execution engine.

You can use [Garbage Collection Notifications](#) to determine whether a server is about to have a generation 2 collection, and whether rerouting requests to another server could ease any problems with pauses.

PERFORMANCE CHECKS

Determine the length of time in a garbage collection.

Determine what caused a garbage collection.

Issue: Generation 0 Is Too Big

Generation 0 is likely to have a larger number of objects on a 64-bit system, especially when you use server garbage collection instead of workstation garbage collection. This is because the threshold to trigger a generation 0 garbage collection is higher in these environments, and generation 0 collections can get much bigger. Performance is improved when an application allocates more memory before a garbage collection is triggered.

Issue: CPU Usage During a Garbage Collection Is Too High

CPU usage will be high during a garbage collection. If a significant amount of process time is spent in a garbage collection, the number of collections is too frequent or the collection is lasting too long. An increased allocation rate of objects on the managed heap causes garbage collection to occur more frequently. Decreasing the allocation rate reduces the frequency of garbage collections.

You can monitor allocation rates by using the `Allocated Bytes/second` performance counter. For more information, see [Performance Counters in .NET](#).

The duration of a collection is primarily a factor of the number of objects that survive after allocation. The garbage collector must go through a large amount of memory if many objects remain to be collected. The work to compact the survivors is time-consuming. To determine how many objects were handled during a collection, set a breakpoint in the debugger at the end of a garbage collection for a specified generation.

PERFORMANCE CHECKS

Determine if high CPU usage is caused by garbage collection.

Set a breakpoint at the end of garbage collection.

Troubleshooting guidelines

This section describes guidelines that you should consider as you begin your investigations.

Workstation or Server Garbage Collection

Determine if you are using the correct type of garbage collection. If your application uses multiple threads and object instances, use server garbage collection instead of workstation garbage collection. Server garbage collection operates on multiple threads, whereas workstation garbage collection requires multiple instances of an application to run their own garbage collection threads and compete for CPU time.

An application that has a low load and that performs tasks infrequently in the background, such as a service,

could use workstation garbage collection with concurrent garbage collection disabled.

When to Measure the Managed Heap Size

Unless you are using a profiler, you will have to establish a consistent measuring pattern to effectively diagnose performance issues. Consider the following points to establish a schedule:

- If you measure after a generation 2 garbage collection, the entire managed heap will be free of garbage (dead objects).
- If you measure immediately after a generation 0 garbage collection, the objects in generations 1 and 2 will not be collected yet.
- If you measure immediately before a garbage collection, you will measure as much allocation as possible before the garbage collection starts.
- Measuring during a garbage collection is problematic, because the garbage collector data structures are not in a valid state for traversal and may not be able to give you the complete results. This is by design.
- When you are using workstation garbage collection with concurrent garbage collection, the reclaimed objects are not compacted, so the heap size can be the same or larger (fragmentation can make it appear to be larger).
- Concurrent garbage collection on generation 2 is delayed when the physical memory load is too high.

The following procedure describes how to set a breakpoint so that you can measure the managed heap.

To set a breakpoint at the end of garbage collection

- In WinDbg with the SOS debugger extension loaded, enter the following command:

```
bp mscorewks!WKS::GCHeap::RestartEE "j (dwo(mscorwks!WKS::GCHeap::GcCondemnedGeneration)==2) 'kb';'g'"
```

Set `GcCondemnedGeneration` to the desired generation. This command requires private symbols.

This command forces a break if `RestartEE` is executed after generation 2 objects have been reclaimed for garbage collection.

In server garbage collection, only one thread calls `RestartEE`, so the breakpoint will occur only once during a generation 2 garbage collection.

Performance check procedures

This section describes the following procedures to isolate the cause of your performance issue:

- Determine whether the problem is caused by garbage collection.
- Determine whether the out-of-memory exception is managed.
- Determine how much virtual memory can be reserved.
- Determine whether there is enough physical memory.
- Determine how much memory the managed heap is committing.
- Determine how much memory the managed heap reserves.
- Determine large objects in generation 2.
- Determine references to objects.
- Determine whether a finalizer has been run.
- Determine whether there are objects waiting to be finalized.
- Determine the amount of free space in the managed heap.
- Determine the number of pinned objects.
- Determine the length of time in a garbage collection.
- Determine what triggered a garbage collection.
- Determine whether high CPU usage is caused by garbage collection.

To determine whether the problem is caused by garbage collection

- Examine the following two memory performance counters:
 - **% Time in GC.** Displays the percentage of elapsed time that was spent performing a garbage collection after the last garbage collection cycle. Use this counter to determine whether the garbage collector is spending too much time to make managed heap space available. If the time spent in garbage collection is relatively low, that could indicate a resource problem outside the managed heap. This counter may not be accurate when concurrent or background garbage collection is involved.
 - **# Total committed Bytes.** Displays the amount of virtual memory currently committed by the garbage collector. Use this counter to determine whether the memory consumed by the garbage collector is an excessive portion of the memory that your application uses.

Most of the memory performance counters are updated at the end of each garbage collection. Therefore, they may not reflect the current conditions that you want information about.

To determine whether the out-of-memory exception is managed

1. In the WinDbg or Visual Studio debugger with the SOS debugger extension loaded, enter the print exception (`!pe`) command:

```
!pe
```

If the exception is managed, `OutOfMemoryException` is displayed as the exception type, as shown in the following example.

```
Exception object: 39594518
Exception type: System.OutOfMemoryException
Message: <none>
InnerException: <none>
StackTrace (generated):
```

2. If the output does not specify an exception, you have to determine which thread the out-of-memory exception is from. Enter the following command in the debugger to show all the threads with their call stacks:

```
~\*kb
```

The thread with the stack that has exception calls is indicated by the `RaiseTheException` argument. This is the managed exception object.

```
28adfb44 7923918f 5b61f2b4 00000000 5b61f2b4 mscorwks!RaiseTheException+0xa0
```

3. You can use the following command to dump nested exceptions.

```
!pe -nested
```

If you do not find any exceptions, the out-of-memory exception originated from unmanaged code.

To determine how much virtual memory can be reserved

- In WinDbg with the SOS debugger extension loaded, enter the following command to get the largest free region:

```
!address -summary
```

The largest free region is displayed as shown in the following output.

```
Largest free region: Base 54000000 - Size 0003A980
```

In this example, the size of the largest free region is approximately 24000 KB (3A980 in hexadecimal). This region is much smaller than what the garbage collector needs for a segment.

-or-

- Use the `vmstat` command:

```
!vmstat
```

The largest free region is the largest value in the MAXIMUM column, as shown in the following output.

TYPE	MINIMUM	MAXIMUM	AVERAGE	BLK COUNT	TOTAL
~~~~~	~~~~~	~~~~~	~~~~~	~~~~~	~~~~
<b>Free:</b>					
Small	8K	64K	46K	36	1,671K
Medium	80K	864K	349K	3	1,047K
Large	1,384K	1,278,848K	151,834K	12	1,822,015K
Summary	8K	1,278,848K	35,779K	51	1,824,735K

### To determine whether there is enough physical memory

1. Start Windows Task Manager.
2. On the `Performance` tab, look at the committed value. (In Windows 7, look at `Commit (KB)` in the `System group`.)

If the `Total` is close to the `Limit`, you are running low on physical memory.

### To determine how much memory the managed heap is committing

- Use the `# Total committed bytes` memory performance counter to get the number of bytes that the managed heap is committing. The garbage collector commits chunks on a segment as needed, not all at the same time.

#### NOTE

Do not use the `# Bytes in all Heaps` performance counter, because it does not represent actual memory usage by the managed heap. The size of a generation is included in this value and is actually its threshold size, that is, the size that induces a garbage collection if the generation is filled with objects. Therefore, this value is usually zero.

### To determine how much memory the managed heap reserves

- Use the `# Total reserved bytes` memory performance counter.

The garbage collector reserves memory in segments, and you can determine where a segment starts by using the `eeheap` command.

#### IMPORTANT

Although you can determine the amount of memory the garbage collector allocates for each segment, segment size is implementation-specific and is subject to change at any time, including in periodic updates. Your app should never make assumptions about or depend on a particular segment size, nor should it attempt to configure the amount of memory available for segment allocations.

- In the WinDbg or Visual Studio debugger with the SOS debugger extension loaded, enter the following command:

```
!eeheap -gc
```

The result is as follows.

```
Number of GC Heaps: 2
-----
Heap 0 (002db550)
generation 0 starts at 0x02abe29c
generation 1 starts at 0x02abdd08
generation 2 starts at 0x02ab0038
ephemeral segment allocation context: none
    segment      begin allocated      size
02ab0000 02ab0038 02aceff4 0x0001efbc(126908)
Large object heap starts at 0x0aab0038
    segment      begin allocated      size
0aab0000 0aab0038 0aab2278 0x00002240(8768)
Heap Size 0x211fc(135676)
-----
Heap 1 (002dc958)
generation 0 starts at 0x06ab1bd8
generation 1 starts at 0x06ab1bcc
generation 2 starts at 0x06ab0038
ephemeral segment allocation context: none
    segment      begin allocated      size
06ab0000 06ab0038 06ab3be4 0x00003bac(15276)
Large object heap starts at 0x0cab0038
    segment      begin allocated      size
0cab0000 0cab0038 0cab0048 0x00000010(16)
Heap Size 0x3bbc(15292)
-----
GC Heap Size 0x24db8(150968)
```

The addresses indicated by "segment" are the starting addresses of the segments.

### To determine large objects in generation 2

- In the WinDbg or Visual Studio debugger with the SOS debugger extension loaded, enter the following command:

```
!dumpheap -stat
```

If the managed heap is big, `dumpheap` may take a while to finish.

You can start analyzing from the last few lines of the output, because they list the objects that use the most space. For example:

```
2c6108d4 173712     14591808 DevExpress.XtraGrid.Views.Grid.ViewInfo.GridCellInfo
00155f80      533     15216804     Free
7a747c78 791070     15821400 System.Collections.Specialized.ListDictionary+DictionaryNode
7a747bac 700930     19626040 System.Collections.Specialized.ListDictionary
2c64e36c 78644     20762016 DevExpress.XtraEditors.ViewInfo.TextEditViewInfo
79124228 121143     29064120 System.Object[]
035f0ee4 81626     35588936 Toolkit.TlkOrder
00fcac40 6193      44911636 WaveBasedStrategy.Tick_Snap[]
791242ec 40182      90664128 System.Collections.Hashtable+bucket[]
790fa3e0 3154024    137881448 System.String
Total 8454945 objects
```

The last object listed is a string and occupies the most space. You can examine your application to see

how your string objects can be optimized. To see strings that are between 150 and 200 bytes, enter the following:

```
!dumpheap -type System.String -min 150 -max 200
```

An example of the results is as follows.

```
Address MT      Size  Gen
1875d2c0 790fa3e0    152    2 System.String HighlightNullStyle_Blotter_PendingOrder-
11_Blotter_PendingOrder-11
...
...
```

Using an integer instead of a string for an ID can be more efficient. If the same string is being repeated thousands of times, consider string interning. For more information about string interning, see the reference topic for the [String.Intern](#) method.

### To determine references to objects

- In WinDbg with the SOS debugger extension loaded, enter the following command to list references to objects:

```
!gcroot
```

-or-

- To determine the references for a specific object, include the address:

```
!gcroot 1c37b2ac
```

Roots found on stacks may be false positives. For more information, use the command [!help gcroot](#).

```
ebx:Root:19011c5c(System.Windows.Forms.Application+ThreadContext)->
19010b78(DemoApp.FormDemoApp)->
19011158(System.Windows.Forms.PropertyStore)->
... [omitted]
1c3745ec(System.Data.DataTable)->
1c3747a8(System.Data.DataColumnCollection)->
1c3747f8(System.Collections.Hashtable)->
1c376590(System.Collections.Hashtable+bucket[])->
1c376c98(System.Data.DataColumn)->
1c37b270(System.Data.Common.DoubleStorage)->
1c37b2ac(System.Double[])
Scan Thread 0 OSThread 99c
Scan Thread 6 OSThread 484
```

The `gcroot` command can take a long time to finish. Any object that is not reclaimed by garbage collection is a live object. This means that some root is directly or indirectly holding onto the object, so `gcroot` should return path information to the object. You should examine the graphs returned and see why these objects are still referenced.

### To determine whether a finalizer has been run

- Run a test program that contains the following code:

```
GC.Collect();
GC.WaitForPendingFinalizers();
GC.Collect();
```

If the test resolves the problem, this means that the garbage collector was not reclaiming objects, because

the finalizers for those objects had been suspended. The [GC.WaitForPendingFinalizers](#) method enables the finalizers to complete their tasks, and fixes the problem.

### To determine whether there are objects waiting to be finalized

1. In the WinDbg or Visual Studio debugger with the SOS debugger extension loaded, enter the following command:

```
!finalizequeue
```

Look at the number of objects that are ready for finalization. If the number is high, you must examine why these finalizers cannot progress at all or cannot progress fast enough.

2. To get an output of threads, enter the following command:

```
!threads -special
```

This command provides output such as the following.

OSID	Special thread type
2 cd0	DbgHelper
3 c18	Finalizer
4 df0	GC SuspendEE

The finalizer thread indicates which finalizer, if any, is currently being run. When a finalizer thread is not running any finalizers, it is waiting for an event to tell it to do its work. Most of the time you will see the finalizer thread in this state because it runs at THREAD_HIGHEST_PRIORITY and is supposed to finish running finalizers, if any, very quickly.

### To determine the amount of free space in the managed heap

- In the WinDbg or Visual Studio debugger with the SOS debugger extension loaded, enter the following command:

```
!dumpheap -type Free -stat
```

This command displays the total size of all the free objects on the managed heap, as shown in the following example.

total 230 objects
Statistics:
MT Count TotalSize Class Name
00152b18 230 40958584 Free
Total 230 objects

- To determine the free space in generation 0, enter the following command for memory consumption information by generation:

```
!eeheap -gc
```

This command displays output similar to the following. The last line shows the ephemeral segment.

```
Heap 0 (0015ad08)
generation 0 starts at 0x49521f8c
generation 1 starts at 0x494d7f64
generation 2 starts at 0x007f0038
ephemeral segment allocation context: none
segment begin allocated size
00178250 7a80d84c 7a82f1cc 0x00021980(137600)
00161918 78c50e40 78c7056c 0x0001f72c(128812)
007f0000 007f0038 047eed28 0x03ffecf0(67103984)
3a120000 3a120038 3a3e84f8 0x002c84c0(2917568)
46120000 46120038 49e05d04 0x03ce5ccc(63855820)
```

- Calculate the space used by generation 0:

```
? 49e05d04-0x49521f8c
```

The result is as follows. Generation 0 is approximately 9 MB.

```
Evaluate expression: 9321848 = 008e3d78
```

- The following command dumps the free space within the generation 0 range:

```
!dumpheap -type Free -stat 0x49521f8c 49e05d04
```

The result is as follows.

```
-----
Heap 0
total 409 objects
-----
Heap 1
total 0 objects
-----
Heap 2
total 0 objects
-----
Heap 3
total 0 objects
-----
total 409 objects
Statistics:
      MT      Count TotalSize Class Name
0015a498        409    7296540     Free
Total 409 objects
```

This output shows that the generation 0 portion of the heap is using 9 MB of space for objects and has 7 MB free. This analysis shows the extent to which generation 0 contributes to fragmentation. This amount of heap usage should be discounted from the total amount as the cause of fragmentation by long-term objects.

### To determine the number of pinned objects

- In the WinDbg or Visual Studio debugger with the SOS debugger extension loaded, enter the following command:

```
!ghandles
```

The statistics displayed includes the number of pinned handles, as the following example shows.

```

GC Handle Statistics:
Strong Handles:      29
Pinned Handles:     10

```

## To determine the length of time in a garbage collection

- Examine the `% Time in GC` memory performance counter.

The value is calculated by using a sample interval time. Because the counters are updated at the end of each garbage collection, the current sample will have the same value as the previous sample if no collections occurred during the interval.

Collection time is obtained by multiplying the sample interval time with the percentage value.

The following data shows four sampling intervals of two seconds, for an 8-second study. The `Gen0`, `Gen1`, and `Gen2` columns show the total number of garbage collections that have completed by the end of the interval for that generation.

Interval	Gen0	Gen1	Gen2	% Time in GC
1	9	3	1	10
2	10	3	1	1
3	11	3	1	3
4	11	3	1	3

This information does not show when the garbage collection occurred, but you can determine the number of garbage collections that occurred in an interval of time. Assuming the worst case, the tenth generation 0 garbage collection finished at the start of the second interval, and the eleventh generation 0 garbage collection finished at the end of the third interval. The time between the end of the tenth and the end of the eleventh garbage collection is about 2 seconds, and the performance counter shows 3%, so the duration of the eleventh generation 0 garbage collection was  $(2 \text{ seconds} * 3\%) = 60\text{ms}$ .

In the next example, there are five intervals.

Interval	Gen0	Gen1	Gen2	% Time in GC
1	9	3	1	3
2	10	3	1	1
3	11	4	1	1
4	11	4	1	1
5	11	4	2	20

The second generation 2 garbage collection started during the fourth interval and finished at the fifth interval. Assuming the worst case, the last garbage collection was for a generation 0 collection that finished at the start of the third interval, and the generation 2 garbage collection finished at the end of the fifth interval. Therefore, the time between the end of the generation 0 garbage collection and the end of the generation 2 garbage collection is 4 seconds. Because the `% Time in GC` counter is 20%, the maximum amount of time the generation 2 garbage collection could have taken is  $(4 \text{ seconds} * 20\%) = 800\text{ms}$ .

- Alternatively, you can determine the length of a garbage collection by using [garbage collection ETW events](#), and analyze the information to determine the duration of garbage collection.

For example, the following data shows an event sequence that occurred during a non-concurrent garbage collection.

Timestamp	Event name
513052	GCSuspendEEBegin_V1
513078	GCSuspendEEEnd
513090	GCStart_V1
517890	GCEnd_V1
517894	GCHeapStats
517897	GCRestartEEBegin
517918	GCRestartEEEnd

Suspending the managed thread took 26us ( GCSuspendEEEnd – GCSuspendEEBegin_V1 ).

The actual garbage collection took 4.8ms ( GCEnd_V1 – GCStart_V1 ).

Resuming the managed threads took 21us ( GCRestartEEEnd – GCRestartEEBegin ).

The following output provides an example for background garbage collection, and includes the process, thread, and event fields. (Not all data is shown.)

timestamp(us)	event name	process	thread	event field
42504385	GCSuspendEEBegin_V1	Test.exe	4372	1
42504648	GCSuspendEEEnd	Test.exe	4372	
42504816	GCStart_V1	Test.exe	4372	102019
42504907	GCStart_V1	Test.exe	4372	102020
42514170	GCEnd_V1	Test.exe	4372	
42514204	GCHeapStats	Test.exe	4372	102020
42832052	GCRestartEEBegin	Test.exe	4372	
42832136	GCRestartEEEnd	Test.exe	4372	
63685394	GCSuspendEEBegin_V1	Test.exe	4744	6
63686347	GCSuspendEEEnd	Test.exe	4744	
63784294	GCRestartEEBegin	Test.exe	4744	
63784407	GCRestartEEEnd	Test.exe	4744	
89931423	GCEnd_V1	Test.exe	4372	102019
89931464	GCHeapStats	Test.exe	4372	

The GCStart_V1 event at 42504816 indicates that this is a background garbage collection, because the last field is 1. This becomes garbage collection No. 102019.

The GCStart event occurs because there is a need for an ephemeral garbage collection before you start a background garbage collection. This becomes garbage collection No. 102020.

At 42514170, garbage collection No. 102020 finishes. The managed threads are restarted at this point. This is completed on thread 4372, which triggered this background garbage collection.

On thread 4744, a suspension occurs. This is the only time at which the background garbage collection has to suspend managed threads. This duration is approximately 99ms ((63784407-63685394)/1000).

The GCEnd event for the background garbage collection is at 89931423. This means that the background garbage collection lasted for about 47seconds ((89931423-42504816)/1000).

While the managed threads are running, you can see any number of ephemeral garbage collections occurring.

### To determine what triggered a garbage collection

- In the WinDbg or Visual Studio debugger with the SOS debugger extension loaded, enter the following command to show all the threads with their call stacks:

```
~*kb
```

This command displays output similar to the following.

```
0012f3b0 79ff0bf8 mscorewks!WKS::GCHeap::GarbageCollect
0012f454 30002894 mscorewks!GCInterface::CollectGeneration+0xa4
0012f490 79fa22bd fragment_ni!request.Main(System.String[])+0x48
```

If the garbage collection was caused by a low memory notification from the operating system, the call stack is similar, except that the thread is the finalizer thread. The finalizer thread gets an asynchronous low memory notification and induces the garbage collection.

If the garbage collection was caused by memory allocation, the stack appears as follows:

```
0012f230 7a07c551 mscorewks!WKS::GCHeap::GarbageCollectGeneration
0012f2b8 7a07cba8 mscorewks!WKS::gc_heap::try_allocate_more_space+0x1a1
0012f2d4 7a07cef8 mscorewks!WKS::gc_heap::allocate_more_space+0x18
0012f2f4 7a02a51b mscorewks!WKS::GCHeap::Alloc+0x4b
0012f310 7a02ae4c mscorewks!Alloc+0x60
0012f364 7a030e46 mscorewks!FastAllocatePrimitiveArray+0xbd
0012f424 300027f4 mscorewks!JIT_NewArr1+0x148
000af70f 3000299f fragment_ni!request..ctor(Int32, Single)+0x20c
0000002a 79fa22bd fragment_ni!request.Main(System.String[])+0x153
```

A just-in-time helper (`JIT_New*`) eventually calls `GCHeap::GarbageCollectGeneration`. If you determine that generation 2 garbage collections are caused by allocations, you must determine which objects are collected by a generation 2 garbage collection and how to avoid them. That is, you want to determine the difference between the start and the end of a generation 2 garbage collection, and the objects that caused the generation 2 collection.

For example, enter the following command in the debugger to show the beginning of a generation 2 collection:

```
!dumpheap -stat
```

Example output (abridged to show the objects that use the most space):

```
79124228      31857      9862328 System.Object[]
035f0384      25668      11601936 Toolkit.TlkPosition
00155f80      21248      12256296   Free
79103b6c      297003     13068132 System.Threading.ReaderWriterLock
7a747ad4      708732     14174640 System.Collections.Specialized.HybridDictionary
7a747c78      786498     15729960 System.Collections.Specialized.ListDictionary+DictionaryNode
7a747bac      700298     19608344 System.Collections.Specialized.ListDictionary
035f0ee4      89192      38887712 Toolkit.TlkOrder
00fc当地40      6193      44911636 WaveBasedStrategy.Tick_Snap[]
7912c444      91616      71887080 System.Double[]
791242ec      32451      82462728 System.Collections.Hashtable+bucket[]
790fa3e0      2459154    112128436 System.String
Total 6471774 objects
```

Repeat the command at the end of generation 2:

```
!dumpheap -stat
```

Example output (abridged to show the objects that use the most space):

79124228	26648	9314256 System.Object[]
035f0384	25668	11601936 Toolkit.TlkPosition
79103b6c	296770	13057880 System.Threading.ReaderWriterLock
7a747ad4	708730	14174600 System.Collections.Specialized.HybridDictionary
7a747c78	786497	15729940 System.Collections.Specialized.ListDictionary+DictionaryNode
7a747bac	700298	19608344 System.Collections.Specialized.ListDictionary
00155f80	13806	34007212 Free
035f0ee4	89187	38885532 Toolkit.TlkOrder
00fcac40	6193	44911636 WaveBasedStrategy.Tick_Snap[]
791242ec	32370	82359768 System.Collections.Hashtable+bucket[]
790fa3e0	2440020	111341808 System.String
Total 6417525 objects		

The `double[]` objects disappeared from the end of the output, which means that they were collected. These objects account for approximately 70 MB. The remaining objects did not change much. Therefore, these `double[]` objects were the reason why this generation 2 garbage collection occurred. Your next step is to determine why the `double[]` objects are there and why they died. You can ask the code developer where these objects came from, or you can use the `gcroot` command.

### To determine whether high CPU usage is caused by garbage collection

- Correlate the `% Time in GC` memory performance counter value with the process time.

If the `% Time in GC` value spikes at the same time as process time, garbage collection is causing a high CPU usage. Otherwise, profile the application to find where the high usage is occurring.

## See also

- [Garbage Collection](#)

# Induced Collections

9/20/2022 • 2 minutes to read • [Edit Online](#)

In most cases, the garbage collector can determine the best time to perform a collection, and you should let it run independently. There are rare situations when a forced collection might improve your application's performance. In these cases, you can induce garbage collection by using the [GC.Collect](#) method to force a garbage collection.

Use the [GC.Collect](#) method when there is a significant reduction in the amount of memory being used at a specific point in your application's code. For example, if your application uses a complex dialog box that has several controls, calling [Collect](#) when the dialog box is closed could improve performance by immediately reclaiming the memory used by the dialog box. Be sure that your application is not inducing garbage collection too frequently, because that can decrease performance if the garbage collector is trying to reclaim objects at non-optimal times. You can supply a [GCCollectionMode.Optimized](#) enumeration value to the [Collect](#) method to collect only when collection would be productive, as discussed in the next section.

## GC collection mode

You can use one of the [GC.Collect](#) method overloads that includes a [GCCollectionMode](#) value to specify the behavior for a forced collection as follows.

GCCOLLECTIONMODE VALUE	DESCRIPTION
Default	Uses the default garbage collection setting for the running version of .NET.
Forced	Forces garbage collection to occur immediately. This is equivalent to calling the <a href="#">GC.Collect()</a> overload. It results in a full blocking collection of all generations.  You can also compact the large object heap by setting the <a href="#">GCSettings.LargeObjectHeapCompactionMode</a> property to <a href="#">GCLargeObjectHeapCompactionMode.CompactOnce</a> before forcing an immediate full blocking garbage collection.
Optimized	Enables the garbage collector to determine whether the current time is optimal to reclaim objects.  The garbage collector could determine that a collection would not be productive enough to be justified, in which case it will return without reclaiming objects.

## Background or blocking collections

You can call the [GC.Collect\(Int32, GCCollectionMode, Boolean\)](#) method overload to specify whether an induced collection is blocking or not. The type of collection performed depends on a combination of the method's `mode` and `blocking` parameters. `mode` is a member of the [GCCollectionMode](#) enumeration, and `blocking` is a [Boolean](#) value. The following table summarizes the interaction of the `mode` and `blocking` arguments.

MODE	BLOCKING = TRUE	BLOCKING = FALSE
Forced or Default	A blocking collection is performed as soon as possible. If a background collection is in progress and generation is 0 or 1, the <a href="#">Collect(Int32, GCCollectionMode, Boolean)</a> method immediately triggers a blocking collection and returns when the collection is finished. If a background collection is in progress and the <code>generation</code> parameter is 2, the method waits until the background collection is finished, triggers a blocking generation 2 collection, and then returns.	A collection is performed as soon as possible. The <a href="#">Collect(Int32, GCCollectionMode, Boolean)</a> method requests a background collection, but this is not guaranteed; depending on the circumstances, a blocking collection may still be performed. If a background collection is already in progress, the method returns immediately.
Optimized	A blocking collection may be performed, depending on the state of the garbage collector and the <code>generation</code> parameter. The garbage collector tries to provide optimal performance.	A collection may be performed, depending on the state of the garbage collector. The <a href="#">Collect(Int32, GCCollectionMode, Boolean)</a> method requests a background collection, but this is not guaranteed; depending on the circumstances, a blocking collection may still be performed. The garbage collector tries to provide optimal performance. If a background collection is already in progress, the method returns immediately.

## See also

- [Latency Modes](#)
- [Garbage Collection](#)

# Latency modes

9/20/2022 • 2 minutes to read • [Edit Online](#)

To reclaim objects, the garbage collector (GC) must stop all the executing threads in an application. The period of time during which the garbage collector is active is referred to as its *latency*.

In some situations, such as when an application retrieves data or displays content, a full garbage collection can occur at a critical time and impede performance. You can adjust the intrusiveness of the garbage collector by setting the `GCSettings.LatencyMode` property to one of the `System.Runtime.GCLatencyMode` values.

## Low latency settings

Using a "low" latency setting means the garbage collector intrudes less in your application. Garbage collection is more conservative about reclaiming memory.

The `System.Runtime.GCLatencyMode` enumeration provides two low latency settings:

- `GCLatencyMode.LowLatency` suppresses generation 2 collections and performs only generation 0 and 1 collections. It can be used only for short periods of time. Over longer periods, if the system is under memory pressure, the garbage collector will trigger a collection, which can briefly pause the application and disrupt a time-critical operation. This setting is available only for workstation garbage collection.
- `GCLatencyMode.SustainedLowLatency` suppresses foreground generation 2 collections and performs only generation 0, 1, and background generation 2 collections. It can be used for longer periods of time, and is available for both workstation and server garbage collection. This setting cannot be used if background garbage collection is disabled.

During low latency periods, generation 2 collections are suppressed unless the following occurs:

- The system receives a low memory notification from the operating system.
- Application code induces a collection by calling the `GC.Collect` method and specifying 2 for the `generation` parameter.

## Scenarios

The following table lists the application scenarios for using the `GCLatencyMode` values:

LATENCY MODE	APPLICATION SCENARIOS
Batch	<p>For applications that have no user interface (UI) or server-side operations.</p> <p>When background garbage collection is disabled, this is the default mode for workstation and server garbage collection. <code>Batch</code> mode also overrides the <code>gcConcurrent</code> setting, that is, it prevents background or concurrent collections.</p>
Interactive	<p>For most applications that have a UI.</p> <p>This is the default mode for workstation and server garbage collection. However, if an app is hosted, the garbage collector settings of the hosting process take precedence.</p>

LATENCY MODE	APPLICATION SCENARIOS
<a href="#">LowLatency</a>	For applications that have short-term, time-sensitive operations during which interruptions from the garbage collector could be disruptive. For example, applications that render animations or data acquisition functions.
<a href="#">SustainedLowLatency</a>	<p>For applications that have time-sensitive operations for a contained but potentially longer duration of time during which interruptions from the garbage collector could be disruptive. For example, applications that need quick response times as market data changes during trading hours.</p> <p>This mode results in a larger managed heap size than other modes. Because it does not compact the managed heap, higher fragmentation is possible. Ensure that sufficient memory is available.</p>

## Guidelines for using low latency

When you use [GCLatencyMode.LowLatency](#) mode, consider the following guidelines:

- Keep the period of time in low latency as short as possible.
- Avoid allocating high amounts of memory during low latency periods. Low memory notifications can occur because garbage collection reclaims fewer objects.
- While in the low latency mode, minimize the number of new allocations, in particular allocations onto the large object heap and pinned objects.
- Be aware of threads that could be allocating. Because the [LatencyMode](#) property setting is process-wide, [OutOfMemoryException](#) exceptions can be generated on any thread that is allocating.
- Wrap the low latency code in constrained execution regions. For more information, see [Constrained execution regions](#).
- You can force generation 2 collections during a low latency period by calling the [GC.Collect\(Int32, GCCollectionMode\)](#) method.

## See also

- [System.GC](#)
- [Induced Collections](#)
- [Garbage Collection](#)

# Optimization for Shared Web Hosting

9/20/2022 • 2 minutes to read • [Edit Online](#)

If you are the administrator for a server that is shared by hosting several small Web sites, you can optimize performance and increase site capacity by adding the following `gcTrimCommitOnLowMemory` setting to the `runtime` node in the `Aspnet.config` file in the .NET directory:

```
<gcTrimCommitOnLowMemory enabled="true|false"/>
```

## NOTE

This setting is recommended only for shared Web hosting scenarios.

Because the garbage collector retains memory for future allocations, its committed space can be more than what is strictly needed. You can reduce this space to accommodate times when there is a heavy load on system memory. Reducing this committed space improves performance and expands the capacity to host more sites.

When the `gcTrimCommitOnLowMemory` setting is enabled, the garbage collector evaluates the system memory load and enters a trimming mode when the load reaches 90%. It maintains the trimming mode until the load drops under 85%.

When conditions permit, the garbage collector can decide that the `gcTrimCommitOnLowMemory` setting will not help the current application and ignore it.

## Example

The following XML fragment shows how to enable the `gcTrimCommitOnLowMemory` setting. Ellipses indicate other settings that would be in the `runtime` node.

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <runtime>
    ...
    <gcTrimCommitOnLowMemory enabled="true"/>
  </runtime>
  ...
</configuration>
```

## See also

- [Garbage Collection](#)

# Garbage Collection Notifications

9/20/2022 • 25 minutes to read • [Edit Online](#)

There are situations in which a full garbage collection (that is, a generation 2 collection) by the common language runtime may adversely affect performance. This can be an issue particularly with servers that process large volumes of requests; in this case, a long garbage collection can cause a request time-out. To prevent a full collection from occurring during a critical period, you can be notified that a full garbage collection is approaching and then take action to redirect the workload to another server instance. You can also induce a collection yourself, provided that the current server instance does not need to process requests.

The [RegisterForFullGCNotification](#) method registers for a notification to be raised when the runtime senses that a full garbage collection is approaching. There are two parts to this notification: when the full garbage collection is approaching and when the full garbage collection has completed.

## WARNING

When the `<gcConcurrent>` configuration element is enabled, [WaitForFullGCCComplete](#) may return `NotApplicable` [GCNotificationStatus](#) if the full GC was done as a background GC.

To determine when a notification has been raised, use the [WaitForFullGCAccroach](#) and [WaitForFullGCCComplete](#) methods. Typically, you use these methods in a `while` loop to continually obtain a [GCNotificationStatus](#) enumeration that shows the status of the notification. If that value is [Succeeded](#), you can do the following:

- In response to a notification obtained with the [WaitForFullGCAccroach](#) method, you can redirect the workload and possibly induce a collection yourself.
- In response to a notification obtained with the [WaitForFullGCCComplete](#) method, you can make the current server instance available to process requests again. You can also gather information. For example, you can use the [CollectionCount](#) method to record the number of collections.

The [WaitForFullGCAccroach](#) and the [WaitForFullGCCComplete](#) methods are designed to work together. Using one without the other can produce indeterminate results.

## Full Garbage Collection

The runtime causes a full garbage collection when any of the following scenarios are true:

- Enough memory has been promoted into generation 2 to cause the next generation 2 collection.
- Enough memory has been promoted into the large object heap to cause the next generation 2 collection.
- A collection of generation 1 is escalated to a collection of generation 2 due to other factors.

The thresholds you specify in the [RegisterForFullGCNotification](#) method apply to the first two scenarios. However, in the first scenario you will not always receive the notification at the time proportional to the threshold values you specify for two reasons:

- The runtime does not check each small object allocation (for performance reasons).
- Only generation 1 collections promote memory into generation 2.

The third scenario also contributes to the uncertainty of when you will receive the notification. Although this is not a guarantee, it does prove to be a useful way to mitigate the effects of an inopportune full garbage

collection by redirecting the requests during this time or inducing the collection yourself when it can be better accommodated.

## Notification Threshold Parameters

The [RegisterForFullGCNotification](#) method has two parameters to specify the threshold values of the generation 2 objects and the large object heap. When those values are met, a garbage collection notification should be raised. The following table describes these parameters.

PARAMETER	DESCRIPTION
<code>maxGenerationThreshold</code>	A number between 1 and 99 that specifies when the notification should be raised based on the objects promoted in generation 2.
<code>largeObjectHeapThreshold</code>	A number between 1 and 99 that specifies when the notification should be raised based on the objects that are allocated in the large object heap.

If you specify a value that is too high, there is a high probability that you will receive a notification, but it could be too long a period to wait before the runtime causes a collection. If you induce a collection yourself, you may reclaim more objects than would be reclaimed if the runtime causes the collection.

If you specify a value that is too low, the runtime may cause the collection before you have had sufficient time to be notified.

## Example

### Description

In the following example, a group of servers service incoming Web requests. To simulate the workload of processing requests, byte arrays are added to a `List<T>` collection. Each server registers for a garbage collection notification and then starts a thread on the `WaitForFullGCProm` user method to continuously monitor the `GCNotificationStatus` enumeration that is returned by the `WaitForFullGCAccroach` and the `WaitForFullGCCComplete` methods.

The `WaitForFullGCAccroach` and the `WaitForFullGCCComplete` methods call their respective event-handling user methods when a notification is raised:

- `OnFullGCAccroachNotify`

This method calls the `RedirectRequests` user method, which instructs the request queuing server to suspend sending requests to the server. This is simulated by setting the class-level variable `bAllocate` to `false` so that no more objects are allocated.

Next, the `FinishExistingRequests` user method is called to finish processing the pending server requests. This is simulated by clearing the `List<T>` collection.

Finally, a garbage collection is induced because the workload is light.

- `OnFullGCCCompleteNotify`

This method calls the user method `AcceptRequests` to resume accepting requests because the server is no longer susceptible to a full garbage collection. This action is simulated by setting the `bAllocate` variable to `true` so that objects can resume being added to the `List<T>` collection.

The following code contains the `Main` method of the example.

```

using namespace System;
using namespace System::Collections::Generic;
using namespace System::Threading;

namespace GCNotify
{
    ref class Program
    {
private:
    // Variable for continual checking in the
    // While loop in the WaitForFullGCPoc method.
    static bool checkForNotify = false;

    // Variable for suspending work
    // (such servicing allocated server requests)
    // after a notification is received and then
    // resuming allocation after inducing a garbage collection.
    static bool bAllocate = false;

    // Variable for ending the example.
    static bool finalExit = false;

    // Collection for objects that
    // simulate the server request workload.
    static List<array<Byte>>>^ load = gcnew List<array<Byte>>();
}

public:
    static void Main()
    {
        try
        {
            // Register for a notification.
            GC::RegisterForFullGCNotification(10, 10);
            Console::WriteLine("Registered for GC notification.");

            checkForNotify = true;
            bAllocate = true;

            // Start a thread using WaitForFullGCPoc.
            Thread^ thWaitForFullGC = gcnew Thread(gcnew ThreadStart(&WaitForFullGCPoc));
            thWaitForFullGC->Start();

            // While the thread is checking for notifications in
            // WaitForFullGCPoc, create objects to simulate a server workload.
            try
            {
                int lastCollCount = 0;
                int newCollCount = 0;

                while (true)
                {
                    if (bAllocate)
                    {
                        load->Add(gcnew array<Byte>(1000));
                        newCollCount = GC::CollectionCount(2);
                        if (newCollCount != lastCollCount)
                        {
                            // Show collection count when it increases:
                            Console::WriteLine("Gen 2 collection count: {0}",
GC::CollectionCount(2).ToString());
                            lastCollCount = newCollCount;
                        }
                    }

                    // For ending the example (arbitrary).
                    if (newCollCount == 500)
                    {

```

```

        finalExit = true;
        checkForNotify = false;
        break;
    }
}
}

}

catch (OutOfMemoryException^)
{
    Console::WriteLine("Out of memory.");
}

finalExit = true;
checkForNotify = false;
GC::CancelFullGCNotification();

}

catch (InvalidOperationException^ invalidOp)
{
    Console::WriteLine("GC Notifications are not supported while concurrent GC is enabled.\n"
        + invalidOp->Message);
}

public:
static void OnFullGCApproachNotify()
{
    Console::WriteLine("Redirecting requests.");

    // Method that tells the request queuing
    // server to not direct requests to this server.
    RedirectRequests();

    // Method that provides time to
    // finish processing pending requests.
    FinishExistingRequests();

    // This is a good time to induce a GC collection
    // because the runtime will induce a full GC soon.
    // To be very careful, you can check precede with a
    // check of the GC.GCCollectionCount to make sure
    // a full GC did not already occur since last notified.
    GC::Collect();
    Console::WriteLine("Induced a collection.");
}

public:
static void OnFullGCCompleteEndNotify()
{
    // Method that informs the request queuing server
    // that this server is ready to accept requests again.
    AcceptRequests();
    Console::WriteLine("Accepting requests again.");
}

public:
static void WaitForFullGCProm()
{
    while (true)
    {
        // CheckForNotify is set to true and false in Main.
        while (checkForNotify)
        {
            // Check for a notification of an approaching collection.
        }
    }
}

```

```

GCNotificationStatus s = GC::WaitForFullGCAccroach();
if (s == GCNotificationStatus::Succeeded)
{
    Console::WriteLine("GC Notifiction raised.");
    OnFullGCAccroachNotify();
}
else if (s == GCNotificationStatus::Canceled)
{
    Console::WriteLine("GC Notification cancelled.");
    break;
}
else
{
    // This can occur if a timeout period
    // is specified for WaitForFullGCAccroach(Timeout)
    // or WaitForFullGCCComplete(Timeout)
    // and the time out period has elapsed.
    Console::WriteLine("GC Notification not applicable.");
    break;
}

// Check for a notification of a completed collection.
s = GC::WaitForFullGCCComplete();
if (s == GCNotificationStatus::Succeeded)
{
    Console::WriteLine("GC Notification raised.");
    OnFullGCCCompleteEndNotify();
}
else if (s == GCNotificationStatus::Canceled)
{
    Console::WriteLine("GC Notification cancelled.");
    break;
}
else
{
    // Could be a time out.
    Console::WriteLine("GC Notification not applicable.");
    break;
}
}

Thread::Sleep(500);
// FinalExit is set to true right before
// the main thread cancelled notification.
if (finalExit)
{
    break;
}
}

private:
static void RedirectRequests()
{
    // Code that sends requests
    // to other servers.

    // Suspend work.
    bAllocate = false;

}

static void FinishExistingRequests()
{
    // Code that waits a period of time
    // for pending requests to finish.

    // Clear the simulated workload.
}

```

```
load->Clear();

}

static void AcceptRequests()
{
    // Code that resumes processing
    // requests on this server.

    // Resume work.
    bAllocate = true;
}
};

}

int main()
{
    GCNotify::Program::Main();
}
```

```

public static void Main(string[] args)
{
    try
    {
        // Register for a notification.
        GC.RegisterForFullGCNotification(10, 10);
        Console.WriteLine("Registered for GC notification.");

        checkForNotify = true;
        bAllocate = true;

        // Start a thread using WaitForFullGCProc.
        Thread thWaitForFullGC = new Thread(new ThreadStart(WaitForFullGCProc));
        thWaitForFullGC.Start();

        // While the thread is checking for notifications in
        // WaitForFullGCProc, create objects to simulate a server workload.
        try
        {

            int lastCollCount = 0;
            int newCollCount = 0;

            while (true)
            {
                if (bAllocate)
                {
                    load.Add(new byte[1000]);
                    newCollCount = GC.CollectionCount(2);
                    if (newCollCount != lastCollCount)
                    {
                        // Show collection count when it increases:
                        Console.WriteLine("Gen 2 collection count: {0}", GC.CollectionCount(2).ToString());
                        lastCollCount = newCollCount;
                    }
                }

                // For ending the example (arbitrary).
                if (newCollCount == 500)
                {
                    finalExit = true;
                    checkForNotify = false;
                    break;
                }
            }
        }
        catch (OutOfMemoryException)
        {
            Console.WriteLine("Out of memory.");
        }

        finalExit = true;
        checkForNotify = false;
        GC.CancelFullGCNotification();
    }
    catch (InvalidOperationException invalidOp)
    {

        Console.WriteLine("GC Notifications are not supported while concurrent GC is enabled.\n"
            + invalidOp.Message);
    }
}

```

Imports System.Collections.Generic  
Imports System.Threading

```

Class Program
    ' Variables for continual checking in the
    ' While loop in the WaitForFullGcProc method.
    Private Shared checkForNotify As Boolean = False

    ' Variable for suspending work
    ' (such as servicing allocated server requests)
    ' after a notification is received and then
    ' resuming allocation after inducing a garbage collection.
    Private Shared bAllocate As Boolean = False

    ' Variable for ending the example.
    Private Shared finalExit As Boolean = False

    ' Collection for objects that
    ' simulate the server request workload.
    Private Shared load As New List(Of Byte())

Public Shared Sub Main(ByVal args() As String)
    Try
        ' Register for a notification.
        GC.RegisterForFullGCNotification(10, 10)
        Console.WriteLine("Registered for GC notification.")

        bAllocate = True
        checkForNotify = True

        ' Start a thread using WaitForFullGCPoc.
        Dim thWaitForFullGC As Thread = _
            New Thread(New ThreadStart(AddressOf WaitForFullGCPoc))
        thWaitForFullGC.Start()

        ' While the thread is checking for notifications in
        ' WaitForFullGCPoc, create objects to simulate a server workload.
        Try
            Dim lastCollCount As Integer = 0
            Dim newCollCount As Integer = 0

            While (True)
                If bAllocate = True Then

                    load.Add(New Byte(1000) {})
                    newCollCount = GC.CollectionCount(2)
                    If (newCollCount <> lastCollCount) Then
                        ' Show collection count when it increases:
                        Console.WriteLine("Gen 2 collection count: {0}", _
                            GC.CollectionCount(2).ToString)
                        lastCollCount = newCollCount
                    End If

                    ' For ending the example (arbitrary).
                    If newCollCount = 500 Then
                        finalExit = True
                        checkForNotify = False
                        bAllocate = False
                        Exit While
                    End If

                End If
            End While

            Catch outofMem As OutOfMemoryException
                Console.WriteLine("Out of memory.")
            End Try

            finalExit = True
            checkForNotify = False
        End Try
    End Try

```

```

    GC.CancelFullGCNotification()

    Catch invalidOp As InvalidOperationException
        Console.WriteLine("GC Notifications are not supported while concurrent GC is enabled." _
            & vbCrLf & invalidOp.Message)
    End Try
End Sub

Public Shared Sub OnFullGCApproachNotify()
    Console.WriteLine("Redirecting requests.")

    ' Method that tells the request queuing
    ' server to not direct requests to this server.
    RedirectRequests()

    ' Method that provides time to
    ' finish processing pending requests.
    FinishExistingRequests()

    ' This is a good time to induce a GC collection
    ' because the runtime will induce a full GC soon.
    ' To be very careful, you can check precede with a
    ' check of the GC.GCCollectionCount to make sure
    ' a full GC did not already occur since last notified.
    GC.Collect()
    Console.WriteLine("Induced a collection.")
End Sub

Public Shared Sub OnFullGCCompleteEndNotify()
    ' Method that informs the request queuing server
    ' that this server is ready to accept requests again.
    AcceptRequests()
    Console.WriteLine("Accepting requests again.")
End Sub

Public Shared Sub WaitForFullGCPoc()
    While True
        ' CheckForNotify is set to true and false in Main.

        While checkForNotify
            ' Check for a notification of an approaching collection.
            Dim s As GCNotificationStatus = GC.WaitForFullGCApproach
            If (s = GCNotificationStatus.Succeeded) Then
                Console.WriteLine("GC Notification raised.")
                OnFullGCApproachNotify()
            ElseIf (s = GCNotificationStatus.Canceled) Then
                Console.WriteLine("GC Notification cancelled.")
                Exit While
            Else
                ' This can occur if a timeout period
                ' is specified for WaitForFullGCApproach(Timeout)
                ' or WaitForFullGCComplete(Timeout)
                ' and the time out period has elapsed.
                Console.WriteLine("GC Notification not applicable.")
                Exit While
            End If

            ' Check for a notification of a completed collection.
            s = GC.WaitForFullGCComplete
            If (s = GCNotificationStatus.Succeeded) Then
                Console.WriteLine("GC Notification raised.")
                OnFullGCCompleteEndNotify()
            ElseIf (s = GCNotificationStatus.Canceled) Then
                Console.WriteLine("GC Notification cancelled.")
                Exit While
            Else
                ' Could be a time out.
                Console.WriteLine("GC Notification not applicable.")
            End If
        End While
    End Sub

```

```

        Exit While
    End If

End While
Thread.Sleep(500)
' FinalExit is set to true right before
' the main thread cancelled notification.
If finalExit Then
    Exit While
End If

End While
End Sub

Private Shared Sub RedirectRequests()
    ' Code that sends requests
    ' to other servers.

    ' Suspend work.
    bAllocate = False
End Sub

Private Shared Sub FinishExistingRequests()
    ' Code that waits a period of time
    ' for pending requests to finish.

    ' Clear the simulated workload.
    load.Clear()

End Sub

Private Shared Sub AcceptRequests()
    ' Code that resumes processing
    ' requests on this server.

    ' Resume work.
    bAllocate = True
End Sub
End Class

```

The following code contains the `WaitForFullGCProc` user method, that contains a continuous while loop to check for garbage collection notifications.

```

public:
    static void WaitForFullGCProc()
    {
        while (true)
        {
            // CheckForNotify is set to true and false in Main.
            while (checkForNotify)
            {
                // Check for a notification of an approaching collection.
                GCNotificationStatus s = GC::WaitForFullGCApproach();
                if (s == GCNotificationStatus::Succeeded)
                {
                    Console::WriteLine("GC Notifiction raised.");
                    OnFullGCApproachNotify();
                }
                else if (s == GCNotificationStatus::Canceled)
                {
                    Console::WriteLine("GC Notification cancelled.");
                    break;
                }
                else
                {
                    // This can occur if a timeout period
                    // is specified for WaitForFullGCApproach(Timeout)
                    // or WaitForFullGCCComplete(Timeout)
                    // and the time out period has elapsed.
                    Console::WriteLine("GC Notification not applicable.");
                    break;
                }
            }

            // Check for a notification of a completed collection.
            s = GC::WaitForFullGCCComplete();
            if (s == GCNotificationStatus::Succeeded)
            {
                Console::WriteLine("GC Notification raised.");
                OnFullGCCCompleteEndNotify();
            }
            else if (s == GCNotificationStatus::Canceled)
            {
                Console::WriteLine("GC Notification cancelled.");
                break;
            }
            else
            {
                // Could be a time out.
                Console::WriteLine("GC Notification not applicable.");
                break;
            }
        }

        Thread::Sleep(500);
        // FinalExit is set to true right before
        // the main thread cancelled notification.
        if (finalExit)
        {
            break;
        }
    }
}

```

```

public static void WaitForFullGCProc()
{
    while (true)
    {
        // CheckForNotify is set to true and false in Main.
        while (checkForNotify)
        {
            // Check for a notification of an approaching collection.
            GCNotificationStatus s = GC.WaitForFullGCApproach();
            if (s == GCNotificationStatus.Succeeded)
            {
                Console.WriteLine("GC Notification raised.");
                OnFullGCApproachNotify();
            }
            else if (s == GCNotificationStatus.Canceled)
            {
                Console.WriteLine("GC Notification cancelled.");
                break;
            }
            else
            {
                // This can occur if a timeout period
                // is specified for WaitForFullGCApproach(Timeout)
                // or WaitForFullGCCComplete(Timeout)
                // and the time out period has elapsed.
                Console.WriteLine("GC Notification not applicable.");
                break;
            }

            // Check for a notification of a completed collection.
            GCNotificationStatus status = GC.WaitForFullGCCComplete();
            if (status == GCNotificationStatus.Succeeded)
            {
                Console.WriteLine("GC Notification raised.");
                OnFullGCCCompleteEndNotify();
            }
            else if (status == GCNotificationStatus.Canceled)
            {
                Console.WriteLine("GC Notification cancelled.");
                break;
            }
            else
            {
                // Could be a time out.
                Console.WriteLine("GC Notification not applicable.");
                break;
            }
        }

        Thread.Sleep(500);
        // FinalExit is set to true right before
        // the main thread cancelled notification.
        if (finalExit)
        {
            break;
        }
    }
}

```

```

Public Shared Sub WaitForFullGCPoc()

    While True
        ' CheckForNotify is set to true and false in Main.

        While checkForNotify
            ' Check for a notification of an approaching collection.
            Dim s As GCNotificationStatus = GC.WaitForFullGCApproach
            If (s = GCNotificationStatus.Succeeded) Then
                Console.WriteLine("GC Notification raised.")
                OnFullGCApproachNotify()
            ElseIf (s = GCNotificationStatus.Canceled) Then
                Console.WriteLine("GC Notification cancelled.")
                Exit While
            Else
                ' This can occur if a timeout period
                ' is specified for WaitForFullGCApproach(Timeout)
                ' or WaitForFullGCCComplete(Timeout)
                ' and the time out period has elapsed.
                Console.WriteLine("GC Notification not applicable.")
                Exit While
            End If

            ' Check for a notification of a completed collection.
            s = GC.WaitForFullGCCComplete
            If (s = GCNotificationStatus.Succeeded) Then
                Console.WriteLine("GC Notifiction raised.")
                OnFullGCCCompleteEndNotify()
            ElseIf (s = GCNotificationStatus.Canceled) Then
                Console.WriteLine("GC Notification cancelled.")
                Exit While
            Else
                ' Could be a time out.
                Console.WriteLine("GC Notification not applicable.")
                Exit While
            End If

        End While
        Thread.Sleep(500)
        ' FinalExit is set to true right before
        ' the main thread cancelled notification.
        If finalExit Then
            Exit While
        End If

    End While
End Sub

```

The following code contains the `OnFullGCApproachNotify` method as called from the

`WaitForFullGCPoc` method.

```
public:  
    static void OnFullGCApproachNotify()  
    {  
        Console::WriteLine("Redirecting requests.");  
  
        // Method that tells the request queuing  
        // server to not direct requests to this server.  
        RedirectRequests();  
  
        // Method that provides time to  
        // finish processing pending requests.  
        FinishExistingRequests();  
  
        // This is a good time to induce a GC collection  
        // because the runtime will induce a full GC soon.  
        // To be very careful, you can check precede with a  
        // check of the GC.GCCount to make sure  
        // a full GC did not already occur since last notified.  
        GC::Collect();  
        Console::WriteLine("Induced a collection.");  
    }  
}
```

```
public static void OnFullGCApproachNotify()  
{  
  
    Console.WriteLine("Redirecting requests.");  
  
    // Method that tells the request queuing  
    // server to not direct requests to this server.  
    RedirectRequests();  
  
    // Method that provides time to  
    // finish processing pending requests.  
    FinishExistingRequests();  
  
    // This is a good time to induce a GC collection  
    // because the runtime will induce a full GC soon.  
    // To be very careful, you can check precede with a  
    // check of the GC.GCCount to make sure  
    // a full GC did not already occur since last notified.  
    GC.Collect();  
    Console.WriteLine("Induced a collection.");  
}  
}
```

```

Public Shared Sub OnFullGCApproachNotify()
    Console.WriteLine("Redirecting requests.")

    ' Method that tells the request queuing
    ' server to not direct requests to this server.
    RedirectRequests()

    ' Method that provides time to
    ' finish processing pending requests.
    FinishExistingRequests()

    ' This is a good time to induce a GC collection
    ' because the runtime will induce a full GC soon.
    ' To be very careful, you can check precede with a
    ' check of the GC.GCCount to make sure
    ' a full GC did not already occur since last notified.
    GC.Collect()
    Console.WriteLine("Induced a collection.")
End Sub

```

The following code contains the `OnFullGCApproachNotify` method as called from the `WaitForFullGCProc` method.

```

public:
    static void OnFullGCCCompleteEndNotify()
    {
        // Method that informs the request queuing server
        // that this server is ready to accept requests again.
        AcceptRequests();
        Console::WriteLine("Accepting requests again.");
    }

```

```

public static void OnFullGCCCompleteEndNotify()
{
    // Method that informs the request queuing server
    // that this server is ready to accept requests again.
    AcceptRequests();
    Console.WriteLine("Accepting requests again.");
}

```

```

Public Shared Sub OnFullGCCCompleteEndNotify()
    ' Method that informs the request queuing server
    ' that this server is ready to accept requests again.
    AcceptRequests()
    Console.WriteLine("Accepting requests again.")
End Sub

```

The following code contains the user methods that are called from the `OnFullGCApproachNotify` and `OnFullGCCCompleteNotify` methods. The user methods redirect requests, finish existing requests, and then resume requests after a full garbage collection has occurred.

```

private:
    static void RedirectRequests()
    {
        // Code that sends requests
        // to other servers.

        // Suspend work.
        bAllocate = false;

    }

    static void FinishExistingRequests()
    {
        // Code that waits a period of time
        // for pending requests to finish.

        // Clear the simulated workload.
        load->Clear();

    }

    static void AcceptRequests()
    {
        // Code that resumes processing
        // requests on this server.

        // Resume work.
        bAllocate = true;
    }

```

```

private static void RedirectRequests()
{
    // Code that sends requests
    // to other servers.

    // Suspend work.
    bAllocate = false;
}

private static void FinishExistingRequests()
{
    // Code that waits a period of time
    // for pending requests to finish.

    // Clear the simulated workload.
    load.Clear();
}

private static void AcceptRequests()
{
    // Code that resumes processing
    // requests on this server.

    // Resume work.
    bAllocate = true;
}

```

```

Private Shared Sub RedirectRequests()
    ' Code that sends requests
    ' to other servers.

    ' Suspend work.
    bAllocate = False
End Sub

Private Shared Sub FinishExistingRequests()
    ' Code that waits a period of time
    ' for pending requests to finish.

    ' Clear the simulated workload.
    load.Clear()

End Sub

Private Shared Sub AcceptRequests()
    ' Code that resumes processing
    ' requests on this server.

    ' Resume work.
    bAllocate = True
End Sub

```

The entire code sample is as follows:

```

using namespace System;
using namespace System::Collections::Generic;
using namespace System::Threading;

namespace GCNotify
{
    ref class Program
    {
private:
    // Variable for continual checking in the
    // While loop in the WaitForFullGCProc method.
    static bool checkForNotify = false;

    // Variable for suspending work
    // (such servicing allocated server requests)
    // after a notification is received and then
    // resuming allocation after inducing a garbage collection.
    static bool bAllocate = false;

    // Variable for ending the example.
    static bool finalExit = false;

    // Collection for objects that
    // simulate the server request workload.
    static List<array<Byte>^>^ load = gcnew List<array<Byte>^>();
}

public:
    static void Main()
    {
        try
        {
            // Register for a notification.
            GC::RegisterForFullGCNotification(10, 10);
            Console::WriteLine("Registered for GC notification.");

            checkForNotify = true;
            bAllocate = true;
        }
    }
}

```

```

// Start a thread using WaitForFullGCPProc.
Thread^ thWaitForFullGC = gcnew Thread(gcnew ThreadStart(&WaitForFullGCPProc));
thWaitForFullGC->Start();

// While the thread is checking for notifications in
// WaitForFullGCPProc, create objects to simulate a server workload.
try
{
    int lastCollCount = 0;
    int newCollCount = 0;

    while (true)
    {
        if (bAllocate)
        {
            load->Add(gcnew array<Byte>(1000));
            newCollCount = GC::CollectionCount(2);
            if (newCollCount != lastCollCount)
            {
                // Show collection count when it increases:
                Console::WriteLine("Gen 2 collection count: {0}",
GC::CollectionCount(2).ToString());
                lastCollCount = newCollCount;
            }
        }

        // For ending the example (arbitrary).
        if (newCollCount == 500)
        {
            finalExit = true;
            checkForNotify = false;
            break;
        }
    }
}

catch (OutOfMemoryException^)
{
    Console::WriteLine("Out of memory.");
}

finalExit = true;
checkForNotify = false;
GC::CancelFullGCNotification();

}

catch (InvalidOperationException^ invalidOp)
{

    Console::WriteLine("GC Notifications are not supported while concurrent GC is enabled.\n"
+ invalidOp->Message);
}
}

public:
    static void OnFullGCApproachNotify()
{
    Console::WriteLine("Redirecting requests.");

    // Method that tells the request queuing
    // server to not direct requests to this server.
    RedirectRequests();

    // Method that provides time to
    // finish processing pending requests.
    FinishExistingRequests();
}

```

```

// This is a good time to induce a GC collection
// because the runtime will induce a full GC soon.
// To be very careful, you can check precede with a
// check of the GC.GCCollectionCount to make sure
// a full GC did not already occur since last notified.
GC::Collect();
Console::WriteLine("Induced a collection.");

}

public:
static void OnFullGCCompleteEndNotify()
{
    // Method that informs the request queuing server
    // that this server is ready to accept requests again.
    AcceptRequests();
    Console::WriteLine("Accepting requests again.");
}

public:
static void WaitForFullGCPProc()
{
    while (true)
    {
        // CheckForNotify is set to true and false in Main.
        while (checkForNotify)
        {
            // Check for a notification of an approaching collection.
            GCNotificationStatus s = GC::WaitForFullGCApproach();
            if (s == GCNotificationStatus::Succeeded)
            {
                Console::WriteLine("GC Notifiction raised.");
                OnFullGCApproachNotify();
            }
            else if (s == GCNotificationStatus::Canceled)
            {
                Console::WriteLine("GC Notification cancelled.");
                break;
            }
            else
            {
                // This can occur if a timeout period
                // is specified for WaitForFullGCApproach(Timeout)
                // or WaitForFullGCComplete(Timeout)
                // and the time out period has elapsed.
                Console::WriteLine("GC Notification not applicable.");
                break;
            }
        }

        // Check for a notification of a completed collection.
        s = GC::WaitForFullGCComplete();
        if (s == GCNotificationStatus::Succeeded)
        {
            Console::WriteLine("GC Notification raised.");
            OnFullGCCompleteEndNotify();
        }
        else if (s == GCNotificationStatus::Canceled)
        {
            Console::WriteLine("GC Notification cancelled.");
            break;
        }
        else
        {
            // Could be a time out.
            Console::WriteLine("GC Notification not applicable.");
            break;
        }
    }
}

```

```

        Thread::Sleep(500);
        // FinalExit is set to true right before
        // the main thread cancelled notification.
        if (finalExit)
        {
            break;
        }
    }

private:
    static void RedirectRequests()
    {
        // Code that sends requests
        // to other servers.

        // Suspend work.
        bAllocate = false;

    }

    static void FinishExistingRequests()
    {
        // Code that waits a period of time
        // for pending requests to finish.

        // Clear the simulated workload.
        load->Clear();

    }

    static void AcceptRequests()
    {
        // Code that resumes processing
        // requests on this server.

        // Resume work.
        bAllocate = true;
    }
};

}

int main()
{
    GCNotify::Program::Main();
}

```

```

using System;
using System.Collections.Generic;
using System.Threading;

namespace GCNotify
{
    class Program
    {
        // Variable for continual checking in the
        // While loop in the WaitForFullGCProc method.
        static bool checkForNotify = false;

        // Variable for suspending work
        // (such servicing allocated server requests)
        // after a notification is received and then
        // resuming allocation after inducing a garbage collection.
        static bool bAllocate = false;
    }
}

```

```

// Variable for ending the example.
static bool finalExit = false;

// Collection for objects that
// simulate the server request workload.
static List<byte[]> load = new List<byte[]>();

public static void Main(string[] args)
{
    try
    {
        // Register for a notification.
        GC.RegisterForFullGCNotification(10, 10);
        Console.WriteLine("Registered for GC notification.");

        checkForNotify = true;
        bAllocate = true;

        // Start a thread using WaitForFullGCPoc.
        Thread thWaitForFullGC = new Thread(new ThreadStart(WaitForFullGCPoc));
        thWaitForFullGC.Start();

        // While the thread is checking for notifications in
        // WaitForFullGCPoc, create objects to simulate a server workload.
        try
        {
            int lastCollCount = 0;
            int newCollCount = 0;

            while (true)
            {
                if (bAllocate)
                {
                    load.Add(new byte[1000]);
                    newCollCount = GC.CollectionCount(2);
                    if (newCollCount != lastCollCount)
                    {
                        // Show collection count when it increases:
                        Console.WriteLine("Gen 2 collection count: {0}",
GC.CollectionCount(2).ToString());
                        lastCollCount = newCollCount;
                    }
                }

                // For ending the example (arbitrary).
                if (newCollCount == 500)
                {
                    finalExit = true;
                    checkForNotify = false;
                    break;
                }
            }
        }
        catch (OutOfMemoryException)
        {
            Console.WriteLine("Out of memory.");
        }

        finalExit = true;
        checkForNotify = false;
        GC.CancelFullGCNotification();
    }
    catch (InvalidOperationException invalidOp)
    {

        Console.WriteLine("GC Notifications are not supported while concurrent GC is enabled.\n"
+ invalidOp.Message);
    }
}

```

```

        }

    public static void OnFullGCApproachNotify()
    {

        Console.WriteLine("Redirecting requests.");

        // Method that tells the request queuing
        // server to not direct requests to this server.
        RedirectRequests();

        // Method that provides time to
        // finish processing pending requests.
        FinishExistingRequests();

        // This is a good time to induce a GC collection
        // because the runtime will induce a full GC soon.
        // To be very careful, you can check precede with a
        // check of the GC.GCCount to make sure
        // a full GC did not already occur since last notified.
        GC.Collect();
        Console.WriteLine("Induced a collection.");
    }

    public static void OnFullGCCCompleteEndNotify()
    {
        // Method that informs the request queuing server
        // that this server is ready to accept requests again.
        AcceptRequests();
        Console.WriteLine("Accepting requests again.");
    }

    public static void WaitForFullGCProc()
    {
        while (true)
        {
            // CheckForNotify is set to true and false in Main.
            while (checkForNotify)
            {
                // Check for a notification of an approaching collection.
                GCNotificationStatus s = GC.WaitForFullGCApproach();
                if (s == GCNotificationStatus.Succeeded)
                {
                    Console.WriteLine("GC Notification raised.");
                    OnFullGCApproachNotify();
                }
                else if (s == GCNotificationStatus.Canceled)
                {
                    Console.WriteLine("GC Notification cancelled.");
                    break;
                }
                else
                {
                    // This can occur if a timeout period
                    // is specified for WaitForFullGCApproach(Timeout)
                    // or WaitForFullGCCComplete(Timeout)
                    // and the time out period has elapsed.
                    Console.WriteLine("GC Notification not applicable.");
                    break;
                }
            }

            // Check for a notification of a completed collection.
            GCNotificationStatus status = GC.WaitForFullGCCComplete();
            if (status == GCNotificationStatus.Succeeded)
            {
                Console.WriteLine("GC Notification raised.");
                OnFullGCCCompleteEndNotify();
            }
            else if (status == GCNotificationStatus.Canceled)
        }
    }
}

```

```

        else if (status == GCNotificationStatus.Cancelled)
        {
            Console.WriteLine("GC Notification cancelled.");
            break;
        }
        else
        {
            // Could be a time out.
            Console.WriteLine("GC Notification not applicable.");
            break;
        }
    }

    Thread.Sleep(500);
    // FinalExit is set to true right before
    // the main thread cancelled notification.
    if (finalExit)
    {
        break;
    }
}
}

private static void RedirectRequests()
{
    // Code that sends requests
    // to other servers.

    // Suspend work.
    bAllocate = false;
}

private static void FinishExistingRequests()
{
    // Code that waits a period of time
    // for pending requests to finish.

    // Clear the simulated workload.
    load.Clear();
}

private static void AcceptRequests()
{
    // Code that resumes processing
    // requests on this server.

    // Resume work.
    bAllocate = true;
}
}
}

```

```

Imports System.Collections.Generic
Imports System.Threading

Class Program
    ' Variables for continual checking in the
    ' While loop in the WaitForFullGcProc method.
    Private Shared checkForNotify As Boolean = False

    ' Variable for suspending work
    ' (such as servicing allocated server requests)
    ' after a notification is received and then
    ' resuming allocation after inducing a garbage collection.
    Private Shared bAllocate As Boolean = False

    ' Variable for ending the example.
    Private Shared finalExit As Boolean = False

```

```

    Private Shared initialized As Boolean = False

    ' Collection for objects that
    ' simulate the server request workload.
    Private Shared load As New List(Of Byte())

    Public Shared Sub Main(ByVal args() As String)
        Try
            ' Register for a notification.
            GC.RegisterForFullGCNotification(10, 10)
            Console.WriteLine("Registered for GC notification.")

            bAllocate = True
            checkForNotify = True

            ' Start a thread using WaitForFullGCPoc.
            Dim thWaitForFullGC As Thread = _
                New Thread(New ThreadStart(AddressOf WaitForFullGCPoc))
            thWaitForFullGC.Start()

            ' While the thread is checking for notifications in
            ' WaitForFullGCPoc, create objects to simulate a server workload.
            Try
                Dim lastCollCount As Integer = 0
                Dim newCollCount As Integer = 0

                While (True)
                    If bAllocate = True Then

                        load.Add(New Byte(1000) {})
                        newCollCount = GC.CollectionCount(2)
                        If (newCollCount <> lastCollCount) Then
                            ' Show collection count when it increases:
                            Console.WriteLine("Gen 2 collection count: {0}", _
                                GC.CollectionCount(2).ToString)
                            lastCollCount = newCollCount
                        End If

                        ' For ending the example (arbitrary).
                        If newCollCount = 500 Then
                            finalExit = True
                            checkForNotify = False
                            bAllocate = False
                            Exit While
                        End If

                    End If
                End While

                Catch outofMem As OutOfMemoryException
                    Console.WriteLine("Out of memory.")
                End Try

                finalExit = True
                checkForNotify = False
                GC.CancelFullGCNotification()

                Catch invalidOp As InvalidOperationException
                    Console.WriteLine("GC Notifications are not supported while concurrent GC is enabled." _ 
                        & vbCrLf & invalidOp.Message)
                End Try
            End Sub

            Public Shared Sub OnFullGCApproachNotify()
                Console.WriteLine("Redirecting requests.")

                ' Method that tells the request queuing
                ' -----

```

```

    ' server to not direct requests to this server.
    RedirectRequests()

    ' Method that provides time to
    ' finish processing pending requests.
    FinishExistingRequests()

    ' This is a good time to induce a GC collection
    ' because the runtime will induce a full GC soon.
    ' To be very careful, you can check precede with a
    ' check of the GC.GCCount to make sure
    ' a full GC did not already occur since last notified.
    GC.Collect()
    Console.WriteLine("Induced a collection.")

End Sub

Public Shared Sub OnFullGCCompleteEndNotify()
    ' Method that informs the request queuing server
    ' that this server is ready to accept requests again.
    AcceptRequests()
    Console.WriteLine("Accepting requests again.")

End Sub

Public Shared Sub WaitForFullGCPoc()

    While True
        ' CheckForNotify is set to true and false in Main.

        While checkForNotify
            ' Check for a notification of an approaching collection.
            Dim s As GCNotificationStatus = GC.WaitForFullGCAccroach
            If (s = GCNotificationStatus.Succeeded) Then
                Console.WriteLine("GC Notification raised.")
                OnFullGCAccroachNotify()
            ElseIf (s = GCNotificationStatus.Canceled) Then
                Console.WriteLine("GC Notification cancelled.")
                Exit While
            Else
                ' This can occur if a timeout period
                ' is specified for WaitForFullGCAccroach(Timeout)
                ' or WaitForFullGCComplete(Timeout)
                ' and the time out period has elapsed.
                Console.WriteLine("GC Notification not applicable.")
                Exit While
            End If

            ' Check for a notification of a completed collection.
            s = GC.WaitForFullGCComplete
            If (s = GCNotificationStatus.Succeeded) Then
                Console.WriteLine("GC Notifiction raised.")
                OnFullGCCompleteEndNotify()
            ElseIf (s = GCNotificationStatus.Canceled) Then
                Console.WriteLine("GC Notification cancelled.")
                Exit While
            Else
                ' Could be a time out.
                Console.WriteLine("GC Notification not applicable.")
                Exit While
            End If

        End While
        Thread.Sleep(500)
        ' FinalExit is set to true right before
        ' the main thread cancelled notification.
        If finalExit Then
            Exit While
        End If

    End While

```

```
End Sub

Private Shared Sub RedirectRequests()
    ' Code that sends requests
    ' to other servers.

    ' Suspend work.
    bAllocate = False
End Sub

Private Shared Sub FinishExistingRequests()
    ' Code that waits a period of time
    ' for pending requests to finish.

    ' Clear the simulated workload.
    load.Clear()

End Sub

Private Shared Sub AcceptRequests()
    ' Code that resumes processing
    ' requests on this server.

    ' Resume work.
    bAllocate = True
End Sub
End Class
```

## See also

- [Garbage Collection](#)

# Application Domain Resource Monitoring

9/20/2022 • 4 minutes to read • [Edit Online](#)

Application domain resource monitoring (ARM) enables hosts to monitor CPU and memory usage by application domain. This is useful for hosts such as ASP.NET that use many application domains in a long-running process. The host can unload the application domain of an application that is adversely affecting the performance of the entire process, but only if it can identify the problematic application. ARM provides information that can be used to assist in making such decisions.

For example, a hosting service might have many applications running on an ASP.NET server. If one application in the process begins consuming too much memory or too much processor time, the hosting service can use ARM to identify the application domain that is causing the problem.

ARM is sufficiently lightweight to use in live applications. You can access the information by using event tracing for Windows (ETW) or directly through managed or native APIs.

## Enabling Resource Monitoring

ARM can be enabled in four ways: by supplying a configuration file when the common language runtime (CLR) is started, by using an unmanaged hosting API, by using managed code, or by listening to ARM ETW events.

As soon as ARM is enabled, it begins collecting data on all application domains in the process. If an application domain was created before ARM is enabled, cumulative data starts when ARM is enabled, not when the application domain was created. Once it is enabled, ARM cannot be disabled.

- You can enable ARM at CLR startup by adding the `<appDomainResourceMonitoring>` element to the configuration file, and setting the `enabled` attribute to `true`. A value of `false` (the default) means only that ARM is not enabled at startup; you can activate it later by using one of the other activation mechanisms.
- The host can enable ARM by requesting the `ICLRApDomainResourceMonitor` hosting interface. Once this interface is successfully obtained, ARM is enabled.
- Managed code can enable ARM by setting the static (`Shared` in Visual Basic) `AppDomain.MonitoringEnabled` property to `true`. As soon as the property is set, ARM is enabled.
- You can enable ARM after startup by listening to ETW events. ARM is enabled and begins raising events for all application domains when you enable the public provider `Microsoft-Windows-DotNETRuntime` by using the `AppDomainResourceManagementKeyword` keyword. To correlate data with application domains and threads, you must also enable the `Microsoft-Windows-DotNETRuntimeRundown` provider with the `ThreadingKeyword` keyword.

## Using ARM

ARM provides the total processor time that is used by an application domain and three kinds of information about memory use.

- **Total processor time for an application domain, in seconds:** This is calculated by adding up the thread times reported by the operating system for all threads that spent time executing in the application domain during its lifetime. Blocked or sleeping threads do not use processor time. When a thread calls into native code, the time that the thread spends in native code is included in the count for the application domain where the call was made.

- Managed API: [AppDomain.MonitoringTotalProcessorTime](#) property.
- Hosting API: [ICLRApDomainResourceMonitor::GetCurrentCpuTime](#) method.
- ETW events: `ThreadCreated`, `ThreadAppDomainEnter`, and `ThreadTerminated` events. For information about providers and keywords, see "AppDomain Resource Monitoring Events" in [CLR ETW Events](#).
- **Total managed allocations made by an application domain during its lifetime, in bytes:** Total allocations do not always reflect memory use by an application domain, because the allocated objects might be short-lived. However, if an application allocates and frees huge numbers of objects, the cost of the allocations could be significant.
  - Managed API: [AppDomain.MonitoringTotalAllocatedMemorySize](#) property.
  - Hosting API: [ICLRApDomainResourceMonitor::GetCurrentAllocated](#) method.
  - ETW events: `AppDomainMemAllocated` event, `Allocated` field.
- **Managed memory, in bytes, that is referenced by an application domain and that survived the most recent full, blocking collection:** This number is accurate only after a full, blocking collection. (This is in contrast to concurrent collections, which occur in the background and do not block the application.) For example, the [GC.Collect\(\)](#) method overload causes a full, blocking collection.
  - Managed API: [AppDomain.MonitoringSurvivedMemorySize](#) property.
  - Hosting API: [ICLRApDomainResourceMonitor::GetCurrentSurvived](#) method, `pAppDomainBytesSurvived` parameter.
  - ETW events: `AppDomainMemSurvived` event, `Survived` field.
- **Total managed memory, in bytes, that is referenced by the process and that survived the most recent full, blocking collection:** The survived memory for individual application domains can be compared to this number.
  - Managed API: [AppDomain.MonitoringSurvivedProcessMemorySize](#) property.
  - Hosting API: [ICLRApDomainResourceMonitor::GetCurrentSurvived](#) method, `pTotalBytesSurvived` parameter.
  - ETW events: `AppDomainMemSurvived` event, `ProcessSurvived` field.

## Determining When a Full, Blocking Collection Occurs

To determine when counts of survived memory are accurate, you need to know when a full, blocking collection has just occurred. The method for doing this depends on the API you use to examine ARM statistics.

### Managed API

If you use the properties of the [AppDomain](#) class, you can use the [GC.RegisterForFullGCNotification](#) method to register for notification of full collections. The threshold you use is not important, because you are waiting for the completion of a collection rather than the approach of a collection. You can then call the [GC.WaitForFullGCComplete](#) method, which blocks until a full collection has completed. You can create a thread that calls the method in a loop and does any necessary analysis whenever the method returns.

Alternatively, you can call the [GC.CollectionCount](#) method periodically to see if the count of generation 2 collections has increased. Depending on the polling frequency, this technique might not provide as accurate an indication of the occurrence of a full collection.

### Hosting API

If you use the unmanaged hosting API, your host must pass the CLR an implementation of the [IHostGCManager](#) interface. The CLR calls the [IHostGCManager::SuspensionEnding](#) method when it resumes execution of threads that have been suspended while a collection occurs. The CLR passes the generation of the completed collection

as a parameter of the method, so the host can determine whether the collection was full or partial. Your implementation of the [IHostGCManager::SuspensionEnding](#) method can query for survived memory, to ensure that the counts are retrieved as soon as they are updated.

## See also

- [AppDomain.MonitoringIsEnabled](#)
- [ICLRApDomainResourceMonitor Interface](#)
- [<appDomainResourceMonitoring>](#)
- [CLR ETW Events](#)

# Weak References

9/20/2022 • 2 minutes to read • [Edit Online](#)

The garbage collector cannot collect an object in use by an application while the application's code can reach that object. The application is said to have a strong reference to the object.

A weak reference permits the garbage collector to collect the object while still allowing the application to access the object. A weak reference is valid only during the indeterminate amount of time until the object is collected when no strong references exist. When you use a weak reference, the application can still obtain a strong reference to the object, which prevents it from being collected. However, there is always the risk that the garbage collector will get to the object first before a strong reference is reestablished.

Weak references are useful for objects that use a lot of memory, but can be recreated easily if they are reclaimed by garbage collection.

Suppose a tree view in a Windows Forms application displays a complex hierarchical choice of options to the user. If the underlying data is large, keeping the tree in memory is inefficient when the user is involved with something else in the application.

When the user switches away to another part of the application, you can use the [WeakReference](#) class to create a weak reference to the tree and destroy all strong references. When the user switches back to the tree, the application attempts to obtain a strong reference to the tree and, if successful, avoids reconstructing the tree.

To establish a weak reference with an object, you create a [WeakReference](#) using the instance of the object to be tracked. For a code example, see [WeakReference](#) in the class library.

## Short and Long Weak References

You can create a short weak reference or a long weak reference:

- Short

The target of a short weak reference becomes `null` when the object is reclaimed by garbage collection.

The weak reference is itself a managed object, and is subject to garbage collection just like any other managed object. A short weak reference is the parameterless constructor for [WeakReference](#).

- Long

A long weak reference is retained after the object's [Finalize](#) method has been called. This allows the object to be recreated, but the state of the object remains unpredictable. To use a long reference, specify `true` in the [WeakReference](#) constructor.

If the object's type does not have a [Finalize](#) method, the short weak reference functionality applies and the weak reference is valid only until the target is collected, which can occur anytime after the finalizer is run.

To establish a strong reference and use the object again, cast the [Target](#) property of a [WeakReference](#) to the type of the object. If the [Target](#) property returns `null`, the object was collected; otherwise, you can continue to use the object because the application has regained a strong reference to it.

## Guidelines for Using Weak References

Use long weak references only when necessary as the state of the object is unpredictable after finalization.

Avoid using weak references to small objects because the pointer itself may be as large or larger.

Avoid using weak references as an automatic solution to memory management problems. Instead, develop an effective caching policy for handling your application's objects.

## See also

- [Garbage Collection](#)

# Memory- and span-related types

9/20/2022 • 2 minutes to read • [Edit Online](#)

Starting with .NET Core 2.1, .NET includes a number of interrelated types that represent a contiguous, strongly typed region of arbitrary memory. These include:

- [System.Span<T>](#), a type that is used to access a contiguous region of memory. A [Span<T>](#) instance can be backed by an array of type `T`, a [String](#), a buffer allocated with [stackalloc](#), or a pointer to unmanaged memory. Because it has to be allocated on the stack, it has a number of restrictions. For example, a field in a class cannot be of type [Span<T>](#), nor can span be used in asynchronous operations.
- [System.ReadOnlySpan<T>](#), an immutable version of the [Span<T>](#) structure.
- [System.Memory<T>](#), a wrapper over a contiguous region of memory. A [Memory<T>](#) instance can be backed by an array of type `T`, or a [String](#), or a memory manager. As it can be stored on the managed heap, [Memory<T>](#) has none of the limitations of [Span<T>](#).
- [System.ReadOnlyMemory<T>](#), an immutable version of the [Memory<T>](#) structure.
- [System.Buffers.MemoryPool<T>](#), which allocates strongly typed blocks of memory from a memory pool to an owner. [IMemoryOwner<T>](#) instances can be rented from the pool by calling [MemoryPool<T>.Rent](#) and released back to the pool by calling [MemoryPool<T>.Dispose\(\)](#).
- [System.Buffers.IMemoryOwner<T>](#), which represents the owner of a block of memory and controls its lifetime management.
- [MemoryManager<T>](#), an abstract base class that can be used to replace the implementation of [Memory<T>](#) so that [Memory<T>](#) can be backed by additional types, such as safe handles. [MemoryManager<T>](#) is intended for advanced scenarios.
- [ArraySegment<T>](#), a wrapper for a particular number of array elements starting at a particular index.
- [System.MemoryExtensions](#), a collection of extension methods for converting strings, arrays, and array segments to [Memory<T>](#) blocks.

[System.Span<T>](#), [System.Memory<T>](#), and their readonly counterparts are designed to allow the creation of algorithms that avoid copying memory or allocating on the managed heap more than necessary. Creating them (either via `slice` or their constructors) does not involve duplicating the underlying buffers: only the relevant references and offsets, which represent the "view" of the wrapped memory, are updated.

## NOTE

For earlier frameworks, [Span<T>](#) and [Memory<T>](#) are available in the [System.Memory NuGet package](#).

For more information, see the [System.Buffers](#) namespace.

## Working with memory and span

Because the memory- and span-related types are typically used to store data in a processing pipeline, it is important that developers follow a set of best practices when using [Span<T>](#), [Memory<T>](#), and related types. These best practices are documented in [Memory<T> and Span<T> usage guidelines](#).

## See also

- [System.Memory<T>](#)
- [System.ReadOnlyMemory<T>](#)
- [System.Span<T>](#)
- [System.ReadOnlySpan<T>](#)
- [System.Buffers](#)

# Memory<T> and Span<T> usage guidelines

9/20/2022 • 15 minutes to read • [Edit Online](#)

.NET Core includes a number of types that represent an arbitrary contiguous region of memory. .NET Core 2.0 introduced [Span<T>](#) and [ReadOnlySpan<T>](#), which are lightweight memory buffers that wrap references to managed or unmanaged memory. Because these types can only be stored on the stack, they are unsuitable for a number of scenarios, including asynchronous method calls. .NET Core 2.1 adds a number of additional types, including [Memory<T>](#), [ReadOnlyMemory<T>](#), [IMemoryOwner<T>](#), and [MemoryPool<T>](#). Like [Span<T>](#), [Memory<T>](#) and its related types can be backed by both managed and unmanaged memory. Unlike [Span<T>](#), [Memory<T>](#) can be stored on the managed heap.

Both [Span<T>](#) and [Memory<T>](#) are wrappers over buffers of structured data that can be used in pipelines. That is, they are designed so that some or all of the data can be efficiently passed to components in the pipeline, which can process them and optionally modify the buffer. Because [Memory<T>](#) and its related types can be accessed by multiple components or by multiple threads, it's important that developers follow some standard usage guidelines to produce robust code.

## Owners, consumers, and lifetime management

Since buffers can be passed around between APIs, and since buffers can sometimes be accessed from multiple threads, it's important to consider lifetime management. There are three core concepts:

- **Ownership.** The owner of a buffer instance is responsible for lifetime management, including destroying the buffer when it's no longer in use. All buffers have a single owner. Generally the owner is the component that created the buffer or that received the buffer from a factory. Ownership can also be transferred; **Component-A** can relinquish control of the buffer to **Component-B**, at which point **Component-A** may no longer use the buffer, and **Component-B** becomes responsible for destroying the buffer when it's no longer in use.
- **Consumption.** The consumer of a buffer instance is allowed to use the buffer instance by reading from it and possibly writing to it. Buffers can have one consumer at a time unless some external synchronization mechanism is provided. The active consumer of a buffer isn't necessarily the buffer's owner.
- **Lease.** The lease is the length of time that a particular component is allowed to be the consumer of the buffer.

The following pseudo-code example illustrates these three concepts. `Buffer` in the pseudo-code represents a [Memory<T>](#) or [Span<T>](#) buffer of type [Char](#). The `Main` method instantiates the buffer, calls the `WriteInt32ToBuffer` method to write the string representation of an integer to the buffer, and then calls the `DisplayBufferToConsole` method to display the value of the buffer.

```

using System;

class Program
{
    // Write 'value' as a human-readable string to the output buffer.
    void WriteInt32ToBuffer(int value, Buffer buffer);

    // Display the contents of the buffer to the console.
    void DisplayBufferToConsole(Buffer buffer);

    // Application code
    static void Main()
    {
        var buffer = CreateBuffer();
        try
        {
            int value = Int32.Parse(Console.ReadLine());
            WriteInt32ToBuffer(value, buffer);
            DisplayBufferToConsole(buffer);
        }
        finally
        {
            buffer.Destroy();
        }
    }
}

```

The `Main` method creates the buffer and so is its owner. Therefore, `Main` is responsible for destroying the buffer when it's no longer in use. The pseudo-code illustrates this by calling a `Destroy` method on the buffer. (Neither `Memory<T>` nor `Span<T>` actually has a `Destroy` method. You'll see actual code examples later in this article.)

The buffer has two consumers, `WriteInt32ToBuffer` and `DisplayBufferToConsole`. There is only one consumer at a time (first `WriteInt32ToBuffer`, then `DisplayBufferToConsole`), and neither of the consumers owns the buffer. Note also that "consumer" in this context doesn't imply a read-only view of the buffer; consumers can modify the buffer's contents, as `WriteInt32ToBuffer` does, if given a read/write view of the buffer.

The `WriteInt32ToBuffer` method has a lease on (can consume) the buffer between the start of the method call and the time the method returns. Similarly, `DisplayBufferToConsole` has a lease on the buffer while it's executing, and the lease is released when the method unwinds. (There is no API for lease management; a "lease" is a conceptual matter.)

## Memory<T> and the owner/consumer model

As the [Owners, consumers, and lifetime management](#) section notes, a buffer always has an owner. .NET Core supports two ownership models:

- A model that supports single ownership. A buffer has a single owner for its entire lifetime.
- A model that supports ownership transfer. Ownership of a buffer can be transferred from its original owner (its creator) to another component, which then becomes responsible for the buffer's lifetime management. That owner can in turn transfer ownership to another component, and so on.

You use the `System.Buffers.IMemoryOwner<T>` interface to explicitly manage the ownership of a buffer. `IMemoryOwner<T>` supports both ownership models. The component that has an `IMemoryOwner<T>` reference owns the buffer. The following example uses an `IMemoryOwner<T>` instance to reflect the ownership of a `Memory<T>` buffer.

```

using System;
using System.Buffers;

class Example
{
    static void Main()
    {
        IMemoryOwner<char> owner = MemoryPool<char>.Shared.Rent();

        Console.WriteLine("Enter a number: ");
        try {
            var value = Int32.Parse(Console.ReadLine());

            var memory = owner.Memory;

            WriteInt32ToBuffer(value, memory);

            DisplayBufferToConsole(owner.Memory.Slice(0, value.ToString().Length));
        }
        catch (FormatException) {
            Console.WriteLine("You did not enter a valid number.");
        }
        catch (OverflowException) {
            Console.WriteLine($"You entered a number less than {Int32.MinValue:N0} or greater than
{Int32.MaxValue:N0}.");
        }
        finally {
            owner?.Dispose();
        }
    }

    static void WriteInt32ToBuffer(int value, Memory<char> buffer)
    {
        var strValue = value.ToString();

        var span = buffer.Span;
        for (int ctr = 0; ctr < strValue.Length; ctr++)
            span[ctr] = strValue[ctr];
    }

    static void DisplayBufferToConsole(Memory<char> buffer) =>
        Console.WriteLine($"Contents of the buffer: '{buffer}'");
}

```

We can also write this example with the `using`:

```

using System;
using System.Buffers;

class Example
{
    static void Main()
    {
        using (IMemoryOwner<char> owner = MemoryPool<char>.Shared.Rent())
        {
            Console.Write("Enter a number: ");
            try {
                var value = Int32.Parse(Console.ReadLine());

                var memory = owner.Memory;
                WriteInt32ToBuffer(value, memory);
                DisplayBufferToConsole(memory.Slice(0, value.ToString().Length));
            }
            catch (FormatException) {
                Console.WriteLine("You did not enter a valid number.");
            }
            catch (OverflowException) {
                Console.WriteLine($"You entered a number less than {Int32.MinValue:N0} or greater than
{Int32.MaxValue:N0}.");
            }
        }
    }

    static void WriteInt32ToBuffer(int value, Memory<char> buffer)
    {
        var strValue = value.ToString();

        var span = buffer.Slice(0, strValue.Length).Span;
        strValue.AsSpan().CopyTo(span);
    }

    static void DisplayBufferToConsole(Memory<char> buffer) =>
        Console.WriteLine($"Contents of the buffer: '{buffer}'");
}

```

In this code:

- The `Main` method holds the reference to the `IMemoryOwner<T>` instance, so the `Main` method is the owner of the buffer.
- The `WriteInt32ToBuffer` and `DisplayBufferToConsole` methods accept `Memory<T>` as a public API. Therefore, they are consumers of the buffer. And they only consume it one at a time.

Although the `WriteInt32ToBuffer` method is intended to write a value to the buffer, the `DisplayBufferToConsole` method isn't. To reflect this, it could have accepted an argument of type `ReadOnlyMemory<T>`. For more information on `ReadOnlyMemory<T>`, see [Rule #2: Use `ReadOnlySpan<T>` or `ReadOnlyMemory<T>` if the buffer should be read-only](#).

### "Ownerless" `Memory<T>` instances

You can create a `Memory<T>` instance without using `IMemoryOwner<T>`. In this case, ownership of the buffer is implicit rather than explicit, and only the single-owner model is supported. You can do this by:

- Calling one of the `Memory<T>` constructors directly, passing in a `T[]`, as the following example does.
- Calling the `String.AsMemory` extension method to produce a `ReadOnlyMemory<char>` instance.

```

using System;

class Example
{
    static void Main()
    {
        Memory<char> memory = new char[64];

        Console.WriteLine("Enter a number: ");
        var value = Int32.Parse(Console.ReadLine());

        WriteInt32ToBuffer(value, memory);
        DisplayBufferToConsole(memory);
    }

    static void WriteInt32ToBuffer(int value, Memory<char> buffer)
    {
        var strValue = value.ToString();
        strValue.AsSpan().CopyTo(buffer.Slice(0, strValue.Length).Span);
    }

    static void DisplayBufferToConsole(Memory<char> buffer) =>
        Console.WriteLine($"Contents of the buffer: '{buffer}'");
    }
}

```

The method that initially creates the `Memory<T>` instance is the implicit owner of the buffer. Ownership cannot be transferred to any other component because there is no `IMemoryOwner<T>` instance to facilitate the transfer. (As an alternative, you can also imagine that the runtime's garbage collector owns the buffer, and all methods just consume the buffer.)

## Usage guidelines

Because a memory block is owned but is intended to be passed to multiple components, some of which may operate upon a particular memory block simultaneously, it is important to establish guidelines for using both `Memory<T>` and `Span<T>`. Guidelines are necessary because:

- It is possible for a component to retain a reference to a memory block after its owner has released it.
- It is possible for a component to operate on a buffer at the same time that another component is operating on it, in the process corrupting the data in the buffer.
- While the stack-allocated nature of `Span<T>` optimizes performance and makes `Span<T>` the preferred type for operating on a memory block, it also subjects `Span<T>` to some major restrictions. It is important to know when to use a `Span<T>` and when to use `Memory<T>`.

The following are our recommendations for successfully using `Memory<T>` and its related types. Guidance that applies to `Memory<T>` and `Span<T>` also applies to `ReadOnlyMemory<T>` and `ReadOnlySpan<T>` unless we explicitly note otherwise.

### Rule #1: For a synchronous API, use `Span<T>` instead of `Memory<T>` as a parameter if possible.

`Span<T>` is more versatile than `Memory<T>` and can represent a wider variety of contiguous memory buffers. `Span<T>` also offers better performance than `Memory<T>`. Finally, you can use the `Memory<T>.Span` property to convert a `Memory<T>` instance to a `Span<T>`, although `Span<T>`-to-`Memory<T>` conversion isn't possible. So if your callers happen to have a `Memory<T>` instance, they'll be able to call your methods with `Span<T>` parameters anyway.

Using a parameter of type `Span<T>` instead of type `Memory<T>` also helps you write a correct consuming method implementation. You'll automatically get compile-time checks to ensure that you're not attempting to access the buffer beyond your method's lease (more on this later).

Sometimes, you'll have to use a `Memory<T>` parameter instead of a `Span<T>` parameter, even if you're fully synchronous. Perhaps an API that you depend on accepts only `Memory<T>` arguments. This is fine, but be aware of the tradeoffs involved when using `Memory<T>` synchronously.

#### Rule #2: Use `ReadOnlySpan<T>` or `ReadOnlyMemory<T>` if the buffer should be read-only.

In the earlier examples, the `DisplayBufferToConsole` method only reads from the buffer; it doesn't modify the contents of the buffer. The method signature should be changed to the following.

```
void DisplayBufferToConsole(ReadOnlyMemory<char> buffer);
```

In fact, if we combine this rule and Rule #1, we can do even better and rewrite the method signature as follows:

```
void DisplayBufferToConsole(ReadOnlySpan<char> buffer);
```

The `DisplayBufferToConsole` method now works with virtually every buffer type imaginable: `T[]`, storage allocated with `stackalloc`, and so on. You can even pass a `String` directly into it! For more information, see GitHub issue [dotnet/docs #25551](#).

#### Rule #3: If your method accepts `Memory<T>` and returns `void`, you must not use the `Memory<T>` instance after your method returns.

This relates to the "lease" concept mentioned earlier. A void-returning method's lease on the `Memory<T>` instance begins when the method is entered, and it ends when the method exits. Consider the following example, which calls `Log` in a loop based on input from the console.

```

using System;
using System.Buffers;

public class Example
{
    // implementation provided by third party
    static extern void Log(ReadOnlyMemory<char> message);

    // user code
    public static void Main()
    {
        using (var owner = MemoryPool<char>.Shared.Rent())
        {
            var memory = owner.Memory;
            var span = memory.Span;
            while (true)
            {
                int value = Int32.Parse(Console.ReadLine());
                if (value < 0)
                    return;

                int numCharsWritten = ToBuffer(value, span);
                Log(memory.Slice(0, numCharsWritten));
            }
        }
    }

    private static int ToBuffer(int value, Span<char> span)
    {
        string strValue = value.ToString();
        int length = strValue.Length;
        strValue.AsSpan().CopyTo(span.Slice(0, length));
        return length;
    }
}

```

If `Log` is a fully synchronous method, this code will behave as expected because there is only one active consumer of the memory instance at any given time. But imagine instead that `Log` has this implementation.

```

// !!! INCORRECT IMPLEMENTATION !!!
static void Log(ReadOnlyMemory<char> message)
{
    // Run in background so that we don't block the main thread while performing IO.
    Task.Run(() =>
    {
        StreamWriter sw = File.AppendText(@".\input-numbers.dat");
        sw.WriteLine(message);
    });
}

```

In this implementation, `Log` violates its lease because it still attempts to use the `Memory<T>` instance in the background after the original method has returned. The `Main` method could mutate the buffer while `Log` attempts to read from it, which could result in data corruption.

There are several ways to resolve this:

- The `Log` method can return a `Task` instead of `void`, as the following implementation of the `Log` method does.

```
// An acceptable implementation.
static Task Log(ReadOnlyMemory<char> message)
{
    // Run in the background so that we don't block the main thread while performing IO.
    return Task.Run(() => {
        StreamWriter sw = File.AppendText(@".\input-numbers.dat");
        sw.WriteLine(message);
        sw.Flush();
    });
}
```

- `Log` can instead be implemented as follows:

```
// An acceptable implementation.
static void Log(ReadOnlyMemory<char> message)
{
    string defensiveCopy = message.ToString();
    // Run in the background so that we don't block the main thread while performing IO.
    Task.Run(() => {
        StreamWriter sw = File.AppendText(@".\input-numbers.dat");
        sw.WriteLine(defensiveCopy);
        sw.Flush();
    });
}
```

**Rule #4: If your method accepts a `Memory<T>` and returns a `Task`, you must not use the `Memory<T>` instance after the `Task` transitions to a terminal state.**

This is just the async variant of Rule #3. The `Log` method from the earlier example can be written as follows to comply with this rule:

```
// An acceptable implementation.
static void Log(ReadOnlyMemory<char> message)
{
    // Run in the background so that we don't block the main thread while performing IO.
    Task.Run(() => {
        string defensiveCopy = message.ToString();
        StreamWriter sw = File.AppendText(@".\input-numbers.dat");
        sw.WriteLine(defensiveCopy);
        sw.Flush();
    });
}
```

Here, "terminal state" means that the task transitions to a completed, faulted, or canceled state. In other words, "terminal state" means "anything that would cause await to throw or to continue execution."

This guidance applies to methods that return `Task`, `Task<TResult>`, `ValueTask<TResult>`, or any similar type.

**Rule #5: If your constructor accepts `Memory<T>` as a parameter, instance methods on the constructed object are assumed to be consumers of the `Memory<T>` instance.**

Consider the following example:

```

class OddValueExtractor
{
    public OddValueExtractor(ReadOnlyMemory<int> input);
    public bool TryReadNextOddValue(out int value);
}

void PrintAllOddValues(ReadOnlyMemory<int> input)
{
    var extractor = new OddValueExtractor(input);
    while (extractor.TryReadNextOddValue(out int value))
    {
        Console.WriteLine(value);
    }
}

```

Here, the `OddValueExtractor` constructor accepts a `ReadOnlyMemory<int>` as a constructor parameter, so the constructor itself is a consumer of the `ReadOnlyMemory<int>` instance, and all instance methods on the returned value are also consumers of the original `ReadOnlyMemory<int>` instance. This means that `TryReadNextOddValue` consumes the `ReadOnlyMemory<int>` instance, even though the instance isn't passed directly to the `TryReadNextOddValue` method.

**Rule #6: If you have a settable `Memory<T>`-typed property (or an equivalent instance method) on your type, instance methods on that object are assumed to be consumers of the `Memory<T>` instance.**

This is really just a variant of Rule #5. This rule exists because property setters or equivalent methods are assumed to capture and persist their inputs, so instance methods on the same object may utilize the captured state.

The following example triggers this rule:

```

class Person
{
    // Settable property.
    public Memory<char> FirstName { get; set; }

    // alternatively, equivalent "setter" method
    public SetFirstName(Memory<char> value);

    // alternatively, a public settable field
    public Memory<char> FirstName;
}

```

**Rule #7: If you have an `IMemoryOwner<T>` reference, you must at some point dispose of it or transfer its ownership (but not both).**

Since a `Memory<T>` instance may be backed by either managed or unmanaged memory, the owner must call `Dispose` on `IMemoryOwner<T>` when work performed on the `Memory<T>` instance is complete. Alternatively, the owner may transfer ownership of the `IMemoryOwner<T>` instance to a different component, at which point the acquiring component becomes responsible for calling `Dispose` at the appropriate time (more on this later).

Failure to call the `Dispose` method on an `IMemoryOwner<T>` instance may lead to unmanaged memory leaks or other performance degradation.

This rule also applies to code that calls factory methods like `MemoryPool<T>.Rent`. The caller becomes the owner of the returned `IMemoryOwner<T>` and is responsible for disposing of the instance when finished.

**Rule #8: If you have an `IMemoryOwner<T>` parameter in your API surface, you are accepting ownership of that instance.**

Accepting an instance of this type signals that your component intends to take ownership of this instance. Your component becomes responsible for proper disposal according to Rule #7.

Any component that transfers ownership of the `IMemoryOwner<T>` instance to a different component should no longer use that instance after the method call completes.

#### IMPORTANT

If your constructor accepts `IMemoryOwner<T>` as a parameter, its type should implement `IDisposable`, and your `Dispose` method should call `Dispose` on the `IMemoryOwner<T>` object.

**Rule #9: If you're wrapping a synchronous p/invoke method, your API should accept `Span<T>` as a parameter.**

According to Rule #1, `Span<T>` is generally the correct type to use for synchronous APIs. You can pin `Span<T>` instances via the `fixed` keyword, as in the following example.

```
using System.Runtime.InteropServices;

[DllImport(...)]
private static extern unsafe int ExportedMethod(byte* pbData, int cbData);

public unsafe int ManagedWrapper(Span<byte> data)
{
    fixed (byte* pbData = &MemoryMarshal.GetReference(data))
    {
        int retVal = ExportedMethod(pbData, data.Length);

        /* error checking retVal goes here */

        return retVal;
    }
}
```

In the previous example, `pbData` can be null if, for example, the input span is empty. If the exported method absolutely requires that `pbData` be non-null, even if `cbData` is 0, the method can be implemented as follows:

```
public unsafe int ManagedWrapper(Span<byte> data)
{
    fixed (byte* pbData = &MemoryMarshal.GetReference(data))
    {
        byte dummy = 0;
        int retVal = ExportedMethod((pbData != null) ? pbData : &dummy, data.Length);

        /* error checking retVal goes here */

        return retVal;
    }
}
```

**Rule #10: If you're wrapping an asynchronous p/invoke method, your API should accept `Memory<T>` as a parameter.**

Since you cannot use the `fixed` keyword across asynchronous operations, you use the `Memory<T>.Pin` method to pin `Memory<T>` instances, regardless of the kind of contiguous memory the instance represents. The following example shows how to use this API to perform an asynchronous p/invoke call.

```
using System.Runtime.InteropServices;
```

```

[UnmanagedFunctionPointer(...)]
private delegate void OnCompletedCallback(IntPtr state, int result);

[DllImport(...)]
private static extern unsafe int ExportedAsyncMethod(byte* pbData, int cbData, IntPtr pState, IntPtr lpfnOnCompletedCallback);

private static readonly IntPtr _callbackPtr = GetCompletionCallbackPointer();

public unsafe Task<int> ManagedWrapperAsync(Memory<byte> data)
{
    // setup
    var tcs = new TaskCompletionSource<int>();
    var state = new MyCompletedCallbackState
    {
        Tcs = tcs
    };
    var pState = (IntPtr)GCHandle.Alloc(state);

    var memoryHandle = data.Pin();
    state.MemoryHandle = memoryHandle;

    // make the call
    int result;
    try
    {
        result = ExportedAsyncMethod((byte*)memoryHandle.Pointer, data.Length, pState, _callbackPtr);
    }
    catch
    {
        ((GCHandle)pState).Free(); // cleanup since callback won't be invoked
        memoryHandle.Dispose();
        throw;
    }

    if (result != PENDING)
    {
        // Operation completed synchronously; invoke callback manually
        // for result processing and cleanup.
        MyCompletedCallbackImplementation(pState, result);
    }
}

return tcs.Task;
}

private static void MyCompletedCallbackImplementation(IntPtr state, int result)
{
    GCHandle handle = (GCHandle)state;
    var actualState = (MyCompletedCallbackState)(handle.Target);
    handle.Free();
    actualState.MemoryHandle.Dispose();

    /* error checking result goes here */

    if (error)
    {
        actualState.Tcs.SetException(...);
    }
    else
    {
        actualState.Tcs.SetResult(result);
    }
}

private static IntPtr GetCompletionCallbackPointer()
{
    OnCompletedCallback callback = MyCompletedCallbackImplementation;
    GCHandle Alloc(callback); // keep alive for lifetime of application
    return Marshal.GetFunctionPointerForDelegate(callback);
}

```

```
}

private class MyCompletedCallbackState
{
    public TaskCompletionSource<int> Tcs;
    public MemoryHandle MemoryHandle;
}
```

## See also

- [System.Memory<T>](#)
- [System.Buffers.IMemoryOwner<T>](#)
- [System.Span<T>](#)

# Use SIMD-accelerated numeric types

9/20/2022 • 3 minutes to read • [Edit Online](#)

SIMD (Single instruction, multiple data) provides hardware support for performing an operation on multiple pieces of data, in parallel, using a single instruction. In .NET, there's set of SIMD-accelerated types under the [System.Numerics](#) namespace. SIMD operations can be parallelized at the hardware level. That increases the throughput of the vectorized computations, which are common in mathematical, scientific, and graphics apps.

## .NET SIMD-accelerated types

The .NET SIMD-accelerated types include the following types:

- The [Vector2](#), [Vector3](#), and [Vector4](#) types, which represent vectors with 2, 3, and 4 [Single](#) values.
- Two matrix types, [Matrix3x2](#), which represents a 3x2 matrix, and [Matrix4x4](#), which represents a 4x4 matrix of [Single](#) values.
- The [Plane](#) type, which represents a plane in three-dimensional space using [Single](#) values.
- The [Quaternion](#) type, which represents a vector that is used to encode three-dimensional physical rotations using [Single](#) values.
- The [Vector<T>](#) type, which represents a vector of a specified numeric type and provides a broad set of operators that benefit from SIMD support. The count of a [Vector<T>](#) instance is fixed for the lifetime of an application, but its value [Vector<T>.Count](#) depends on the CPU of the machine running the code.

### NOTE

The [Vector<T>](#) type is not included in the .NET Framework. You must install the [System.Numerics.Vectors](#) NuGet package to get access to this type.

The SIMD-accelerated types are implemented in such a way that they can be used with non-SIMD-accelerated hardware or JIT compilers. To take advantage of SIMD instructions, your 64-bit apps must be run by the runtime that uses the [RyuJIT](#) compiler. A [RyuJIT](#) compiler is included in .NET Core and in .NET Framework 4.6 and later. SIMD support is only provided when targeting 64-bit processors.

## How to use SIMD?

Before executing custom SIMD algorithms, it's possible to check if the host machine supports SIMD by using [Vector.IsHardwareAccelerated](#), which returns a [Boolean](#). This doesn't guarantee that SIMD-acceleration is enabled for a specific type, but is an indicator that it's supported by some types.

## Simple Vectors

The most primitive SIMD-accelerated types in .NET are [Vector2](#), [Vector3](#), and [Vector4](#) types, which represent vectors with 2, 3, and 4 [Single](#) values. The example below uses [Vector2](#) to add two vectors.

```
var v1 = new Vector2(0.1f, 0.2f);
var v2 = new Vector2(1.1f, 2.2f);
var vResult = v1 + v2;
```

It's also possible to use .NET vectors to calculate other mathematical properties of vectors such as [Dot product](#), [Transform](#), [Clamp](#) and so on.

```
var v1 = new Vector2(0.1f, 0.2f);
var v2 = new Vector2(1.1f, 2.2f);
var vResult1 = Vector2.Dot(v1, v2);
var vResult2 = Vector2.Distance(v1, v2);
var vResult3 = Vector2.Clamp(v1, Vector2.Zero, Vector2.One);
```

## Matrix

[Matrix3x2](#), which represents a 3x2 matrix, and [Matrix4x4](#), which represents a 4x4 matrix. Can be used for matrix-related calculations. The example below demonstrates multiplication of a matrix to its correspondent transpose matrix using SIMD.

```
var m1 = new Matrix4x4(
    1.1f, 1.2f, 1.3f, 1.4f,
    2.1f, 2.2f, 3.3f, 4.4f,
    3.1f, 3.2f, 3.3f, 3.4f,
    4.1f, 4.2f, 4.3f, 4.4f);

var m2 = Matrix4x4.Transpose(m1);
var mResult = Matrix4x4.Multiply(m1, m2);
```

## Vector<T>

The [Vector<T>](#) gives the ability to use longer vectors. The count of a [Vector<T>](#) instance is fixed, but its value [Vector<T>.Count](#) depends on the CPU of the machine running the code.

The following example demonstrates how to calculate the element-wise sum of two arrays using [Vector<T>](#).

```

double[] Sum(double[] left, double[] right)
{
    if (left is null)
    {
        throw new ArgumentNullException(nameof(left));
    }

    if (right is null)
    {
        throw new ArgumentNullException(nameof(right));
    }

    if (left.Length != right.Length)
    {
        throw new ArgumentException($"{nameof(left)} and {nameof(right)} are not the same length");
    }

    int length = left.Length;
    double[] result = new double[length];

    // Get the number of elements that can't be processed in the vector
    // NOTE: Vector<T>.Count is a JIT time constant and will get optimized accordingly
    int remaining = length % Vector<double>.Count;

    for (int i = 0; i < length - remaining; i += Vector<double>.Count)
    {
        var v1 = new Vector<double>(left, i);
        var v2 = new Vector<double>(right, i);
        (v1 + v2).CopyTo(result, i);
    }

    for (int i = length - remaining; i < length; i++)
    {
        result[i] = left[i] + right[i];
    }

    return result;
}

```

## Remarks

SIMD is more likely to remove one bottleneck and expose the next, for example memory throughput. In general the performance benefit of using SIMD varies depending on the specific scenario, and in some cases it can even perform worse than simpler non-SIMD equivalent code.

# Native interoperability

9/20/2022 • 2 minutes to read • [Edit Online](#)

The following articles show the various ways of doing "native interoperability" in .NET.

There are a few reasons why you'd want to call into native code:

- Operating systems come with a large volume of APIs that aren't present in the managed class libraries. A prime example for this scenario would be access to hardware or operating system management functions.
- Communicating with other components that have or can produce C-style ABIs (native ABIs), such as Java code that is exposed via [Java Native Interface \(JNI\)](#) or any other managed language that could produce a native component.
- On Windows, most of the software that gets installed, such as the Microsoft Office suite, registers COM components that represent their programs and allow developers to automate them or use them. This also requires native interoperability.

The previous list doesn't cover all of the potential situations and scenarios in which the developer would want/like/need to interface with native components. .NET class library, for instance, uses the native interoperability support to implement a fair number of its APIs, like console support and manipulation, file system access and others. However, it's important to note that there's an option if needed.

## NOTE

Most of the examples in this section will be presented for all three supported platforms for .NET Core (Windows, Linux and macOS). However, for some short and illustrative examples, just one sample is shown that uses Windows filenames and extensions (that is, "dll" for libraries). This doesn't mean that those features aren't available on Linux or macOS, it was done merely for convenience sake.

## See also

- [Platform Invoke \(P/Invoke\)](#)
- [Type marshalling](#)
- [Native interoperability best practices](#)
- [Disabled runtime marshalling](#)

# Platform Invoke (P/Invoke)

9/20/2022 • 7 minutes to read • [Edit Online](#)

P/Invoke is a technology that allows you to access structs, callbacks, and functions in unmanaged libraries from your managed code. Most of the P/Invoke API is contained in two namespaces: `System` and `System.Runtime.InteropServices`. Using these two namespaces give you the tools to describe how you want to communicate with the native component.

Let's start from the most common example, and that is calling unmanaged functions in your managed code. Let's show a message box from a command-line application:

```
using System;
using System.Runtime.InteropServices;

public class Program
{
    // Import user32.dll (containing the function we need) and define
    // the method corresponding to the native function.
    [DllImport("user32.dll", CharSet = CharSet.Unicode, SetLastError = true)]
    private static extern int MessageBox(IntPtr hWnd, string lpText, string lpCaption, uint uType);

    public static void Main(string[] args)
    {
        // Invoke the function as a regular managed method.
        MessageBox(IntPtr.Zero, "Command-line message box", "Attention!", 0);
    }
}
```

The previous example is simple, but it does show off what's needed to invoke unmanaged functions from managed code. Let's step through the example:

- Line #2 shows the using statement for the `System.Runtime.InteropServices` namespace that holds all the items needed.
- Line #8 introduces the `[DllImport]` attribute. This attribute is crucial, as it tells the runtime that it should load the unmanaged DLL. The string passed in is the DLL our target function is in. Additionally, it specifies which `character set` to use for marshalling the strings. Finally, it specifies that this function calls `SetLastError` and that the runtime should capture that error code so the user can retrieve it via `Marshal.GetLastWin32Error()`.
- Line #9 is the crux of the P/Invoke work. It defines a managed method that has the **exact same signature** as the unmanaged one. The declaration has a new keyword that you can notice, `extern`, which tells the runtime this is an external method, and that when you invoke it, the runtime should find it in the DLL specified in `[DllImport]` attribute.

The rest of the example is just invoking the method as you would any other managed method.

The sample is similar for macOS. The name of the library in the `[DllImport]` attribute needs to change since macOS has a different scheme of naming dynamic libraries. The following sample uses the `getpid(2)` function to get the process ID of the application and print it out to the console:

```

using System;
using System.Runtime.InteropServices;

namespace PInvokeSamples
{
    public static class Program
    {
        // Import the libSystem shared library and define the method
        // corresponding to the native function.
        [DllImport("libSystem.dylib")]
        private static extern int getpid();

        public static void Main(string[] args)
        {
            // Invoke the function and get the process ID.
            int pid = getpid();
            Console.WriteLine(pid);
        }
    }
}

```

It is also similar on Linux. The function name is the same, since `getpid(2)` is a standard [POSIX](#) system call.

```

using System;
using System.Runtime.InteropServices;

namespace PInvokeSamples
{
    public static class Program
    {
        // Import the libc shared library and define the method
        // corresponding to the native function.
        [DllImport("libc.so.6")]
        private static extern int getpid();

        public static void Main(string[] args)
        {
            // Invoke the function and get the process ID.
            int pid = getpid();
            Console.WriteLine(pid);
        }
    }
}

```

## Invoking managed code from unmanaged code

The runtime allows communication to flow in both directions, enabling you to call back into managed code from native functions by using function pointers. The closest thing to a function pointer in managed code is a **delegate**, so this is what is used to allow callbacks from native code into managed code.

The way to use this feature is similar to the managed to native process previously described. For a given callback, you define a delegate that matches the signature and pass that into the external method. The runtime will take care of everything else.

```

using System;
using System.Runtime.InteropServices;

namespace ConsoleApplication1
{
    public static class Program
    {
        // Define a delegate that corresponds to the unmanaged function.
        private delegate bool EnumWC(IntPtr hwnd, IntPtr lParam);

        // Import user32.dll (containing the function we need) and define
        // the method corresponding to the native function.
        [DllImport("user32.dll")]
        private static extern int EnumWindows(EnumWC lpEnumFunc, IntPtr lParam);

        // Define the implementation of the delegate; here, we simply output the window handle.
        private static bool OutputWindow(IntPtr hwnd, IntPtr lParam)
        {
            Console.WriteLine(hwnd.ToInt64());
            return true;
        }

        public static void Main(string[] args)
        {
            // Invoke the method; note the delegate as a first parameter.
            EnumWindows(OutputWindow, IntPtr.Zero);
        }
    }
}

```

Before walking through the example, it's good to review the signatures of the unmanaged functions you need to work with. The function to be called to enumerate all of the windows has the following signature:

```
BOOL EnumWindows (WNDENUMPROC lpEnumFunc, LPARAM lParam);
```

The first parameter is a callback. The said callback has the following signature:

```
BOOL CALLBACK EnumWindowsProc (HWND hwnd, LPARAM lParam);
```

Now, let's walk through the example:

- Line #9 in the example defines a delegate that matches the signature of the callback from unmanaged code. Notice how the LPARAM and HWND types are represented using `IntPtr` in the managed code.
- Lines #13 and #14 introduce the `EnumWindows` function from the user32.dll library.
- Lines #17 - 20 implement the delegate. For this simple example, we just want to output the handle to the console.
- Finally, in line #24, the external method is called and passed in the delegate.

The Linux and macOS examples are shown below. For them, we use the `ftw` function that can be found in `libc`, the C library. This function is used to traverse directory hierarchies and it takes a pointer to a function as one of its parameters. The said function has the following signature:

```
int (*fn) (const char *fpath, const struct stat *sb, int typeflag) .
```

```

using System;
using System.Runtime.InteropServices;

namespace PInvokeSamples
{
    public static class Program
    {
        // Define a delegate that has the same signature as the native function.
        private delegate int DirClbk(string fName, StatClass stat, int typeFlag);

        // Import the libc and define the method to represent the native function.
        [DllImport("libc.so.6")]
        private static extern int ftw(string dirpath, DirClbk cl, int descriptors);

        // Implement the above DirClbk delegate;
        // this one just prints out the filename that is passed to it.
        private static int DisplayEntry(string fName, StatClass stat, int typeFlag)
        {
            Console.WriteLine(fName);
            return 0;
        }

        public static void Main(string[] args)
        {
            // Call the native function.
            // Note the second parameter which represents the delegate (callback).
            ftw(".", DisplayEntry, 10);
        }
    }

    // The native callback takes a pointer to a struct. The below class
    // represents that struct in managed code. You can find more information
    // about this in the section on marshalling below.
    [StructLayout(LayoutKind.Sequential)]
    public class StatClass
    {
        public uint DeviceID;
        public uint InodeNumber;
        public uint Mode;
        public uint HardLinks;
        public uint UserID;
        public uint GroupID;
        public uint SpecialDeviceID;
        public ulong Size;
        public ulong BlockSize;
        public uint Blocks;
        public long TimeLastAccess;
        public long TimeLastModification;
        public long TimeLastStatusChange;
    }
}

```

macOS example uses the same function, and the only difference is the argument to the `DllImport` attribute, as macOS keeps `libc` in a different place.

```

using System;
using System.Runtime.InteropServices;

namespace PInvokeSamples
{
    public static class Program
    {
        // Define a delegate that has the same signature as the native function.
        private delegate int DirClbk(string fName, StatClass stat, int typeFlag);

        // Import the libc and define the method to represent the native function.
        [DllImport("libc")]
        private static extern int ftw(string dirpath, DirClbk cl, int descriptors);

        // Implement the above DirClbk delegate;
        // this one just prints out the filename that is passed to it.
        private static int DisplayEntry(string fName, StatClass stat, int typeFlag)
        {
            Console.WriteLine(fName);
            return 0;
        }

        public static void Main(string[] args)
        {
            // Call the native function.
            // Note the second parameter which represents the delegate (callback).
            ftw(".", DisplayEntry, 10);
        }
    }

    // The native callback takes a pointer to a struct. The below class
    // represents that struct in managed code.
    [StructLayout(LayoutKind.Sequential)]
    public class StatClass
    {
        public uint DeviceID;
        public uint InodeNumber;
        public uint Mode;
        public uint HardLinks;
        public uint UserID;
        public uint GroupID;
        public uint SpecialDeviceID;
        public ulong Size;
        public ulong BlockSize;
        public uint Blocks;
        public long TimeLastAccess;
        public long TimeLastModification;
        public long TimeLastStatusChange;
    }
}

```

Both of the previous examples depend on parameters, and in both cases, the parameters are given as managed types. Runtime does the "right thing" and processes these into its equivalents on the other side. Learn about how types are marshalled to native code in our page on [Type marshalling](#).

## More resources

- [PInvoke.net wiki](#) an excellent Wiki with information on common Windows APIs and how to call them.
- [P/Invoke in C++/CLI](#)
- [Mono documentation on P/Invoke](#)

# Writing cross platform P/Invoke code

9/20/2022 • 2 minutes to read • [Edit Online](#)

This article explains which paths the runtime searches when loading native libraries via P/Invoke. It also shows how to use [SetDllImportResolver](#).

## Library name variations

To facilitate simpler cross platform P/Invoke code, the runtime adds the canonical shared library extension (`.dll`, `.so` or `.dylib`) to native library names. On Linux and macOS, the runtime will also try prepending `lib`. These library names variations are automatically searched when you use APIs that load unmanaged libraries, such as [DllImportAttribute](#).

### NOTE

Absolute paths in library names (e.g., `/usr/lib/libc`) are treated as-is and no variations will be searched.

Consider the following example of using P/Invoke:

```
[DllImport("nativedep")]
static extern int ExportedFunction();
```

When running on Windows, the DLL is searched for in the following order:

1. `nativedep`
2. `nativedep.dll` (if the library name does not already end with `.dll` or `.exe`)

When running on Linux or macOS, the runtime will try prepending `lib` and appending the canonical shared library extension. On these OSes, library name variations are tried in the following order:

1. `nativedep.so` / `nativedep.dylib`
2. `libnativedep.so` / `libnativedep.dylib`¹
3. `nativedep`
4. `libnativedep`¹

On Linux, the search order is different if the library name ends with `.so` or contains `.so.` (note the trailing `.`). Consider the following example:

```
[DllImport("nativedep.so.6")]
static extern int ExportedFunction();
```

In this case, the library name variations are tried in the following order:

1. `nativedep.so.6`
2. `libnativedep.so.6`¹
3. `nativedep.so.6.so`
4. `libnativedep.so.6.so`¹

¹ Path is checked only if the library name does not contain a directory separator character (`/`).

## Custom import resolver

In more complex scenarios, you can use [SetDllImportResolver](#) to resolve DLL imports at run time. In the following example, `nativedep` is resolved to `nativedep_avx2` if the CPU supports it.

### TIP

This functionality is only available in .NET 5 and .NET Core 3.1 or later.

```
using System;
using System.Reflection;
using System.Runtime.InteropServices;

namespace PInvokeSamples
{
    public static class Program
    {
        [DllImport("nativedep")]
        private static extern int ExportedFunction();

        public static void Main(string[] args)
        {
            // Register the import resolver before calling the imported function.
            // Only one import resolver can be set for a given assembly.
            NativeLibrary.SetDllImportResolver(Assembly.GetExecutingAssembly(), DllImportResolver);

            int value = ExportedFunction();
            Console.WriteLine(value);
        }

        private static IntPtr DllImportResolver(string libraryName, Assembly assembly, DllImportSearchPath? searchPath)
        {
            if (libraryName == "nativedep")
            {
                // On systems with AVX2 support, load a different library.
                if (System.Runtime.Intrinsics.X86.Avx2.IsSupported)
                {
                    return NativeLibrary.Load("nativedep_avx2", assembly, searchPath);
                }
            }

            // Otherwise, fallback to default import resolver.
            return IntPtr.Zero;
        }
    }
}
```

## See also

- [Platform Invoke \(P/Invoke\)](#)

# Source generation for platform invokes

9/20/2022 • 2 minutes to read • [Edit Online](#)

.NET 7 introduces a [source generator](#) for P/Invokes in C# which recognizes the [LibraryImportAttribute](#).

When a P/Invoke is defined and called as shown in the following code, the built-in interop system in the .NET runtime generates an IL stub—a stream of IL instructions that is JIT-ed—at run time to facilitate the transition from managed to unmanaged.

```
[DllImport(
    "nativelib",
    EntryPoint = "to_lower",
    CharSet = CharSet.Unicode)]
internal static extern string ToLower(string str);

// string lower = ToLower("StringToConvert");
```

The IL stub handles [marshalling](#) of parameters and return values and calling the unmanaged code while respecting settings on [DllImportAttribute](#) that affect how the unmanaged code should be invoked (for example, [SetLastError](#)). Since this IL stub is generated at run time, it is not available for ahead-of-time (AOT) compiler or IL trimming scenarios. Debugging the marshalling logic is also a non-trivial exercise.

The P/Invoke source generator, included with the .NET 7 SDK and enabled by default, looks for [LibraryImportAttribute](#) on a `static` and `partial` method to trigger compile-time source generation of marshalling code, removing the need for the generation of an IL stub at run time and allowing the P/Invoke to be inlined. Analyzers and code fixers are also included to help with migration from the built-in system to the source generator and with usage in general.

## Basic usage

The [LibraryImportAttribute](#) is designed to be similar to [DllImportAttribute](#) in usage. We can convert the example above to use P/Invoke source generation by using the [LibraryImportAttribute](#) and marking the method as `partial` instead of `extern`:

```
[LibraryImport(
    "nativelib",
    EntryPoint = "to_lower",
    StringMarshalling = StringMarshalling.Utf16)]
internal static partial string ToLower(string str);
```

During compilation, the source generator will trigger to generate an implementation of the `ToLower` method that handles marshalling of the `string` parameter and return value as UTF-16. Since the marshalling is now generated source code, you can actually look at and step through the logic in a debugger.

`MarshalAs`

The source generator also respects the [MarshalAsAttribute](#). The preceding code could also be written as:

```
[LibraryImport(
    "nativeLib",
    EntryPoint = "to_lower")]
[return: MarshalAs(UnmanagedType.LPWStr)]
internal static partial string ToLower(
    [MarshalAs(UnmanagedType.LPWStr)] string str);
```

Some settings for [MarshalAsAttribute](#) are not supported. The source generator will emit an error if you try to use unsupported settings. For more details, see [Differences from DllImport](#).

## Calling convention

To specify the calling convention, use [UnmanagedCallConvAttribute](#), for example:

```
[LibraryImport(
    "nativeLib",
    EntryPoint = "to_lower",
    StringMarshalling = StringMarshalling.Utf16)]
[UnmanagedCallConv(
    CallConvs = new[] { typeof(CallConvStdcall) })]
internal static partial string ToLower(string str);
```

## Differences from [DllImport](#)

[LibraryImportAttribute](#) is intended to be a straight-forward conversion from [DllImportAttribute](#) in most cases, but there are some intentional changes.

- [CallingConvention](#) has no equivalent on [LibraryImportAttribute](#). [UnmanagedCallConvAttribute](#) should be used instead.
- [CharSet](#) (for  [CharSet](#)) has been replaced with  [StringMarshalling](#) (for  [StringMarshalling](#)). ANSI has been removed and UTF-8 is now available as a first-class option.
- [BestFitMapping](#) and  [ThrowOnUnmappableChar](#) have no equivalent. These were only relevant when marshalling an ANSI string on Windows. The generated code for marshalling an ANSI string will have the equivalent behaviour of `BestFitMapping=false` and `ThrowOnUnmappableChar=false`.
- [ExactSpelling](#) has no equivalent. This was a Windows-centric setting and had no effect on non-Windows. The method name or  [EntryPoint](#) should be the exact spelling of the entry point name. This field has historical uses related to the `A` and `W` suffixes used in Win32 programming.
- [PreserveSig](#) has no equivalent. This was a Windows-centric setting. The generated code always directly translates the signature.

There are also differences in support for some settings on [MarshalAsAttribute](#), default marshalling of certain types, and other interop-related attributes. For more details, see our [documentation on compatibility differences](#).

## See also

- [P/Invoke](#)
- [LibraryImportAttribute](#)

# Type marshalling

9/20/2022 • 4 minutes to read • [Edit Online](#)

**Marshalling** is the process of transforming types when they need to cross between managed and native code.

Marshalling is needed because the types in the managed and unmanaged code are different. In managed code, for instance, you have a `string`, while in the unmanaged world strings can be Unicode ("wide"), non-Unicode, null-terminated, ASCII, etc. By default, the P/Invoke subsystem tries to do the right thing based on the default behavior, described on this article. However, for those situations where you need extra control, you can employ the `MarshalAs` attribute to specify what is the expected type on the unmanaged side. For instance, if you want the string to be sent as a null-terminated ANSI string, you could do it like this:

```
[DllImport("somenativelibrary.dll")]
static extern int MethodA([MarshalAs(UnmanagedType.LPStr)] string parameter);
```

If you apply the `System.Runtime.CompilerServices.DisableRuntimeMarshallingAttribute` attribute to the assembly, the rules in the following section don't apply. For information on how .NET values are exposed to native code when this attribute is applied, see [disabled runtime marshalling](#).

## Default rules for marshalling common types

Generally, the runtime tries to do the "right thing" when marshalling to require the least amount of work from you. The following tables describe how each type is marshalled by default when used in a parameter or field. The C99/C++11 fixed-width integer and character types are used to ensure that the following table is correct for all platforms. You can use any native type that has the same alignment and size requirements as these types.

This first table describes the mappings for various types for whom the marshalling is the same for both P/Invoke and field marshalling.

C# KEYWORD	.NET TYPE	NATIVE TYPE
<code>byte</code>	<code>System.Byte</code>	<code>uint8_t</code>
<code>sbyte</code>	<code>System.SByte</code>	<code>int8_t</code>
<code>short</code>	<code>System.Int16</code>	<code>int16_t</code>
<code>ushort</code>	<code>System.UInt16</code>	<code>uint16_t</code>
<code>int</code>	<code>System.Int32</code>	<code>int32_t</code>
<code>uint</code>	<code>System.UInt32</code>	<code>uint32_t</code>
<code>long</code>	<code>System.Int64</code>	<code>int64_t</code>
<code>ulong</code>	<code>System.UInt64</code>	<code>uint64_t</code>

C# KEYWORD	.NET TYPE	NATIVE TYPE
<code>char</code>	<code>System.Char</code>	Either <code>char</code> or <code>char16_t</code> depending on the <code>CharSet</code> of the P/Invoke or structure. See the <a href="#">charset documentation</a> .
	<code>System.Char</code>	Either <code>char*</code> or <code>char16_t*</code> depending on the <code>CharSet</code> of the P/Invoke or structure. See the <a href="#">charset documentation</a> .
<code>nint</code>	<code>System.IntPtr</code>	<code>intptr_t</code>
<code>nuint</code>	<code>System.UIntPtr</code>	<code>uintptr_t</code>
	.NET Pointer types (ex. <code>void*</code> )	<code>void*</code>
	Type derived from <code>System.Runtime.InteropServices.SafeHandle</code>	<code>void*</code>
	Type derived from <code>System.Runtime.InteropServices.CriticalHandle</code>	<code>void*</code>
<code>bool</code>	<code>System.Boolean</code>	Win32 <code>BOOL</code> type
<code>decimal</code>	<code>System.Decimal</code>	COM <code>DECIMAL</code> struct
	.NET Delegate	Native function pointer
	<code>System.DateTime</code>	Win32 <code>DATE</code> type
	<code>System.Guid</code>	Win32 <code>GUID</code> type

A few categories of marshalling have different defaults if you're marshalling as a parameter or structure.

.NET TYPE	NATIVE TYPE (PARAMETER)	NATIVE TYPE (FIELD)
.NET array	A pointer to the start of an array of native representations of the array elements.	Not allowed without a <code>[MarshalAs]</code> attribute
A class with a <code>LayoutKind</code> of <code>Sequential</code> or <code>Explicit</code>	A pointer to the native representation of the class	The native representation of the class

The following table includes the default marshalling rules that are Windows-only. On non-Windows platforms, you cannot marshal these types.

.NET TYPE	NATIVE TYPE (PARAMETER)	NATIVE TYPE (FIELD)
<code>System.Object</code>	<code>VARIANT</code>	<code>IUnknown*</code>

.NET TYPE	NATIVE TYPE (PARAMETER)	NATIVE TYPE (FIELD)
<code>System.Array</code>	COM interface	Not allowed without a <code>[MarshalAs]</code> attribute
<code>System.ArgIterator</code>	<code>va_list</code>	Not allowed
<code>System.Collections.IEnumerator</code>	<code>IEnumVARIANT*</code>	Not allowed
<code>System.Collections.IEnumerable</code>	<code>IDispatch*</code>	Not allowed
<code>System.DateTimeOffset</code>	<code>int64_t</code> representing the number of ticks since midnight on January 1, 1601	<code>int64_t</code> representing the number of ticks since midnight on January 1, 1601

Some types can only be marshalled as parameters and not as fields. These types are listed in the following table:

.NET TYPE	NATIVE TYPE (PARAMETER ONLY)
<code>System.Text.StringBuilder</code>	Either <code>char*</code> or <code>char16_t*</code> depending on the <code>CharSet</code> of the P/Invoke. See the <a href="#">charset documentation</a> .
<code>System.ArgIterator</code>	<code>va_list</code> (on Windows x86/x64/arm64 only)
<code>System.Runtime.InteropServices.ArrayWithOffset</code>	<code>void*</code>
<code>System.Runtime.InteropServices.HandleRef</code>	<code>void*</code>

If these defaults don't do exactly what you want, you can customize how parameters are marshalled. The [parameter marshalling](#) article walks you through how to customize how different parameter types are marshalled.

## Default marshalling in COM scenarios

When you are calling methods on COM objects in .NET, the .NET runtime changes the default marshalling rules to match common COM semantics. The following table lists the rules that .NET runtimes uses in COM scenarios:

.NET TYPE	NATIVE TYPE (COM METHOD CALLS)
<code>System.Boolean</code>	<code>VARIANT_BOOL</code>
<code>StringBuilder</code>	<code>LPWSTR</code>
<code>System.String</code>	<code>BSTR</code>
Delegate types	<code>_Delegate*</code> in .NET Framework. Disallowed in .NET Core and .NET 5+.
<code>System.Drawing.Color</code>	<code>OLECOLOR</code>
.NET array	<code>SAFEARRAY</code>

.NET TYPE	NATIVE TYPE (COM METHOD CALLS)
System.String[]	SAFEARRAY of BSTRs

## Marshalling classes and structs

Another aspect of type marshalling is how to pass in a struct to an unmanaged method. For instance, some of the unmanaged methods require a struct as a parameter. In these cases, you need to create a corresponding struct or a class in managed part of the world to use it as a parameter. However, just defining the class isn't enough, you also need to instruct the marshaller how to map fields in the class to the unmanaged struct. Here the `StructLayout` attribute becomes useful.

```
[DllImport("kernel32.dll")]
static extern void GetSystemTime(SystemTime systemTime);

[StructLayout(LayoutKind.Sequential)]
class SystemTime {
    public ushort Year;
    public ushort Month;
    public ushort DayOfWeek;
    public ushort Day;
    public ushort Hour;
    public ushort Minute;
    public ushort Second;
    public ushort Millisecond;
}

public static void Main(string[] args) {
    SystemTime st = new SystemTime();
    GetSystemTime(st);
    Console.WriteLine(st.Year);
}
```

The previous code shows a simple example of calling into `GetSystemTime()` function. The interesting bit is on line 4. The attribute specifies that the fields of the class should be mapped sequentially to the struct on the other (unmanaged) side. This means that the naming of the fields isn't important, only their order is important, as it needs to correspond to the unmanaged struct, shown in the following example:

```
typedef struct _SYSTEMTIME {
    WORD wYear;
    WORD wMonth;
    WORD wDayOfWeek;
    WORD wDay;
    WORD wHour;
    WORD wMinute;
    WORD wSecond;
    WORD wMilliseconds;
} SYSTEMTIME, *PSYSTEMTIME;
```

Sometimes the default marshalling for your structure doesn't do what you need. The [Customizing structure marshalling](#) article teaches you how to customize how your structure is marshalled.

# Charsets and marshalling

9/20/2022 • 2 minutes to read • [Edit Online](#)

The way `char` values, `string` objects, and `System.Text.StringBuilder` objects are marshalled depends on the value of the `Charset` field on either the P/Invoke or structure. You can set the `CharSet` of a P/Invoke by setting the `DllImportAttribute.CharSet` field when declaring your P/Invoke. To set the `Charset` for a type, set the `StructLayoutAttribute.CharSet` field on your class or struct declaration. When these attribute fields are not set, it is up to the language compiler to determine which `Charset` to use. C#, Visual Basic, and F# use the `None` charset by default, which has the same behavior as the `Ansi` charset.

If the `System.Runtime.InteropServices.DefaultCharsetAttribute` is applied on the module in C# or Visual Basic code, then the C# or Visual Basic compiler will emit the provided `charset` by default instead of using `CharSet.None`. F# does not support the `DefaultCharsetAttribute`, and always emits `CharSet.None` by default.

The following table shows a mapping between each charset and how a character or string is represented when marshalled with that charset:

CHARSET VALUE	WINDOWS	.NET CORE 2.2 AND EARLIER ON UNIX	.NET CORE 3.0 AND LATER AND MONO ON UNIX
Ansi	<code>char</code> (the system default Windows (ANSI) code page)	<code>char</code> (UTF-8)	<code>char</code> (UTF-8)
Unicode	<code>wchar_t</code> (UTF-16)	<code>char16_t</code> (UTF-16)	<code>char16_t</code> (UTF-16)
Auto	<code>wchar_t</code> (UTF-16)	<code>char16_t</code> (UTF-16)	<code>char</code> (UTF-8)

Make sure you know what representation your native representation expects when picking your charset.

# Disabled runtime marshalling

9/20/2022 • 2 minutes to read • [Edit Online](#)

When the `System.Runtime.CompilerServices.DisableRuntimeMarshallingAttribute` attribute is applied to an assembly, the runtime disables most built-in support for data marshalling between managed and native representations. This article describes the features that are disabled and how .NET types map to native types when marshalling is disabled.

## Scenarios where marshalling is disabled

When the `DisableRuntimeMarshallingAttribute` is applied to an assembly, it affects P/Invokes and Delegate types in the assembly, as well as any calls to unmanaged function pointers in the assembly. It does not affect any P/Invoke or interop delegate types defined in other assemblies. It also does not disable marshalling for the runtime's built-in COM interop support. The built-in COM interop support can be enabled or disabled through a [feature switch](#).

### Disabled features

When the `DisableRuntimeMarshallingAttribute` is applied to an assembly, the following attributes will have no effect or throw an exception:

- `LCIDConversionAttribute` on a P/Invoke or a delegate
- `SetLastError=true` on a P/Invoke
- `ThrowOnUnmappableChar=true` on a P/Invoke
- `BestFitMapping=true` on a P/Invoke
- .NET variadic argument method signatures (varargs)
- `in`, `ref`, `out` parameters

## Default rules for marshalling common types

When marshalling is disabled, the rules for default marshalling change to much simpler rules. These rules are described below. As mentioned in the [interop best practices documentation](#), blittable types are types with the same layout in managed and native code and as such do not require any marshalling. Additionally, these rules cannot be customized with the tools mentioned in [the documentation on customizing parameter marshalling](#).

C# KEYWORD	.NET TYPE	NATIVE TYPE
<code>byte</code>	<code>System.Byte</code>	<code>uint8_t</code>
<code>sbyte</code>	<code>System.SByte</code>	<code>int8_t</code>
<code>short</code>	<code>System.Int16</code>	<code>int16_t</code>
<code>ushort</code>	<code>System.UInt16</code>	<code>uint16_t</code>
<code>int</code>	<code>System.Int32</code>	<code>int32_t</code>
<code>uint</code>	<code>System.UInt32</code>	<code>uint32_t</code>

C# KEYWORD	.NET TYPE	NATIVE TYPE
long	System.Int64	int64_t
ulong	System.UInt64	uint64_t
char	System.Char	char16_t ( Charset on the P/Invoke has no effect)
nint	System.IntPtr	intptr_t
nuint	System.UIntPtr	uintptr_t
	System.Boolean	bool
	User-defined C# unmanaged type with no fields with LayoutKind.Auto	Treated as a blittable type. All customized struct marshalling is ignored.
	All other types	unsupported

## Examples

The following example shows some features that are enabled or disabled when runtime marshalling is disabled:

```
using System.Runtime.InteropServices;

struct Unmanaged
{
    int i;
}

[StructLayout(LayoutKind.Auto)]
struct AutoLayout
{
    int i;
}

struct StructWithAutoLayoutField
{
    AutoLayout f;
}

[UnmanagedFunctionPointer] // OK: UnmanagedFunctionPointer attribute is supported
public delegate void Callback();

[UnmanagedFunctionPointer(CallingConvention.Cdecl)] // OK: Specifying a calling convention is supported
public delegate void Callback2(int i); // OK: primitive value types are allowed

[DllImport("NativeLibrary", EntryPoint = "CustomEntryPointName")] // OK: Specifying a custom entry-point
name is supported
public static extern void Import(int i);

[DllImport("NativeLibrary", CallingConvention = CallingConvention.Cdecl)] // OK: Specifying a custom calling
convention is supported
public static extern void Import(int i);

[UnmanagedCallConv(new[] { typeof(CallConv.Cdecl) })] // OK: Specifying a custom calling convention is
supported
[DllImport("NativeLibrary")]
public static extern void Import(int i);

[DllImport("NativeLibrary", EntryPoint = "CustomEntryPointName", CharSet = CharSet.Unicode, ExactSpelling =
false)] // OK: Specifying a custom entry-point name and using CharSet-based lookup is supported
public static extern void Import(int i);

[DllImport("NativeLibrary")] // OK: Not explicitly specifying an entry-point name is supported
public static extern void Import(Unmanaged u); // OK: unmanaged type

[DllImport("NativeLibrary")] // OK: Not explicitly specifying an entry-point name is supported
public static extern void Import(StructWithAutoLayoutField u); // Error: unmanaged type with auto-layout
field

[DllImport("NativeLibrary")]
public static extern void Import(Callback callback); // Error: managed types are not supported when runtime
marshalling is disabled
```

# Customize structure marshalling

9/20/2022 • 6 minutes to read • [Edit Online](#)

Sometimes the default marshalling rules for structures aren't exactly what you need. The .NET runtimes provide a few extension points for you to customize your structure's layout and how fields are marshalled. Customizing structure layout is supported for all scenarios, but customizing field marshalling is only supported for scenarios where runtime marshalling is enabled. If [runtime marshalling is disabled](#), then any field marshalling must be done manually.

## Customizing structure layout

.NET provides the [System.Runtime.InteropServices.StructLayoutAttribute](#) attribute and the [System.Runtime.InteropServices.LayoutKind](#) enumeration to allow you to customize how fields are placed in memory. The following guidance will help you avoid common issues.

- ✓ CONSIDER using `LayoutKind.Sequential` whenever possible.
- ✓ DO only use `LayoutKind.Explicit` in marshalling when your native struct also has an explicit layout, such as a union.
- ✗ AVOID using `LayoutKind.Explicit` when marshalling structures on non-Windows platforms if you need to target runtimes before .NET Core 3.0. The .NET Core runtime before 3.0 doesn't support passing explicit structures by value to native functions on Intel or AMD 64-bit non-Windows systems. However, the runtime supports passing explicit structures by reference on all platforms.

## Customizing Boolean field marshalling

Native code has many different Boolean representations. On Windows alone, there are three ways to represent Boolean values. The runtime doesn't know the native definition of your structure, so the best it can do is make a guess on how to marshal your Boolean values. The .NET runtime provides a way to indicate how to marshal your Boolean field. The following examples show how to marshal .NET `bool` to different native Boolean types.

Boolean values default to marshalling as a native 4-byte Win32 `BOOL` value as shown in the following example:

```
public struct WinBool
{
    public bool b;
}
```

```
struct WinBool
{
    public BOOL b;
};
```

If you want to be explicit, you can use the [UnmanagedType.Bool](#) value to get the same behavior as above:

```
public struct WinBool
{
    [MarshalAs(UnmanagedType.Bool)]
    public bool b;
}
```

```
struct WinBool
{
    public BOOL b;
};
```

Using the `UnmanagedType.U1` or `UnmanagedType.I1` values below, you can tell the runtime to marshal the `b` field as a 1-byte native `bool` type.

```
public struct CBool
{
    [MarshalAs(UnmanagedType.U1)]
    public bool b;
}
```

```
struct CBool
{
    public bool b;
};
```

On Windows, you can use the `UnmanagedType.VariantBool` value to tell the runtime to marshal your Boolean value to a 2-byte `VARIANT_BOOL` value:

```
public struct VariantBool
{
    [MarshalAs(UnmanagedType.VariantBool)]
    public bool b;
}
```

```
struct VariantBool
{
    public VARIANT_BOOL b;
};
```

#### NOTE

`VARIANT_BOOL` is different than most `bool` types in that `VARIANT_TRUE = -1` and `VARIANT_FALSE = 0`. Additionally, all values that aren't equal to `VARIANT_TRUE` are considered false.

## Customizing array field marshalling

.NET also includes a few ways to customize array marshalling.

By default, .NET marshals arrays as a pointer to a contiguous list of the elements:

```
public struct DefaultArray
{
    public int[] values;
}
```

```
struct DefaultArray
{
    int* values;
};
```

If you're interfacing with COM APIs, you may have to marshal arrays as `SAFEARRAY*` objects. You can use the [System.Runtime.InteropServices.MarshalAsAttribute](#) and the [UnmanagedType.SafeArray](#) value to tell the runtime to marshal an array as a `SAFEARRAY*`:

```
public struct SafeArrayExample
{
    [MarshalAs(UnmanagedType.SafeArray)]
    public int[] values;
}
```

```
struct SafeArrayExample
{
    SAFEARRAY* values;
};
```

If you need to customize what type of element is in the `SAFEARRAY`, then you can use the [MarshalAsAttribute.SafeArraySubType](#) and [MarshalAsAttribute.SafeArrayUserDefinedSubType](#) fields to customize the exact element type of the `SAFEARRAY`.

If you need to marshal the array in-place, you can use the [UnmanagedType.ByValArray](#) value to tell the marshaller to marshal the array in-place. When you're using this marshalling, you also must supply a value to the [MarshalAsAttribute.SizeTypeConst](#) field for the number of elements in the array so the runtime can correctly allocate space for the structure.

```
public struct InPlaceArray
{
    [MarshalAs(UnmanagedType.ByValArray, SizeConst = 4)]
    public int[] values;
}
```

```
struct InPlaceArray
{
    int values[4];
};
```

#### NOTE

.NET doesn't support marshalling a variable length array field as a C99 Flexible Array Member.

## Customizing string field marshalling

.NET also provides a wide variety of customizations for marshalling string fields.

By default, .NET marshals a string as a pointer to a null-terminated string. The encoding depends on the value of the [StructLayoutAttribute.CharSet](#) field in the [System.Runtime.InteropServices.StructLayoutAttribute](#). If no attribute is specified, the encoding defaults to an ANSI encoding.

```
[StructLayout(LayoutKind.Sequential, CharSet = CharSet.Ansi)]
public struct DefaultString
{
    public string str;
}
```

```
struct DefaultString
{
    char* str;
};
```

```
[StructLayout(LayoutKind.Sequential, CharSet = CharSet.Unicode)]
public struct DefaultString
{
    public string str;
}
```

```
struct DefaultString
{
    char16_t* str; // Could also be wchar_t* on Windows.
};
```

If you need to use different encodings for different fields or just prefer to be more explicit in your struct definition, you can use the [UnmanagedType.LPStr](#) or [UnmanagedType.LPWStr](#) values on a [System.Runtime.InteropServices.MarshalAsAttribute](#) attribute.

```
public struct AnsiString
{
    [MarshalAs(UnmanagedType.LPStr)]
    public string str;
}
```

```
struct AnsiString
{
    char* str;
};
```

```
public struct UnicodeString
{
    [MarshalAs(UnmanagedType.LPWStr)]
    public string str;
}
```

```
struct UnicodeString
{
    char16_t* str; // Could also be wchar_t* on Windows.
};
```

If you want to marshal your strings using the UTF-8 encoding, you can use the [UnmanagedType.LPUTF8Str](#) value in your [MarshalAsAttribute](#).

```
public struct UTF8String
{
    [MarshalAs(UnmanagedType.LPUTF8Str)]
    public string str;
}
```

```
struct UTF8String
{
    char* str;
};
```

#### NOTE

Using [UnmanagedType.LPUTF8Str](#) requires either .NET Framework 4.7 (or later versions) or .NET Core 1.1 (or later versions). It isn't available in .NET Standard 2.0.

If you're working with COM APIs, you may need to marshal a string as a [BSTR](#). Using the [UnmanagedType.BStr](#) value, you can marshal a string as a [BSTR](#).

```
public struct BString
{
    [MarshalAs(UnmanagedType.BStr)]
    public string str;
}
```

```
struct BString
{
    BSTR str;
};
```

When using a WinRT-based API, you may need to marshal a string as an [HSTRING](#). Using the [UnmanagedType.HString](#) value, you can marshal a string as a [HSTRING](#). [HSTRING](#) marshalling is only supported on runtimes with built-in WinRT support. WinRT support was [removed in .NET 5](#), so [HSTRING](#) marshalling is not supported in .NET 5 or newer.

```
public struct HString
{
    [MarshalAs(UnmanagedType.HString)]
    public string str;
}
```

```
struct BString
{
    HSTRING str;
};
```

If your API requires you to pass the string in-place in the structure, you can use the [UnmanagedType.ByValTStr](#) value. Do note that the encoding for a string marshalled by [ByValTstr](#) is determined from the [charset](#) attribute. Additionally, it requires that a string length is passed by the [MarshalAsAttribute.SizeConst](#) field.

```
[StructLayout(LayoutKind.Sequential, CharSet = CharSet.Ansi)]
public struct DefaultString
{
    [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 4)]
    public string str;
}
```

```
struct DefaultString
{
    char str[4];
};
```

```
[StructLayout(LayoutKind.Sequential, CharSet = CharSet.Unicode)]
public struct DefaultString
{
    [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 4)]
    public string str;
}
```

```
struct DefaultString
{
    char16_t str[4]; // Could also be wchar_t[4] on Windows.
};
```

## Customizing decimal field marshalling

If you're working on Windows, you might encounter some APIs that use the native `CY` or `CURRENCY` structure. By default, the .NET `decimal` type marshals to the native `DECIMAL` structure. However, you can use a `MarshalAsAttribute` with the `UnmanagedType.Currency` value to instruct the marshaller to convert a `decimal` value to a native `CY` value.

```
public struct Currency
{
    [MarshalAs(UnmanagedType.Currency)]
    public decimal dec;
}
```

```
struct Currency
{
    CY dec;
};
```

## Marshal `System.Object`

On Windows, you can marshal `object`-typed fields to native code. You can marshal these fields to one of three types:

- `VARIANT`
- `IUnknown*`
- `IDispatch*`

By default, an `object`-typed field will be marshalled to an `IUnknown*` that wraps the object.

```
public struct ObjectDefault
{
    public object obj;
}
```

```
struct ObjectDefault
{
    IUnknown* obj;
};
```

If you want to marshal an object field to an `IDispatch*`, add a [MarshalAsAttribute](#) with the `UnmanagedType.IDispatch` value.

```
public struct ObjectDispatch
{
    [MarshalAs(UnmanagedType.IDispatch)]
    public object obj;
}
```

```
struct ObjectDispatch
{
    IDispatch* obj;
};
```

If you want to marshal it as a `VARIANT`, add a [MarshalAsAttribute](#) with the `UnmanagedType.Struct` value.

```
public struct ObjectVariant
{
    [MarshalAs(UnmanagedType.Struct)]
    public object obj;
}
```

```
struct ObjectVariant
{
    VARIANT obj;
};
```

The following table describes how different runtime types of the `obj` field map to the various types stored in a `VARIANT`:

.NET TYPE	VARIANT TYPE
<code>byte</code>	<code>VT_UI1</code>
<code>sbyte</code>	<code>VT_I1</code>
<code>short</code>	<code>VT_I2</code>
<code>ushort</code>	<code>VT_UI2</code>
<code>int</code>	<code>VT_I4</code>

.NET TYPE	VARIANT TYPE
uint	VT_UI4
long	VT_I8
ulong	VT_UI8
float	VT_R4
double	VT_R8
char	VT_UI2
string	VT_BSTR
System.Runtime.InteropServices.BStrWrapper	VT_BSTR
object	VT_DISPATCH
System.Runtime.InteropServices.UnknownWrapper	VT_UNKNOWN
System.Runtime.InteropServices.DispatchWrapper	VT_DISPATCH
System.Reflection.Missing	VT_ERROR
(object)null	VT_EMPTY
bool	VT_BOOL
System.DateTime	VT_DATE
decimal	VT_DECIMAL
System.Runtime.InteropServices.CurrencyWrapper	VT_CURRENCY
System.DBNull	VT_NULL

# Customize parameter marshalling

9/20/2022 • 2 minutes to read • [Edit Online](#)

When the .NET runtime's default parameter marshalling behavior doesn't do what you want, you can use the [System.Runtime.InteropServices.MarshalAsAttribute](#) attribute to customize how your parameters are marshalled. These customization features do not apply when [runtime marshalling is disabled](#).

## Customizing string parameters

.NET has a variety of formats for marshalling strings. These methods are split into distinct sections on C-style strings and Windows-centric string formats.

### C-style strings

Each of these formats passes a null-terminated string to native code. They differ by the encoding of the native string.

SYSTEM.RUNTIME.INTEROPSERVICES.UNMANAGEDTYPE	VALUE	ENCODING
LPStr		ANSI
LPUTF8Str		UTF-8
LPWStr		UTF-16
LPTStr		UTF-16

The [UnmanagedType.VBByRefStr](#) format is slightly different. Like `LPWStr`, it marshals the string to a native C-style string encoded in UTF-16. However, the managed signature has you pass in the string by reference and the matching native signature takes the string by value. This distinction allows you to use a native API that takes a string by value and modifies it in-place without having to use a `StringBuilder`. We recommend against manually using this format since it's prone to cause confusion with the mismatching native and managed signatures.

### Windows-centric string formats

When interacting with COM or OLE interfaces, you'll likely find that the native functions take strings as `BSTR` arguments. You can use the [UnmanagedType.BStr](#) unmanaged type to marshal a string as a `BSTR`.

If you're interacting with WinRT APIs, you can use the [UnmanagedType.HString](#) format to marshal a string as an `HSTRING`.

## Customizing array parameters

.NET also provides you multiple ways to marshal array parameters. If you're calling an API that takes a C-style array, use the [UnmanagedType.LPArray](#) unmanaged type. If the values in the array need customized marshalling, you can use the `ArraySubType` field on the `[MarshalAs]` attribute for that.

If you're using COM APIs, you'll likely have to marshal your array parameters as `SAFEARRAY*`s. To do so, you can use the [UnmanagedType.SafeArray](#) unmanaged type. The default type of the elements of the `SAFEARRAY` can be seen in the table on [customizing object fields](#). You can use the [MarshalAsAttribute.SafeArraySubType](#) and [MarshalAsAttribute.SafeArrayUserDefinedSubType](#) fields to customize the exact element type of the `SAFEARRAY`.

# Customizing Boolean or decimal parameters

For information on marshalling Boolean or decimal parameters, see [Customizing structure marshalling](#).

## Customizing object parameters (Windows-only)

On Windows, the .NET runtime provides a number of different ways to marshal object parameters to native code.

### Marshalling as specific COM interfaces

If your API takes a pointer to a COM object, you can use any of the following `[UnmanagedType]` formats on an `object`-typed parameter to tell .NET to marshal as these specific interfaces:

- `IUnknown`
- `IDispatch`
- `IInspectable`

Additionally, if your type is marked `[ComVisible(true)]` or you're marshalling the `object` type, you can use the `UnmanagedType.Interface` format to marshal your object as a COM callable wrapper for the COM view of your type.

### Marshalling to a `VARIANT`

If your native API takes a Win32 `VARIANT`, you can use the `UnmanagedType.Struct` format on your `object` parameter to marshal your objects as `VARIANT`s. See the documentation on [customizing object fields](#) for a mapping between .NET types and `VARIANT` types.

### Custom marshallers

If you want to project a native COM interface into a different managed type, you can use the `UnmanagedType.CustomMarshaler` format and an implementation of `ICustomMarshaler` to provide your own custom marshalling code.

# Source generation for custom marshalling

9/20/2022 • 4 minutes to read • [Edit Online](#)

.NET 7 introduces a new mechanism for customization of how a type is marshalled when using source-generated interop. The [source generator for P/Invokes](#) recognizes [MarshalUsingAttribute](#) and [NativeMarshallingAttribute](#) as indicators for custom marshalling of a type.

[NativeMarshallingAttribute](#) can be applied to a type to indicate the default custom marshalling for that type. The [MarshalUsingAttribute](#) can be applied to a parameter or return value to indicate the custom marshalling for that particular usage of the type, taking precedence over any [NativeMarshallingAttribute](#) that may be on the type itself. Both of these attributes expect a [Type](#)—the entry-point marshaller type—that's marked with one or more [CustomMarshallerAttribute](#) attributes. Each [CustomMarshallerAttribute](#) indicates which marshaller implementation should be used to marshal the specified managed type for the specified [MarshalMode](#).

## Marshaller implementation

Marshaller implementations can either be stateless or stateful. If the marshaller type is a `static` class, it's considered stateless. If it's a value type, it's considered stateful and one instance of that marshaller will be used to marshal a specific parameter or return value. Different [shapes for the marshaller implementation](#) are expected based on whether a marshaller is stateless or stateful and whether it supports marshalling from managed to unmanaged, unmanaged to managed, or both. The .NET SDK includes analyzers and code fixers to help with implementing marshallers that conform to the require shapes.

### MarshalMode

The [MarshalMode](#) specified in a [CustomMarshallerAttribute](#) determines the expected marshalling support and [shape](#) for the marshaller implementation. All modes support stateless marshaller implementations. Element marshalling modes do not support stateful marshaller implementations.

MARSHALMODE	EXPECTED SUPPORT	CAN BE STATEFUL
ManagedToUnmanagedIn	Managed to unmanaged	Yes
ManagedToUnmanagedRef	Managed to unmanaged and unmanaged to managed	Yes
ManagedToUnmanagedOut	Unmanaged to managed	Yes
UnmanagedTypeIn	Unmanaged to managed	Yes
UnmanagedTypeRefOut	Managed to unmanaged and unmanaged to managed	Yes
UnmanagedTypeOut	Managed to unmanaged	Yes
ElementIn	Managed to unmanaged	No
ElementRef	Managed to unmanaged and unmanaged to managed	No
ElementOut	Unmanaged to managed	No

`MarshalMode.Default` indicates that the marshaller implementation should be used for any mode that it supports. If a marshaller implementation for a more specific `MarshalMode` is also specified, it takes precedence over `MarshalMode.Default`.

## Basic usage

We can specify [NativeMarshallingAttribute](#) on a type, pointing at an entry-point marshaller type that is either a

`static` class or a `struct`.

```
[NativeMarshalling(typeof(ExampleMarshaller))]
public struct Example
{
    public string Message;
    public int Flags;
}
```

`ExampleMarshaller`, the entry-point marshaller type, is marked with [CustomMarshallerAttribute](#), pointing at a [marshaller implementation](#) type. In this example, `ExampleMarshaller` is both the entry point and the implementation. It conforms to [marshaller shapes](#) expected for custom marshalling of a value.

```
[CustomMarshaller(typeof(Example), MarshalMode.Default, typeof(ExampleMarshaller))]
internal static class ExampleMarshaller
{
    public static ExampleUnmanaged ConvertToUnmanaged(Example managed)
        => throw new NotImplementedException();

    public static Example ConvertToManaged(ExampleUnmanaged unmanaged)
        => throw new NotImplementedException();

    public static void Free(ExampleUnmanaged unmanaged)
        => throw new NotImplementedException();

    internal struct ExampleUnmanaged
    {
        public IntPtr Message;
        public int Flags;
    }
}
```

The `ExampleMarshaller` in the example is a stateless marshaller that implements support for marshalling from managed to unmanaged and from unmanaged to managed. The marshalling logic is entirely controlled by your marshaller implementation. Marking fields on a struct with [MarshalAsAttribute](#) has no effect on the generated code.

The `Example` type can then be used in P/Invoke source generation. In the following P/Invoke example, `ExampleMarshaller` will be used to marshal the parameter from managed to unmanaged. It will also be used to marshal the return value from unmanaged to managed.

```
[LibraryImport("nativelib")]
internal static partial Example ConvertExample(Example example);
```

To use a different marshaller for a specific usage of the `Example` type, specify [MarshalUsingAttribute](#) at the use site. In the following P/Invoke example, `ExampleMarshaller` will be used to marshal the parameter from managed to unmanaged. `OtherExampleMarshaller` will be used to marshal the return value from unmanaged to managed.

```
[LibraryImport("nativelib")]
[return: MarshalUsing(typeof(OtherExampleMarshaller))]
internal static partial Example ConvertExample(Example example);
```

## Collections

Apply the [ContiguousCollectionMarshallerAttribute](#) to a marshaller entry-point type to indicate that it's for contiguous collections. The type must have one more type parameter than the associated managed type. The last type parameter is a placeholder and will be filled in by the source generator with the unmanaged type for the collection's element type.

For example, you can specify custom marshalling for a `List<T>`. In the following code, `ListMarshaller` is both the entry point and the implementation. It conforms to [marshaller shapes](#) expected for custom marshalling of a collection.

```
[ContiguousCollectionMarshaller]
[CustomMarshaller(typeof(List<T>), MarshalMode.Default, typeof(ListMarshaller<,>))]
public unsafe static class ListMarshaller<T, TUnmanagedElement> where TUnmanagedElement : unmanaged
{
    public static byte* AllocateContainerForUnmanagedElements(List<T> managed, out int numElements)
        => throw new NotImplementedException();

    public static ReadOnlySpan<T> GetManagedValuesSource(List<T> managed)
        => throw new NotImplementedException();

    public static Span<TUnmanagedElement> GetUnmanagedValuesDestination(byte* unmanaged, int numElements)
        => throw new NotImplementedException();

    public static List<T> AllocateContainerForManagedElements(byte* unmanaged, int length)
        => throw new NotImplementedException();

    public static Span<T> GetManagedValuesDestination(List<T> managed)
        => throw new NotImplementedException();

    public static ReadOnlySpan<TUnmanagedElement> GetUnmanagedValuesSource(byte* nativeValue, int numElements)
        => throw new NotImplementedException();

    public static void Free(byte* unmanaged)
        => throw new NotImplementedException();
}
```

The `ListMarshaller` in the example is a stateless collection marshaller that implements support for marshalling from managed to unmanaged and from unmanaged to managed for a `List<T>`. In the following P/Invoke example, `ListMarshaller` will be used to marshal the parameter from managed to unmanaged and to marshal the return value from unmanaged to managed. `CountElementName` indicates that the `numValues` parameter should be used as the element count when marshalling the return value from unmanaged to managed.

```
[LibraryImport("nativelib")]
[return: MarshalUsing(typeof(ListMarshaller<,>), CountElementName = "numValues")]
internal static partial void ConvertList(
    [MarshalUsing(typeof(ListMarshaller<,>))] List<int> list,
    out int numValues);
```

## See also

- [System.Runtime.InteropServices.Marshalling APIs](#)
- [P/Invoke source generation](#)

- Disabling runtime marshalling
- Marshaller shapes
  - [Values](#)
  - [Collections](#)

# Native interoperability best practices

9/20/2022 • 13 minutes to read • [Edit Online](#)

.NET gives you various ways to customize your native interoperability code. This article includes the guidance that Microsoft's .NET teams follow for native interoperability.

## General guidance

The guidance in this section applies to all interop scenarios.

- ✓ DO use the same naming and capitalization for your methods and parameters as the native method you want to call.
- ✓ CONSIDER using the same naming and capitalization for constant values.
- ✓ DO use .NET types that map closest to the native type. For example, in C#, use `uint` when the native type is `unsigned int`.
- ✓ DO only use `[In]` and `[Out]` attributes when the behavior you want differs from the default behavior.
- ✓ CONSIDER using `System.Buffers.ArrayPool<T>` to pool your native array buffers.
- ✓ CONSIDER wrapping your P/Invoke declarations in a class with the same name and capitalization as your native library.
  - This allows your `[DllImport]` attributes to use the C# `nameof` language feature to pass in the name of the native library and ensure that you didn't misspell the name of the native library.

## DllImport attribute settings

SETTING	DEFAULT	RECOMMENDATION	DETAILS
<code>PreserveSig</code>	<code>true</code>	Keep default	When this is explicitly set to false, failed HRESULT return values will be turned into exceptions (and the return value in the definition becomes null as a result).
<code>SetLastError</code>	<code>false</code>	Depends on the API	Set this to true if the API uses <code>GetLastError</code> and use <code>Marshal.GetLastWin32Error</code> to get the value. If the API sets a condition that says it has an error, get the error before making other calls to avoid inadvertently having it overwritten.

SETTING	DEFAULT	RECOMMENDATION	DETAILS
CharSet	Compiler-defined (specified in the <a href="#">charset documentation</a> )	Explicitly use <code>CharSet.Unicode</code> or <code>CharSet.Ansi</code> when strings or characters are present in the definition	This specifies marshalling behavior of strings and what <code>ExactSpelling</code> does when <code>false</code> . Note that <code>CharSet.Ansi</code> is actually UTF8 on Unix. <i>Most</i> of the time Windows uses Unicode while Unix uses UTF8. See more information on the <a href="#">documentation on charsets</a> .
ExactSpelling	<code>false</code>	<code>true</code>	Set this to true and gain a slight perf benefit as the runtime will not look for alternate function names with either an "A" or "W" suffix depending on the value of the <code>CharSet</code> setting ("A" for <code>CharSet.Ansi</code> and "W" for <code>CharSet.Unicode</code> ).

## String parameters

When the CharSet is Unicode or the argument is explicitly marked as `[MarshalAs(UnmanagedType.LPWSTR)]` and the string is passed by value (not `ref` or `out`), the string will be pinned and used directly by native code (rather than copied).

✗ DO NOT use `[Out] string` parameters. String parameters passed by value with the `[Out]` attribute can destabilize the runtime if the string is an interned string. See more information about string interning in the documentation for [String.Intern](#).

- ✓ CONSIDER setting the `CharSet` property in `[DllImport]` so the runtime knows the expected string encoding.
- ✓ CONSIDER `char[]` or `byte[]` arrays from an `ArrayPool` when native code is expected to fill a character buffer. This requires passing the argument as `[Out]`.
- ✓ CONSIDER avoiding `StringBuilder` parameters. `StringBuilder` marshalling *always* creates a native buffer copy. As such, it can be extremely inefficient. Take the typical scenario of calling a Windows API that takes a string:

1. Create a `StringBuilder` of the desired capacity (allocates managed capacity) {1}.
2. Invoke:
  - a. Allocates a native buffer {2}.
  - b. Copies the contents if `[In]` (*the default for a `StringBuilder` parameter*).
  - c. Copies the native buffer into a newly allocated managed array if `[out]` {3} (*also the default for `StringBuilder`*).
3. `ToString()` allocates yet another managed array {4}.

That's {4} allocations to get a string out of native code. The best you can do to limit this is to reuse the `StringBuilder` in another call, but this still only saves one allocation. It's much better to use and cache a character buffer from `ArrayPool`. You can then get down to just the allocation for the `ToString()` on subsequent calls.

The other issue with `StringBuilder` is that it always copies the return buffer back up to the first null. If the passed back string isn't terminated or is a double-null-terminated string, your P/Invoke is incorrect at best.

If you *do* use `StringBuilder`, one last gotcha is that the capacity does **not** include a hidden null, which is always accounted for in interop. It's common for people to get this wrong as most APIs want the size of the buffer *including* the null. This can result in wasted/unnecessary allocations. Additionally, this gotcha prevents the runtime from optimizing `StringBuilder` marshalling to minimize copies.

For more information on string marshalling, see [Default Marshalling for Strings](#) and [Customizing string marshalling](#).

**Windows Specific** For `[out]` strings the CLR will use `CoTaskMemFree` by default to free strings or `SysStringFree` for strings that are marked as `UnmanagedType.BSTR`. **For most APIs with an output string buffer:** The passed in character count must include the null. If the returned value is less than the passed in character count the call has succeeded and the value is the number of characters *without* the trailing null. Otherwise the count is the required size of the buffer *including* the null character.

- Pass in 5, get 4: The string is 4 characters long with a trailing null.
- Pass in 5, get 6: The string is 5 characters long, need a 6 character buffer to hold the null. [Windows Data Types for Strings](#)

## Boolean parameters and fields

Booleans are easy to mess up. By default, a .NET `bool` is marshalled to a Windows `BOOL`, where it's a 4-byte value. However, the `_Bool`, and `bool` types in C and C++ are a *single* byte. This can lead to hard to track down bugs as half the return value will be discarded, which will only *potentially* change the result. For more for information on marshalling .NET `bool` values to C or C++ `bool` types, see the documentation on [customizing boolean field marshalling](#).

## GUIDs

GUIDs are usable directly in signatures. Many Windows APIs take `GUID&` type aliases like `REFIID`. When passed by ref, they can either be passed by `ref` or with the `[MarshalAs(UnmanagedType.LPStruct)]` attribute.

GUID	BY-REF GUID
<code>KNOWNFOLDERID</code>	<code>REFKNOWNFOLDERID</code>

✗ DO NOT Use `[MarshalAs(UnmanagedType.LPStruct)]` for anything other than `ref` GUID parameters.

## Blittable types

Blittable types are types that have the same bit-level representation in managed and native code. As such they do not need to be converted to another format to be marshalled to and from native code, and as this improves performance they should be preferred. Some types are not blittable but are known to contain blittable contents. These types have similar optimizations as blittable types when they are not contained in another type, but are not considered blittable when in fields of structs or for the purposes of [UnmanagedCallersOnlyAttribute](#).

### Blittable types when runtime marshalling is enabled

Blittable types:

- `byte`, `sbyte`, `short`, `ushort`, `int`, `uint`, `long`, `ulong`, `single`, `double`
- structs with fixed layout that only have blittable value types for instance fields
  - fixed layout requires `[StructLayout(LayoutKind.Sequential)]` or `[StructLayout(LayoutKind.Explicit)]`

- o structs are `LayoutKind.Sequential` by default

### Types with blittable contents:

- non-nested, one-dimensional arrays of blittable primitive types (for example, `int[]`)
- classes with fixed layout that only have blittable value types for instance fields
  - o fixed layout requires `[StructLayout(LayoutKind.Sequential)]` or `[StructLayout(LayoutKind.Explicit)]`
  - o classes are `LayoutKind.Auto` by default

### NOT blittable:

- `bool`

### SOMETIMES blittable:

- `char`

### Types with SOMETIMES blittable contents:

- `string`

When blittable types are passed by reference with `in`, `ref`, or `out`, or when types with blittable contents are passed by value, they're simply pinned by the marshaller instead of being copied to an intermediate buffer.

`char` is blittable in a one-dimensional array **or** if it's part of a type that contains it's explicitly marked with `[StructLayout]` with `CharSet = CharSet.Unicode`.

```
[StructLayout(LayoutKind.Sequential, CharSet = CharSet.Unicode)]
public struct UnicodeCharStruct
{
    public char c;
}
```

`string` contains blittable contents if it isn't contained in another type and it's being passed as an argument that is marked with `[MarshalAs(UnmanagedType.LPWStr)]` or the `[DllImport]` has `CharSet = CharSet.Unicode` set.

You can see if a type is blittable or contains blittable contents by attempting to create a pinned `GCHandle`. If the type isn't a string or considered blittable, `GCHandle.Alloc` will throw an `ArgumentException`.

### Blittable types when runtime marshalling is disabled

When [runtime marshalling is disabled](#), the rules for which types are blittable are significantly simpler. All types that are C# `unmanaged` types and don't have any fields that are marked with `[StructLayout(LayoutKind.Auto)]` are blittable. All types that are not C# `unmanaged` types are not blittable. The concept of types with blittable contents, such as arrays or strings, does not apply when runtime marshalling is disabled. Any type that is not considered blittable by the aforementioned rule is unsupported when runtime marshalling is disabled.

These rules differ from the built-in system primarily in situations where `bool` and `char` are used. When marshalling is disabled, `bool` is passed as a 1-byte value and not normalized and `char` is always passed as a 2-byte value. When runtime marshalling is enabled, `bool` can map to a 1, 2, or 4-byte value and is always normalized, and `char` maps to either a 1 or 2-byte value depending on the `CharSet`.

- ✓ DO make your structures blittable when possible.

For more information, see:

- [Blittable and Non-Blittable Types](#)
- [Type Marshalling](#)

# Keeping managed objects alive

`GC.KeepAlive()` will ensure an object stays in scope until the `KeepAlive` method is hit.

`HandleRef` allows the marshaller to keep an object alive for the duration of a P/Invoke. It can be used instead of `IntPtr` in method signatures. `SafeHandle` effectively replaces this class and should be used instead.

`GCHandle` allows pinning a managed object and getting the native pointer to it. The basic pattern is:

```
GCHandle handle = GCHandle.Alloc(obj, GCHandleType.Pinned);  
IntPtr ptr = handle.AddrOfPinnedObject();  
handle.Free();
```

Pinning isn't the default for `GCHandle`. The other major pattern is for passing a reference to a managed object through native code and back to managed code, usually with a callback. Here is the pattern:

```
GCHandle handle = GCHandle.Alloc(obj);  
SomeNativeEnumerator(callbackDelegate, GCHandle.ToIntPtr(handle));  
  
// In the callback  
GCHandle handle = GCHandle.FromIntPtr(param);  
object managedObject = handle.Target;  
  
// After the last callback  
handle.Free();
```

Don't forget that `GCHandle` needs to be explicitly freed to avoid memory leaks.

## Common Windows data types

Here is a list of data types commonly used in Windows APIs and which C# types to use when calling into the Windows code.

The following types are the same size on 32-bit and 64-bit Windows, despite their names.

WIDTH	WINDOWS	C#	ALTERNATIVE
32	<code>BOOL</code>	<code>int</code>	<code>bool</code>
8	<code>BOOLEAN</code>	<code>byte</code>	<code>[MarshalAs(UnmanagedType.U1)] bool</code>
8	<code>BYTE</code>	<code>byte</code>	
8	<code>CHAR</code>	<code>sbyte</code>	
8	<code>UCHAR</code>	<code>byte</code>	
16	<code>SHORT</code>	<code>short</code>	
16	<code>CSHORT</code>	<code>short</code>	
16	<code>USHORT</code>	<code>ushort</code>	
16	<code>WORD</code>	<code>ushort</code>	

WIDTH	WINDOWS	C#	ALTERNATIVE
16	ATOM	ushort	
32	INT	int	
32	LONG	int	See <a href="#">CLong</a> and <a href="#">CULong</a> .
32	ULONG	uint	See <a href="#">CLong</a> and <a href="#">CULong</a> .
32	DWORD	uint	
64	QWORD	long	
64	LARGE_INTEGER	long	
64	LONGLONG	long	
64	ULLONGLONG	ulong	
64	ULARGE_INTEGER	ulong	
32	HRESULT	int	
32	NTSTATUS	int	

The following types, being pointers, do follow the width of the platform. Use `IntPtr` / `UIntPtr` for these.

SIGNED POINTER TYPES (USE <code>IntPtr</code> )	UNSIGNED POINTER TYPES (USE <code>UIntPtr</code> )
HANDLE	WPARAM
HWND	UINT_PTR
HINSTANCE	ULONG_PTR
LPARAM	SIZE_T
LRESULT	
LONG_PTR	
INT_PTR	

A Windows `PVOID`, which is a C `void*`, can be marshalled as either `IntPtr` or `UIntPtr`, but prefer `void*` when possible.

## Windows Data Types

### Data Type Ranges

### Formerly built-in supported types

There are rare instances when built-in support for a type is removed.

The `UnmanagedType.HString` built-in marshal support was removed in the .NET 5 release. You must recompile binaries that use this marshalling type and that target a previous framework. It's still possible to marshal this type, but you must marshal it manually, as the following code example shows. This code will work moving forward and is also compatible with previous frameworks.

```
static class HSTRING
{
    public static IntPtr FromString(string s)
    {
        Marshal.ThrowExceptionForHR(WindowsCreateString(s, s.Length, out IntPtr h));
        return h;
    }

    public static void Delete(IntPtr s)
    {
        Marshal.ThrowExceptionForHR(WindowsDeleteString(s));
    }

    [DllImport("api-ms-win-core-winrt-string-11-1-0.dll", CallingConvention = CallingConvention.StdCall,
ExactSpelling = true)]
    private static extern int WindowsCreateString(
        [MarshalAs(UnmanagedType.LPWStr)] string sourceString, int length, out IntPtr hstring);

    [DllImport("api-ms-win-core-winrt-string-11-1-0.dll", CallingConvention = CallingConvention.StdCall,
ExactSpelling = true)]
    private static extern int WindowsDeleteString(IntPtr hstring);
}

// Usage example
IntPtr hstring = HSTRING.FromString("HSTRING from .NET to WinRT API");
try
{
    // Pass hstring to WinRT or Win32 API.
}
finally
{
    HSTRING.Delete(hstring);
}
```

## Cross-platform data type considerations

There are types in the C/C++ language that have latitude in how they are defined. When writing cross-platform interop, cases can arise where platforms differ and can cause issues if not considered.

**C/C++** `long`

C/C++ `long` and C# `long` are not the same types. Using C# `long` to interop with C/C++ `long` is almost never correct.

The `long` type in C/C++ is defined to have "at least 32" bits. This means there is a minimum number of required bits, but platforms can choose to use more bits if desired. The following table illustrates the differences in provided bits for the C/C++ `long` data type between platforms.

PLATFORM	32-BIT	64-BIT
Windows	32	32
macOS/*nix	32	64

These differences can make authoring cross-platform P/Invokes difficult when the native function is defined to use `long` on all platforms.

In .NET 6 and later versions, use the `CLong` and `CULong` types for interop with C/C++ `long` and `unsigned long` data types. The following example is for `CLong`, but you can use `ulong` to abstract `unsigned long` in a similar way.

```
// Cross platform C function
// long Function(long a);
[DllImport("NativeLib")]
extern static CLong Function(CLong a);

// Usage
nint result = Function(new CLong(10)).Value;
```

When targeting .NET 5 and earlier versions, you should declare separate Windows and non-Windows signatures to handle the problem.

```
static readonly bool IsWindows = RuntimeInformation.IsOSPlatform(OSPlatform.Windows);

// Cross platform C function
// long Function(long a);

[DllImport("NativeLib", EntryPoint = "Function")]
extern static int FunctionWindows(int a);

[DllImport("NativeLib", EntryPoint = "Function")]
extern static nint FunctionUnix(nint a);

// Usage
nint result;
if (IsWindows)
{
    result = FunctionWindows(10);
}
else
{
    result = FunctionUnix(10);
}
```

## Structs

Managed structs are created on the stack and aren't removed until the method returns. By definition then, they are "pinned" (it won't get moved by the GC). You can also simply take the address in unsafe code blocks if native code won't use the pointer past the end of the current method.

Blittable structs are much more performant as they can simply be used directly by the marshalling layer. Try to make structs blittable (for example, avoid `bool`). For more information, see the [Blittable Types](#) section.

If the struct is blittable, use `sizeof()` instead of `Marshal.SizeOf<MyStruct>()` for better performance. As mentioned above, you can validate that the type is blittable by attempting to create a pinned `GCHandle`. If the type is not a string or considered blittable, `GCHandle.Alloc` will throw an `ArgumentException`.

Pointers to structs in definitions must either be passed by `ref` or use `unsafe` and `*`.

✓ DO match the managed struct as closely as possible to the shape and names that are used in the official platform documentation or header.

✓ DO use the C# `sizeof()` instead of `Marshal.SizeOf<MyStruct>()` for blittable structures to improve

performance.

✗ AVOID using `System.Delegate` or `System.MulticastDelegate` fields to represent function pointer fields in structures.

Since `System.Delegate` and `System.MulticastDelegate` don't have a required signature, they don't guarantee that the delegate passed in will match the signature the native code expects. Additionally, in .NET Framework and .NET Core, marshalling a struct containing a `System.Delegate` or `System.MulticastDelegate` from its native representation to a managed object can destabilize the runtime if the value of the field in the native representation isn't a function pointer that wraps a managed delegate. In .NET 5 and later versions, marshalling a `System.Delegate` or `System.MulticastDelegate` field from a native representation to a managed object is not supported. Use a specific delegate type instead of `System.Delegate` or `System.MulticastDelegate`.

## Fixed Buffers

An array like `INT_PTR Reserved1[2]` has to be marshalled to two `IntPtr` fields, `Reserved1a` and `Reserved1b`. When the native array is a primitive type, we can use the `fixed` keyword to write it a little more cleanly. For example, `SYSTEM_PROCESS_INFORMATION` looks like this in the native header:

```
typedef struct _SYSTEM_PROCESS_INFORMATION {
    ULONG NextEntryOffset;
    ULONG NumberOfThreads;
    BYTE Reserved1[48];
    UNICODE_STRING ImageName;
    ...
} SYSTEM_PROCESS_INFORMATION
```

In C#, we can write it like this:

```
internal unsafe struct SYSTEM_PROCESS_INFORMATION
{
    internal uint NextEntryOffset;
    internal uint NumberOfThreads;
    private fixed byte Reserved1[48];
    internal Interop.UNICODE_STRING ImageName;
    ...
}
```

However, there are some gotchas with fixed buffers. Fixed buffers of non-blittable types won't be correctly marshalled, so the in-place array needs to be expanded out to multiple individual fields. Additionally, in .NET Framework and .NET Core before 3.0, if a struct containing a fixed buffer field is nested within a non-blittable struct, the fixed buffer field won't be correctly marshalled to native code.

# Expose .NET Core components to COM

9/20/2022 • 4 minutes to read • [Edit Online](#)

This article walks you through how to expose a class to COM from .NET Core (or .NET 5+). This tutorial shows you how to:

- Expose a class to COM from .NET Core.
- Generate a COM server as part of building your .NET Core library.
- Automatically generate a side-by-side server manifest for Registry-Free COM.

## Prerequisites

- Install [.NET Core 3.0 SDK](#) or a newer version.

## Create the library

The first step is to create the library.

1. Create a new folder, and in that folder run the following command:

```
dotnet new classlib
```

2. Open `Class1.cs`.

3. Add `using System.Runtime.InteropServices;` to the top of the file.

4. Create an interface named `IInterface`. For example:

```
using System;
using System.Runtime.InteropServices;

[ComVisible(true)]
[Guid(ContractGuids.ServerInterface)]
[InterfaceType(ComInterfaceType.InterfaceIsIUnknown)]
public interface IInterface
{
    /// <summary>
    /// Compute the value of the constant Pi.
    /// </summary>
    double ComputePi();
}
```

5. Add the `[Guid("<IID>")]` attribute to the interface, with the interface GUID for the COM interface you're implementing. For example, `[Guid("fe103d6e-e71b-414c-80bf-982f18f6c1c7")]`. Note that this GUID needs to be unique since it is the only identifier of this interface for COM. In Visual Studio, you can generate a GUID by going to Tools > Create GUID to open the Create GUID tool.

6. Add the `[InterfaceType]` attribute to the interface and specify what base COM interfaces your interface should implement.

7. Create a class named `Server` that implements `IInterface`.

8. Add the `[Guid("<CLSID>")]` attribute to the class, with the class identifier GUID for the COM class you're

implementing. For example, `[Guid("9f35b6f5-2c05-4e7f-93aa-ee087f6e7ab6")]`. As with the interface GUID, this GUID must be unique since it is the only identifier of this interface to COM.

9. Add the `[ComVisible(true)]` attribute to both the interface and the class.

#### IMPORTANT

Unlike in .NET Framework, .NET Core requires you to specify the CLSID of any class you want to be activatable via COM.

## Generate the COM host

1. Open the `.csproj` project file and add `<EnableComHosting>true</EnableComHosting>` inside a `<PropertyGroup></PropertyGroup>` tag.
2. Build the project.

The resulting output will have a `ProjectName.dll`, `ProjectName.deps.json`, `ProjectName.runtimeconfig.json` and `ProjectName.comhost.dll` file.

## Register the COM host for COM

Open an elevated command prompt and run `regsvr32 ProjectName.comhost.dll`. That will register all of your exposed .NET objects with COM.

## Enabling RegFree COM

1. Open the `.csproj` project file and add `<EnableRegFreeCom>true</EnableRegFreeCom>` inside a `<PropertyGroup></PropertyGroup>` tag.
2. Build the project.

The resulting output will now also have a `ProjectName.X.manifest` file. This file is the side-by-side manifest for use with Registry-Free COM.

## Embedding type libraries in the COM host

Unlike in .NET Framework, there is no support in .NET Core or .NET 5+ for generating a [COM Type Library \(TLB\)](#) from a .NET assembly. The guidance is to either manually write an IDL file or a C/C++ header for the native declarations of the COM interfaces. If you decide to write an IDL file, you can compile it with the Visual C++ SDK's MIDL compiler to produce a TLB.

In .NET 6 and later versions, the .NET SDK supports embedding already-compiled TLBs into the COM host as part of your project build.

To embed a type library into your application, follow these steps:

1. Open the `.csproj` project file and add `<ComHostTypeLibrary Include="path/to/typelib.tlb" Id="<id>" />` inside an `<ItemGroup></ItemGroup>` tag.
2. Replace `<id>` with a positive integer value. The value must be unique among the TLBs you specify to be embedded in the COM host.
  - The `Id` attribute is optional if you only add one `ComHostTypeLibrary` to your project.

For example, the following code block adds the `Server.tlb` type library at index `1` to the COM host:

```
<ItemGroup>
  <ComHostTypeLibrary Include="Server.tlb" Id="1" />
</ItemGroup>
```

## Sample

There is a fully functional [COM server sample](#) in the dotnet/samples repository on GitHub.

## Additional notes

### IMPORTANT

In .NET Framework, an "Any CPU" assembly can be consumed by both 32-bit and 64-bit clients. By default, in .NET Core, .NET 5, and later versions, "Any CPU" assemblies are accompanied by a 64-bit *.comhost.dll. Because of this, they can only be consumed by 64-bit clients. That is the default because that is what the SDK represents. This behavior is identical to how the "self-contained" feature is published: by default it uses what the SDK provides. The `NETCoreSdkRuntimeIdentifier` MSBuild property determines the bitness of *.comhost.dll. The managed part is actually bitness agnostic as expected, but the accompanying native asset defaults to the targeted SDK.

During activation, the assembly containing the COM component is loaded in a separate `AssemblyLoadContext` based on the assembly path. If there is one assembly providing multiple COM servers, the `AssemblyLoadContext` is reused such that all of the servers from that assembly reside in the same load context. If there are multiple assemblies providing COM servers, a new `AssemblyLoadContext` is created for each assembly, and each server resides in the load context that corresponds to its assembly.

[Self-contained deployments](#) of COM components are not supported. Only [framework-dependent deployments](#) of COM components are supported.

Additionally, loading both .NET Framework and .NET Core into the same process does have diagnostic limitations. The primary limitation is the debugging of managed components as it is not possible to debug both .NET Framework and .NET Core at the same time. In addition, the two runtime instances don't share managed assemblies. This means that it isn't possible to share actual .NET types across the two runtimes and instead all interactions must be restricted to the exposed COM interface contracts.

# Write a custom .NET host to control the .NET runtime from your native code

9/20/2022 • 4 minutes to read • [Edit Online](#)

Like all managed code, .NET applications are executed by a host. The host is responsible for starting the runtime (including components like the JIT and garbage collector) and invoking managed entry points.

Hosting the .NET runtime is an advanced scenario and, in most cases, .NET developers don't need to worry about hosting because .NET build processes provide a default host to run .NET applications. In some specialized circumstances, though, it can be useful to explicitly host the .NET runtime, either as a means of invoking managed code in a native process or in order to gain more control over how the runtime works.

This article gives an overview of the steps necessary to start the .NET runtime from native code and execute managed code in it.

## Prerequisites

Because hosts are native applications, this tutorial covers constructing a C++ application to host .NET. You will need a C++ development environment (such as that provided by [Visual Studio](#)).

You will also need to build a .NET component to test the host with, so you should install the [.NET SDK](#).

## Hosting APIs

Hosting the .NET runtime in .NET Core 3.0 and above is done with the `nethost` and `hostfxr` libraries' APIs. These entry points handle the complexity of finding and setting up the runtime for initialization and allow both launching a managed application and calling into a static managed method.

Prior to .NET Core 3.0, the only option for hosting the runtime was through the `coreclrhost.h` API. This hosting API is obsolete now and should not be used for hosting .NET Core 3.0 and higher runtimes.

## Create a host using `nethost.h` and `hostfxr.h`

A [sample host](#) demonstrating the steps outlined in the tutorial below is available in the `dotnet/samples` GitHub repository. Comments in the sample clearly associate the numbered steps from this tutorial with where they're performed in the sample. For download instructions, see [Samples and Tutorials](#).

Keep in mind that the sample host is meant to be used for learning purposes, so it is light on error checking and designed to emphasize readability over efficiency.

The following steps detail how to use the `nethost` and `hostfxr` libraries to start the .NET runtime in a native application and call into a managed static method. The [sample](#) uses the `nethost` header and library installed with the .NET SDK and copies of the `coreclr_delegates.h` and `hostfxr.h` files from the `dotnet/runtime` repository.

### Step 1 - Load `hostfxr` and get exported hosting functions

The `nethost` library provides the `get_hostfxr_path` function for locating the `hostfxr` library. The `hostfxr` library exposes functions for hosting the .NET runtime. The full list of functions can be found in `hostfxr.h` and the [native hosting design document](#). The sample and this tutorial use the following:

- `hostfxr_initialize_for_runtime_config` : Initializes a host context and prepares for initialization of the .NET

runtime using the specified runtime configuration.

- `hostfxr_get_runtime_delegate` : Gets a delegate for runtime functionality.
- `hostfxr_close` : Closes a host context.

The `hostfxr` library is found using `get_hostfxr_path` API from `nethost` library. It is then loaded and its exports are retrieved.

```
// Using the nethost library, discover the location of hostfxr and get exports
bool load_hostfxr()
{
    // Pre-allocate a large buffer for the path to hostfxr
    char_t buffer[MAX_PATH];
    size_t buffer_size = sizeof(buffer) / sizeof(char_t);
    int rc = get_hostfxr_path(buffer, &buffer_size, nullptr);
    if (rc != 0)
        return false;

    // Load hostfxr and get desired exports
    void *lib = load_library(buffer);
    init_fptr = (hostfxr_initialize_for_runtime_config_fn) get_export(lib,
"hostfxr_initialize_for_runtime_config");
    get_delegate_fptr = (hostfxr_get_runtime_delegate_fn) get_export(lib, "hostfxr_get_runtime_delegate");
    close_fptr = (hostfxr_close_fn) get_export(lib, "hostfxr_close");

    return (init_fptr && get_delegate_fptr && close_fptr);
}
```

## Step 2 - Initialize and start the .NET runtime

The `hostfxr_initialize_for_runtime_config` and `hostfxr_get_runtime_delegate` functions initialize and start the .NET runtime using the runtime configuration for the managed component that will be loaded. The `hostfxr_get_runtime_delegate` function is used to get a runtime delegate that allows loading a managed assembly and getting a function pointer to a static method in that assembly.

```
// Load and initialize .NET Core and get desired function pointer for scenario
load_assembly_and_get_function_pointer_fn get_dotnet_load_assembly(const char_t *config_path)
{
    // Load .NET Core
    void *load_assembly_and_get_function_pointer = nullptr;
    hostfxr_handle ctxt = nullptr;
    int rc = init_fptr(config_path, nullptr, &ctxt);
    if (rc != 0 || ctxt == nullptr)
    {
        std::cerr << "Init failed: " << std::hex << std::showbase << rc << std::endl;
        close_fptr(ctxt);
        return nullptr;
    }

    // Get the load assembly function pointer
    rc = get_delegate_fptr(
        ctxt,
        hdt_load_assembly_and_get_function_pointer,
        &load_assembly_and_get_function_pointer);
    if (rc != 0 || load_assembly_and_get_function_pointer == nullptr)
        std::cerr << "Get delegate failed: " << std::hex << std::showbase << rc << std::endl;

    close_fptr(ctxt);
    return (load_assembly_and_get_function_pointer_fn)load_assembly_and_get_function_pointer;
}
```

## Step 3 - Load managed assembly and get function pointer to a managed method

The runtime delegate is called to load the managed assembly and get a function pointer to a managed method.

The delegate requires the assembly path, type name, and method name as inputs and returns a function pointer that can be used to invoke the managed method.

```
// Function pointer to managed delegate
component_entry_point_fn hello = nullptr;
int rc = load_assembly_and_get_function_pointer(
    dotnetlib_path.c_str(),
    dotnet_type,
    dotnet_type_method,
    nullptr /*delegate_type_name*/,
    nullptr,
    (void**)&hello);
```

By passing `nullptr` as the delegate type name when calling the runtime delegate, the sample uses a default signature for the managed method:

```
public delegate int ComponentEntryPoint(IntPtr args, int sizeBytes);
```

A different signature can be used by specifying the delegate type name when calling the runtime delegate.

#### Step 4 - Run managed code!

The native host can now call the managed method and pass it the desired parameters.

```
lib_args args
{
    STR("from host!"),
    i
};

hello(&args, sizeof(args));
```

# COM Interop in .NET

9/20/2022 • 2 minutes to read • [Edit Online](#)

The Component Object Model (COM) lets an object expose its functionality to other components and to host applications on Windows platforms. To help enable users to interoperate with their existing code bases, .NET Framework has always provided strong support for interoperating with COM libraries. In .NET Core 3.0, a large portion of this support has been added to .NET Core on Windows. The documentation here explains how the common COM interop technologies work and how you can utilize them to interoperate with your existing COM libraries.

- [COM Wrappers](#)
- [COM Callable Wrappers](#)
- [Runtime Callable Wrappers](#)
- [Qualifying .NET Types for COM Interoperation](#)
- [Trimmer and Native AOT-friendly COM interop](#)

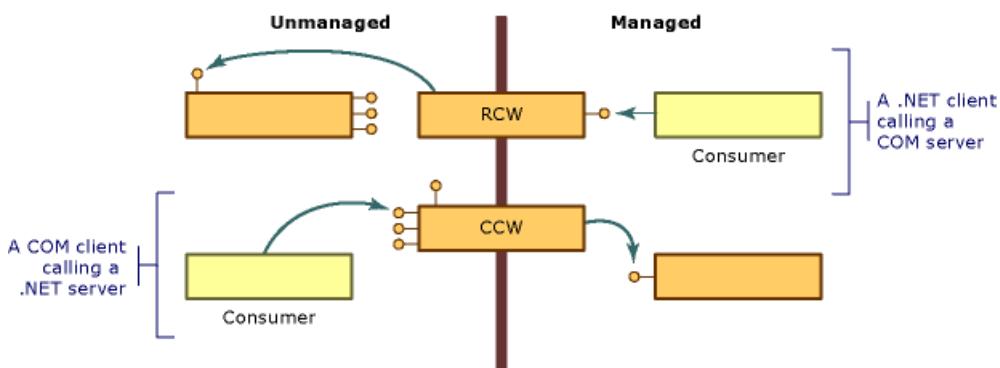
# COM Wrappers

9/20/2022 • 2 minutes to read • [Edit Online](#)

COM differs from the .NET runtime object model in several important ways:

- Clients of COM objects must manage the lifetime of those objects; the common language runtime manages the lifetime of objects in its environment.
- Clients of COM objects discover whether a service is available by requesting an interface that provides that service and getting back an interface pointer, or not. Clients of .NET objects can obtain a description of an object's functionality using reflection.
- .NET objects reside in memory managed by the .NET runtime execution environment. The execution environment can move objects around in memory for performance reasons and update all references to the objects it moves. Unmanaged clients, having obtained a pointer to an object, rely on the object to remain at the same location. These clients have no mechanism for dealing with an object whose location is not fixed.

To overcome these differences, the runtime provides wrapper classes to make both managed and unmanaged clients think they are calling objects within their respective environment. Whenever your managed client calls a method on a COM object, the runtime creates a [runtime callable wrapper](#) (RCW). RCWs abstract the differences between managed and unmanaged reference mechanisms, among other things. The runtime also creates a [COM callable wrapper](#) (CCW) to reverse the process, enabling a COM client to seamlessly call a method on a .NET object. As the following illustration shows, the perspective of the calling code determines which wrapper class the runtime creates.



In most cases, the standard RCW or CCW generated by the runtime provides adequate marshalling for calls that cross the boundary between COM and the .NET runtime. Using custom attributes, you can optionally adjust the way the runtime represents managed and unmanaged code.

## See also

- [Advanced COM Interoperability in .NET Framework](#)
- [Runtime Callable Wrapper](#)
- [COM Callable Wrapper](#)
- [Customizing Standard Wrappers in .NET Framework](#)
- [How to: Customize Runtime Callable Wrappers in .NET Framework](#)

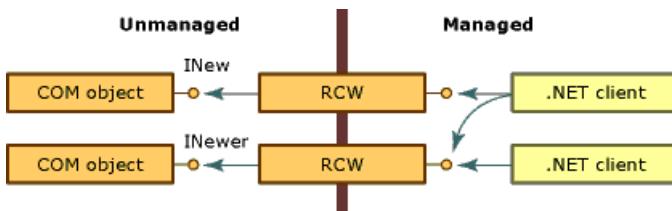
# Runtime Callable Wrapper

9/20/2022 • 3 minutes to read • [Edit Online](#)

The common language runtime exposes COM objects through a proxy called the runtime callable wrapper (RCW). Although the RCW appears to be an ordinary object to .NET clients, its primary function is to marshal calls between a .NET client and a COM object.

The runtime creates exactly one RCW for each COM object, regardless of the number of references that exist on that object. The runtime maintains a single RCW per process for each object. If you create an RCW in one application domain or apartment, and then pass a reference to another application domain or apartment, a proxy to the first object will be used. Note that this proxy is a new managed object and not the same as the initial RCW; this means the two managed objects are not equal but do represent the same COM object. As the following illustration shows, any number of managed clients can hold a reference to the COM objects that expose `INew` and `INewer` interfaces.

The following image shows the process for accessing COM objects through the runtime callable wrapper:



Using metadata derived from a type library, the runtime creates both the COM object being called and a wrapper for that object. Each RCW maintains a cache of interface pointers on the COM object it wraps and releases its reference on the COM object when the RCW is no longer needed. The runtime performs garbage collection on the RCW.

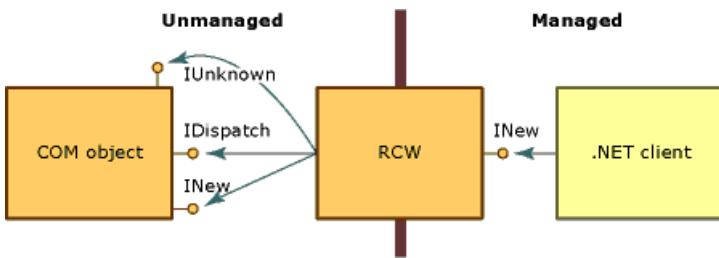
Among other activities, the RCW marshals data between managed and unmanaged code, on behalf of the wrapped object. Specifically, the RCW provides marshalling for method arguments and method return values whenever the client and server have different representations of the data passed between them.

The standard wrapper enforces built-in marshalling rules. For example, when a .NET client passes a `string` type as part of an argument to an unmanaged object, the wrapper converts the `string` to a `BSTR` type. Should the COM object return a `BSTR` to its managed caller, the caller receives a `string`. Both the client and the server send and receive data that is familiar to them. Other types require no conversion. For instance, a standard wrapper will always pass a 4-byte integer between managed and unmanaged code without converting the type.

## Marshalling selected interfaces

The primary goal of the [runtime callable wrapper](#) (RCW) is to hide the differences between the managed and unmanaged programming models. To create a seamless transition, the RCW consumes selected COM interfaces without exposing them to the .NET client, as shown in the following illustration.

The following image shows COM interfaces and the runtime callable wrapper:



When created as an early-bound object, the RCW is a specific type. It implements the interfaces that the COM object implements and exposes the methods, properties, and events from the object's interfaces. In the illustration, the RCW exposes the **INew** interface but consumes the **IUnknown** and **IDispatch** interfaces. Further, the RCW exposes all members of the **INew** interface to the .NET client.

The RCW consumes the interfaces listed in the following table, which are exposed by the object it wraps.

INTERFACE	DESCRIPTION
<b>IDispatch</b>	For late binding to COM objects through reflection.
<b>IErrorInfo</b>	Provides a textual description of the error, its source, a Help file, Help context, and the GUID of the interface that defined the error (always <b>GUID_NULL</b> for .NET classes).
<b>IProvideClassInfo</b>	If the COM object being wrapped implements <b>IProvideClassInfo</b> , the RCW extracts the type information from this interface to provide better type identity.
<b>IUnknown</b>	For object identity, type coercion, and lifetime management: <ul style="list-style-type: none"> <li>- Object identity The runtime distinguishes between COM objects by comparing the value of the <b>IUnknown</b> interface for each object.</li> <li>- Type coercion The RCW recognizes the dynamic type discovery performed by the <b>QueryInterface</b> method.</li> <li>- Lifetime management Using the <b>QueryInterface</b> method, the RCW gets and holds a reference to an unmanaged object until the runtime performs garbage collection on the wrapper, which releases the unmanaged object.</li> </ul>

The RCW optionally consumes the interfaces listed in the following table, which are exposed by the object it wraps.

INTERFACE	DESCRIPTION
<b>IConnectionPoint and IConnectionPointContainer</b>	The RCW converts objects that expose the connection-point event style to delegate-based events.
<b>IDispatchEx</b> (.NET Framework Only)	If the class implements <b>IDispatchEx</b> , the RCW implements <b>IExpando</b> . The <b>IDispatchEx</b> interface is an extension of the <b>IDispatch</b> interface that, unlike <b>IDispatch</b> , enables enumeration, addition, deletion, and case-sensitive calling of members.
<b>IEnumVARIANT</b>	Enables COM types that support enumerations to be treated as collections.

## See also

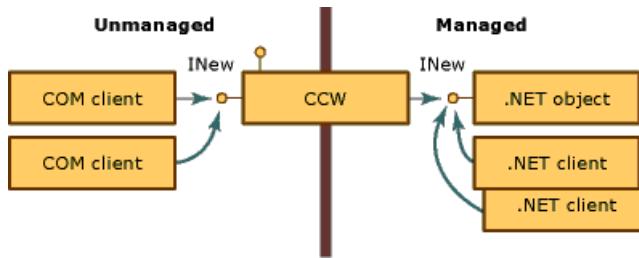
- [COM Wrappers](#)
- [COM Callable Wrapper](#)
- [Type Library to Assembly Conversion Summary](#)
- [Importing a Type Library as an Assembly](#)

# COM Callable Wrapper

9/20/2022 • 8 minutes to read • [Edit Online](#)

When a COM client calls a .NET object, the common language runtime creates the managed object and a COM callable wrapper (CCW) for the object. Unable to reference a .NET object directly, COM clients use the CCW as a proxy for the managed object.

The runtime creates exactly one CCW for a managed object, regardless of the number of COM clients requesting its services. As the following illustration shows, multiple COM clients can hold a reference to the CCW that exposes the `INew` interface. The CCW, in turn, holds a single reference to the managed object that implements the interface and is garbage collected. Both COM and .NET clients can make requests on the same managed object simultaneously.



COM callable wrappers are invisible to other classes running within the .NET runtime. Their primary purpose is to marshal calls between managed and unmanaged code; however, CCWs also manage the object identity and object lifetime of the managed objects they wrap.

## Object Identity

The runtime allocates memory for the .NET object from its garbage-collected heap, which enables the runtime to move the object around in memory as necessary. In contrast, the runtime allocates memory for the CCW from a noncollected heap, making it possible for COM clients to reference the wrapper directly.

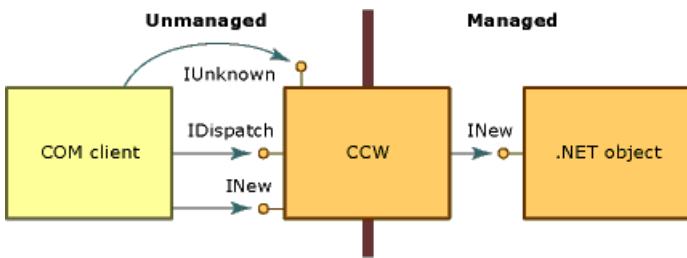
## Object Lifetime

Unlike the .NET client it wraps, the CCW is reference-counted in traditional COM fashion. When the reference count on the CCW reaches zero, the wrapper releases its reference on the managed object. A managed object with no remaining references is collected during the next garbage-collection cycle.

## Simulating COM interfaces

CCW exposes all public, COM-visible interfaces, data types, and return values to COM clients in a manner that is consistent with COM's enforcement of interface-based interaction. For a COM client, invoking methods on a .NET object is identical to invoking methods on a COM object.

To create this seamless approach, the CCW manufactures traditional COM interfaces, such as **IUnknown** and **IDispatch**. As the following illustration shows, the CCW maintains a single reference on the .NET object that it wraps. Both the COM client and the .NET object interact with each other through the proxy and stub construction of the CCW.



In addition to exposing the interfaces that are explicitly implemented by a class in the managed environment, the .NET runtime supplies implementations of the COM interfaces listed in the following table on behalf of the object. A .NET class can override the default behavior by providing its own implementation of these interfaces. However, the runtime always provides the implementation for the **IUnknown** and **IDispatch** interfaces.

INTERFACE	DESCRIPTION
<b>IDispatch</b>	Provides a mechanism for late binding to type.
<b>IErrorInfo</b>	Provides a textual description of the error, its source, a Help file, Help context, and the GUID of the interface that defined the error (always <b>GUID_NULL</b> for .NET classes).
<b>IProvideClassInfo</b>	Enables COM clients to gain access to the <b>ITypeInfo</b> interface implemented by a managed class. Returns <b>COR_E_NOTSUPPORTED</b> on .NET Core for types not imported from COM.
<b>ISupportErrorInfo</b>	Enables a COM client to determine whether the managed object supports the <b>IErrorInfo</b> interface. If so, enables the client to obtain a pointer to the latest exception object. All managed types support the <b>IErrorInfo</b> interface.
<b>ITypeInfo</b> (.NET Framework only)	Provides type information for a class that is exactly the same as the type information produced by <b>Tlbexp.exe</b> .
<b>IUnknown</b>	Provides the standard implementation of the <b>IUnknown</b> interface with which the COM client manages the lifetime of the CCW and provides type coercion.

A managed class can also provide the COM interfaces described in the following table.

INTERFACE	DESCRIPTION
The ( <i>_classname</i> ) class interface	Interface, exposed by the runtime and not explicitly defined, that exposes all public interfaces, methods, properties, and fields that are explicitly exposed on a managed object.
<b>IConnectionPoint</b> and <b>IConnectionPointContainer</b>	Interface for objects that source delegate-based events (an interface for registering event subscribers).
<b>IDispatchEx</b> (.NET Framework only)	Interface supplied by the runtime if the class implements <b>IExpando</b> . The <b>IDispatchEx</b> interface is an extension of the <b>IDispatch</b> interface that, unlike <b>IDispatch</b> , enables enumeration, addition, deletion, and case-sensitive calling of members.

INTERFACE	DESCRIPTION
IEnumVARIANT	Interface for collection-type classes, which enumerates the objects in the collection if the class implements <b>IEnumerable</b> .

## Introducing the class interface

The class interface, which is not explicitly defined in managed code, is an interface that exposes all public methods, properties, fields, and events that are explicitly exposed on the .NET object. This interface can be a dual or dispatch-only interface. The class interface receives the name of the .NET class itself, preceded by an underscore. For example, for class Mammal, the class interface is _Mammal.

For derived classes, the class interface also exposes all public methods, properties, and fields of the base class. The derived class also exposes a class interface for each base class. For example, if class Mammal extends class MammalSuperclass, which itself extends System.Object, the .NET object exposes to COM clients three class interfaces named _Mammal, _MammalSuperclass, and _Object.

For example, consider the following .NET class:

```
' Applies the ClassInterfaceAttribute to set the interface to dual.
<ClassInterface(ClassInterfaceType.AutoDual)> _
' Implicitly extends System.Object.
Public Class Mammal
    Sub Eat()
    Sub Breathe()
    Sub Sleep()
End Class
```

```
// Applies the ClassInterfaceAttribute to set the interface to dual.
[ClassInterface(ClassInterfaceType.AutoDual)]
// Implicitly extends System.Object.
public class Mammal
{
    public void Eat() {}
    public void Breathe() {}
    public void Sleep() {}
}
```

The COM client can obtain a pointer to a class interface named `_Mammal`. On .NET Framework, you can use the [Type Library Exporter \(Tlbexp.exe\)](#) tool to generate a type library containing the `_Mammal` interface definition. The Type Library Exporter is not supported on .NET Core. If the `Mammal` class implemented one or more interfaces, the interfaces would appear under the coclass.

```

[odl, uuid(...), hidden, dual, nonextensible, oleautomation]
interface _Mammal : IDispatch
{
    [id(0x00000000), propget] HRESULT ToString([out, retval] BSTR*
        pRetVal);
    [id(0x60020001)] HRESULT Equals([in] VARIANT obj, [out, retval]
        VARIANT_BOOL* pRetVal);
    [id(0x60020002)] HRESULT GetHashCode([out, retval] short* pRetVal);
    [id(0x60020003)] HRESULT GetType([out, retval] _Type** pRetVal);
    [id(0x6002000d)] HRESULT Eat();
    [id(0x6002000e)] HRESULT Breathe();
    [id(0x6002000f)] HRESULT Sleep();
}
[uuid(...)]
coclass Mammal
{
    [default] interface _Mammal;
}

```

Generating the class interface is optional. By default, COM interop generates a dispatch-only interface for each class you export to a type library. You can prevent or modify the automatic creation of this interface by applying the [ClassInterfaceAttribute](#) to your class. Although the class interface can ease the task of exposing managed classes to COM, its uses are limited.

#### Caution

Using the class interface, instead of explicitly defining your own, can complicate the future versioning of your managed class. Please read the following guidelines before using the class interface.

#### Define an explicit interface for COM clients to use rather than generating the class interface.

Because COM interop generates a class interface automatically, post-version changes to your class can alter the layout of the class interface exposed by the common language runtime. Since COM clients are typically unprepared to handle changes in the layout of an interface, they break if you change the member layout of the class.

This guideline reinforces the notion that interfaces exposed to COM clients must remain unchangeable. To reduce the risk of breaking COM clients by inadvertently reordering the interface layout, isolate all changes to the class from the interface layout by explicitly defining interfaces.

Use the [ClassInterfaceAttribute](#) to disengage the automatic generation of the class interface and implement an explicit interface for the class, as the following code fragment shows:

```

<ClassInterface(ClassInterfaceType.None)>Public Class LoanApp
    Implements IExplicit
    Sub M() Implements IExplicit.M
    ...
End Class

```

```

[ClassInterface(ClassInterfaceType.None)]
public class LoanApp : IExplicit
{
    int IExplicit.M() { return 0; }
}

```

The [ClassInterfaceType.None](#) value prevents the class interface from being generated when the class metadata is exported to a type library. In the preceding example, COM clients can access the `LoanApp` class only through the `IExplicit` interface.

#### Avoid caching dispatch identifiers (DispIds)

Using the class interface is an acceptable option for scripted clients, Microsoft Visual Basic 6.0 clients, or any late-bound client that does not cache the DispIds of interface members. DispIds identify interface members to enable late binding.

For the class interface, generation of DispIds is based on the position of the member in the interface. If you change the order of the member and export the class to a type library, you will alter the DispIds generated in the class interface.

To avoid breaking late-bound COM clients when using the class interface, apply the **ClassInterfaceAttribute** with the **ClassInterfaceType.AutoDispatch** value. This value implements a dispatch-only class interface, but omits the interface description from the type library. Without an interface description, clients are unable to cache DispIds at compile time. Although this is the default interface type for the class interface, you can apply the attribute value explicitly.

```
<ClassInterface(ClassInterfaceType.AutoDispatch)> Public Class LoanApp
    Implements IAnother
    Sub M() Implements IAnother.M
    ...
End Class
```

```
[ClassInterface(ClassInterfaceType.AutoDispatch)]
public class LoanApp
{
    public int M() { return 0; }
}
```

To get the DispId of an interface member at run time, COM clients can call **IDispatch.GetIDsOfNames**. To invoke a method on the interface, pass the returned DispId as an argument to **IDispatch.Invoke**.

#### Restrict using the dual interface option for the class interface.

Dual interfaces enable early and late binding to interface members by COM clients. At design time and during testing, you might find it useful to set the class interface to dual. For a managed class (and its base classes) that will never be modified, this option is also acceptable. In all other cases, avoid setting the class interface to dual.

An automatically generated dual interface might be appropriate in rare cases; however, more often it creates version-related complexity. For example, COM clients using the class interface of a derived class can easily break with changes to the base class. When a third party provides the base class, the layout of the class interface is out of your control. Further, unlike a dispatch-only interface, a dual interface (**ClassInterfaceType.AutoDual**) provides a description of the class interface in the exported type library. Such a description encourages late-bound clients to cache DispIds at compile time.

#### Ensure that all COM event notifications are late-bound.

By default, COM type information is embedded directly into managed assemblies, which eliminates the need for primary interop assemblies (PIAs). However, one of the limitations of embedded type information is that it does not support delivery of COM event notifications by early-bound vtable calls, but only supports late-bound **IDispatch::Invoke** calls.

If your application requires early-bound calls to COM event interface methods, you can set the **Embed Interop Types** property in Visual Studio to **true**, or include the following element in your project file:

```
<EmbedInteropTypes>True</EmbedInteropTypes>
```

## See also

- [ClassInterfaceAttribute](#)
- [COM Wrappers](#)
- [Exposing .NET Framework Components to COM](#)
- [Exposing .NET Core Components to COM](#)
- [Qualifying .NET Types for Interoperation](#)
- [Runtime Callable Wrapper](#)

# Tutorial: Use the ComWrappers API

9/20/2022 • 17 minutes to read • [Edit Online](#)

In this tutorial, you'll learn how to properly subclass the `ComWrappers` type to provide an optimized and AOT-friendly COM interop solution. Before starting this tutorial, you should be familiar with COM, its architecture, and existing COM interop solutions.

In this tutorial, you'll implement the following interface definitions. These interfaces and their implementations will demonstrate:

- Marshalling and unmarshalling types across the COM/.NET boundary.
- Two distinct approaches to consuming native COM objects in .NET.
- A recommended pattern for enabling custom COM interop in .NET 5 and beyond.

All source code used in this tutorial is available in the [dotnet/samples repository](#).

## C# definitions

```
interface IDemoGetType
{
    string? GetString();
}

interface IDemoStoreType
{
    void StoreString(int len, string? str);
}
```

## Win32 C++ definitions

```
IDL_INTERFACE("92BAA992-DB5A-4ADD-977B-B22838EE91FD")
IDemoGetType : public IUnknown
{
    HRESULT STDMETHODCALLTYPE GetString(_Outptr_ wchar_t** str) = 0;
};

IDL_INTERFACE("30619FEA-E995-41EA-8C8B-9A610D32ADCB")
IDemoStoreType : public IUnknown
{
    HRESULT STDMETHODCALLTYPE StoreString(int len, _In_z_ const wchar_t* str) = 0;
};
```

## Overview of the ComWrappers design

The `ComWrappers` API was designed to provide the minimal interaction needed to accomplish COM interop with the .NET 5+ runtime. This means that many of the niceties that exist with the built-in COM interop system are not present and must be built up from basic building blocks. The two primary responsibilities of the API are:

- Efficient object identification (for example, mapping between an `IUnknown*` instance and a managed object).
- [Garbage Collector \(GC\)](#) interaction.

These efficiencies are accomplished by requiring wrapper creation and acquisition to go through the `ComWrappers` API.

Since the `ComWrappers` API has so few responsibilities, it stands to reason that most of the interop work should be handled by the consumer – this is true. However, the additional work is largely mechanical and can be performed by a source-generation solution. As an example, the [C#/WinRT tool chain](#) is a source-generation solution that's built on top of `ComWrappers` to provide WinRT interop support.

## Implement a `ComWrappers` subclass

Providing a `ComWrappers` subclass means giving enough information to the .NET runtime to create and record wrappers for managed objects being projected into COM and COM objects being projected into .NET. Before we look at an outline of the subclass, we should define some terms.

**Managed Object Wrapper** – Managed .NET objects require wrappers to enable usage from a non-.NET environment. These wrappers are historically called COM Callable Wrappers (CCW).

**Native Object Wrapper** – COM objects that are implemented in a non-.NET language require wrappers to enable usage from .NET. These wrappers are historically called Runtime Callable Wrappers (RCW).

### Step 1 – Define methods to implement and understand their intent

To extend the `ComWrappers` type, you must implement the following three methods. Each of these methods represents the user's participation in the creation or deletion of a type of wrapper. The `ComputeVtables()` and `CreateObject()` methods create a Managed Object Wrapper and Native Object Wrapper, respectively. The `ReleaseObjects()` method is used by the runtime to make a request for the supplied collection of wrappers to be "released" from the underlying native object. In most cases, the body of the `ReleaseObjects()` method can simply throw `NotImplementedException`, since it's only called in an advanced scenario involving the [Reference Tracker framework](#).

```
// See referenced sample for implementation.
class DemoComWrappers : ComWrappers
{
    protected override unsafe ComInterfaceEntry* ComputeVtables(object obj, CreateComInterfaceFlags flags,
        out int count) =>
        throw new NotImplementedException();

    protected override object? CreateObject(IntPtr externalComObject, CreateObjectFlags flags) =>
        throw new NotImplementedException();

    protected override void ReleaseObjects(IEnumerable objects) =>
        throw new NotImplementedException();
}
```

To implement the `ComputeVtables()` method, decide which managed types you'd like to support. For this tutorial, we'll support the two previously defined interfaces (`IDemoGetType` and `IDemoStoreType`) and a managed type that implements the two interfaces (`DemoImpl`).

```
class DemoImpl : IDemoGetType, IDemoStoreType
{
    string? _string;
    public string? GetString() => _string;
    public void StoreString(int _, string? str) => _string = str;
}
```

For the `CreateObject()` method, you'll also need to determine what you'd like to support. In this case, though, we only know the COM interfaces we're interested in, not the COM classes. The interfaces being consumed from the COM side are the same as the ones we're projecting from the .NET side (that is, `IDemoGetType` and `IDemoStoreType`).

We won't implement `ReleaseObjects()` in this tutorial.

## Step 2 – Implement `ComputeVtables()`

Let's start with the Managed Object Wrapper – these wrappers are easier. You'll build a **Virtual Method Table**, or *vtable*, for each interface in order to project them into the COM environment. For this tutorial, you'll define a vtable as a sequence of pointers, where each pointer represents an implementation of a function on an interface – order is *very* important here. In COM, every interface inherits from `IUnknown`. The `IUnknown` type has three methods defined in the following order: `QueryInterface()`, `AddRef()`, and `Release()`. After the `IUnknown` methods come the specific interface methods. For example, consider `IDemoGetType` and `IDemoStoreType`. Conceptually, the vtables for the types would look like the following:

```
IDemoGetType      | IDemoStoreType  
=====|=====|  
QueryInterface   | QueryInterface  
AddRef          | AddRef  
Release          | Release  
GetString        | StoreString
```

Looking at `DemoImpl`, we already have an implementation for `GetString()` and `StoreString()`, but what about the `IUnknown` functions? How to **implement an `IUnknown` instance** is beyond the scope of this tutorial, but it can be done manually in `ComWrappers`. However, in this tutorial, you'll let the runtime handle that part. You can get the `IUnknown` implementation using the `ComWrappers.GetIUnknownImpl()` method.

It might seem like you've implemented all the methods, but unfortunately, only the `IUnknown` functions are consumable in a COM vtable. Since COM is outside of the runtime, you'll need to create native function pointers to your `DemoImpl` implementation. This can be done using C# function pointers and the `UnmanagedCallersOnlyAttribute`. You can create a function to insert into the vtable by creating a `static` function that mimics the COM function signature. Following is an example of the COM signature for `IDemoGetType.GetString()` – recall from the COM ABI that the first argument is the instance itself.

```
[UnmanagedCallersOnly]  
public static int GetString(IntPtr _this, IntPtr* str);
```

The wrapper implementation of `IDemoGetType.GetString()` should consist of marshalling logic and then a dispatch to the managed object being wrapped. All the state for dispatch is contained within the provided `_this` argument. The `_this` argument will actually be of type `ComInterfaceDispatch*`. This type represents a low-level structure with a single field, `Vtable`, that will be discussed later. Further details of this type and its layout are an implementation detail of the runtime and should not be depended upon. In order to retrieve the managed instance from a `ComInterfaceDispatch*` instance, use the following code:

```
IDemoGetType inst = ComInterfaceDispatch.GetInstance<IDemoGetType>((ComInterfaceDispatch*)_this);
```

Now that you have a C# method that can be inserted into a vtable, you can construct the vtable. Note the use of `RuntimeHelpers.AllocateTypeAssociatedMemory()` for allocating memory in a way that works with **unloadable** assemblies.

```

GetIUnknownImpl(
    out IntPtr fpQueryInterface,
    out IntPtr fpAddRef,
    out IntPtr fpRelease);

// Local variables with increment act as a guard against incorrect construction of
// the native vtable. It also enables a quick validation of final size.
int tableCount = 4;
int idx = 0;
var vtable = (IntPtr*)RuntimeHelpers.AllocateTypeAssociatedMemory(
    typeof(DemoComWrappers),
    IntPtr.Size * tableCount);
vtable[idx++] = fpQueryInterface;
vtable[idx++] = fpAddRef;
vtable[idx++] = fpRelease;
vtable[idx++] = (IntPtr)(delegate* unmanaged<IntPtr, IntPtr*, int>)&ABI.IDemoGetTypeManagedWrapper.GetString;
Debug.Assert(tableCount == idx);
s_IDemoGetTypeVTable = (IntPtr)vtable;

```

The allocation of vtables is the first part of implementing `ComputeVtables()`. You should also construct comprehensive COM definitions for types that you're planning to support – think `DemoImpl` and what parts of it should be usable from COM. Using the constructed vtables, you can now create a series of `ComInterfaceEntry` instances that represent the complete view of the managed object in COM.

```

s_DemoImplDefinitionLen = 2;
int idx = 0;
var entries = (ComInterfaceEntry*)RuntimeHelpers.AllocateTypeAssociatedMemory(
    typeof(DemoComWrappers),
    sizeof(ComInterfaceEntry) * s_DemoImplDefinitionLen);
entries[idx].IID = IDemoGetType.IID_IDemoGetType;
entries[idx++].Vtable = s_IDemoGetTypeVTable;
entries[idx].IID = IDemoStoreType.IID_IDemoStoreType;
entries[idx++].Vtable = s_IDemoStoreVTable;
Debug.Assert(s_DemoImplDefinitionLen == idx);
s_DemoImplDefinition = entries;

```

The allocation of vtables and entries for the Managed Object Wrapper can and should be done ahead of time since the data can be used for all instances of the type. The work here could be performed in a `static` constructor or a module initializer, but it should be done ahead of time so the `ComputeVtables()` method is as simple and quick as possible.

```

protected override unsafe ComInterfaceEntry* ComputeVtables(object obj, CreateComInterfaceFlags flags,
out int count)
{
    if (obj is DemoImpl)
    {
        count = s_DemoImplDefinitionLen;
        return s_DemoImplDefinition;
    }

    // Unknown type
    count = 0;
    return null;
}

```

Once you've implemented the `ComputeVtables()` method, the `ComWrappers` subclass will be able to produce Managed Object Wrappers for instances of `DemoImpl`. Be aware that the returned Managed Object Wrapper from the call to `GetOrCreateComInterfaceForObject()` is of type `IUnknown*`. If the native API that's being passed to

the wrapper requires a different interface, a `Marshal.QueryInterface()` for that interface must be performed.

```
var cw = new DemoComWrappers();
var demo = new DemoImpl();
IntPtr ccw = cw.GetOrCreateComInterfaceForObject(demo, CreateComInterfaceFlags.None);
```

### Step 3 – Implement `CreateObject()`

Constructing a Native Object Wrapper has more implementation options and a great deal more nuance than constructing a Managed Object Wrapper. The first question to address is how permissive the `ComWrappers` subclass will be in supporting COM types. To support all COM types, which is possible, you'll need to write a substantial amount of code or employ some clever uses of `Reflection.Emit`. For this tutorial, you'll only support COM instances that implement both `IDemoGetType` and `IDemoStoreType`. Since you know there is a finite set and have restricted that any supplied COM instance must implement both interfaces, you could provide a single, statically defined wrapper; however, dynamic cases are common enough in COM that we'll explore both options.

#### Static Native Object Wrapper

Let's look at the static implementation first. The static Native Object Wrapper involves defining a managed type that implements the .NET interfaces and can forward the calls on the managed type to the COM instance. A rough outline of the static wrapper follows.

```
// See referenced sample for implementation.
class DemoNativeStaticWrapper
    : IDemoGetType
    , IDemoStoreType
{
    public string? GetString() =>
        throw new NotImplementedException();

    public void StoreString(int len, string? str) =>
        throw new NotImplementedException();
}
```

To construct an instance of this class and provide it as a wrapper, you must define some policy. If this type is used as a wrapper, it would seem that since it implements both interfaces, the underlying COM instance should implement both interfaces too. Given that you're adopting this policy, you'll need to confirm this through calls to `Marshal.QueryInterface()` on the COM instance.

```
int hr = Marshal.QueryInterface(ptr, ref IDemoGetType.IID_IDemoGetType, out IntPtr IDemoGetTypeInst);
if (hr != 0)
{
    return null;
}

hr = Marshal.QueryInterface(ptr, ref IDemoStoreType.IID_IDemoStoreType, out IntPtr IDemoStoreTypeInst);
if (hr != 0)
{
    Marshal.Release(IDemoGetTypeInst);
    return null;
}

return new DemoNativeStaticWrapper()
{
    IDemoGetTypeInst = IDemoGetTypeInst,
    IDemoStoreTypeInst = IDemoStoreTypeInst
};
```

#### Dynamic Native Object Wrapper

Dynamic wrappers are more flexible because they provide a way for types to be queried at run time instead of

statically. In order to provide this support, you'll utilize `IDynamicInterfaceCastable` – further details can be found [here](#). Observe that `DemoNativeDynamicWrapper` only implements this interface. The functionality that the interface provides is a chance to determine what type is supported at run time. The source for this tutorial does a static check during creation but that is simply for code sharing since the check could be deferred until a call is made to `DemoNativeDynamicWrapper.IsInterfaceImplemented()`.

```
// See referenced sample for implementation.
internal class DemoNativeDynamicWrapper
    : IDynamicInterfaceCastable
{
    public RuntimeTypeHandle GetInterfaceImplementation(RuntimeTypeHandle interfaceType) =>
        throw new NotImplementedException();

    public bool IsInterfaceImplemented(RuntimeTypeHandle interfaceType, bool throwIfNotImplemented) =>
        throw new NotImplementedException();
}
```

Let's look at one of the interfaces that `DemoNativeDynamicWrapper` will dynamically support. The following code provides the implementation of `IDemoStoreType` using the *default interface methods* feature.

```
[DynamicInterfaceCastableImplementation]
unsafe interface IDemoStoreTypeNativeWrapper : IDemoStoreType
{
    public static void StoreString(IntPtr inst, int len, string? str);

    void IDemoStoreType.StoreString(int len, string? str)
    {
        var inst = ((DemoNativeDynamicWrapper)this).IDemoStoreTypeInst;
        StoreString(inst, len, str);
    }
}
```

There are two important things to note in this example:

1. The `DynamicInterfaceCastableImplementationAttribute` attribute. This attribute is required on any type that is returned from a `IDynamicInterfaceCastable` method. It has the added benefit of making IL trimming easier, which means AOT scenarios are more reliable.
2. The cast to `DemoNativeDynamicWrapper`. This is part of the dynamic nature of `IDynamicInterfaceCastable`. The type that's returned from `IDynamicInterfaceCastable.GetInterfaceImplementation()` is used to "blanket" the type that implements `IDynamicInterfaceCastable`. The gist here is the `this` pointer isn't what it pretends to be because we are permitting a cast from `DemoNativeDynamicWrapper` to `IDemoStoreTypeNativeWrapper`.

#### Forward calls to the COM instance

Regardless of which Native Object Wrapper is used, you need the ability to invoke functions on a COM instance. The implementation of `IDemoStoreTypeNativeWrapper.StoreString()` can serve as an example of employing `unmanaged` C# function pointers.

```

public static void StoreString(IntPtr inst, int len, string? str)
{
    IntPtr strLocal = Marshal.StringToCoTaskMemUni(str);
    int hr = ((delegate* unmanaged<IntPtr, int, IntPtr, int>)((*(void***)(inst + 3 /* IDemoStoreType.StoreString slot */))(inst, len, strLocal));
    if (hr != 0)
    {
        Marshal.FreeCoTaskMem(strLocal);
        Marshal.ThrowExceptionForHR(hr);
    }
}

```

Let's examine the dereferencing of the COM instance to access its vtable implementation. The COM ABI defines that the first pointer of an object is to the type's vtable and, from there, the desired slot can be accessed. Let's assume the address of the COM object is `0x10000`. The first pointer-sized value should be the address of the vtable – in this example `0x20000`. Once you're at the vtable, you look for the fourth slot (index 3 in zero-based indexing) to access the `StoreString()` implementation.

```

COM instance
0x10000 0x20000

VTable for IDemoStoreType
0x20000 <Address of QueryInterface>
0x20008 <Address of AddRef>
0x20010 <Address of Release>
0x20018 <Address of StoreString>

```

Having the function pointer then allows you to dispatch to that member function on that object by passing the object instance as the first parameter. This pattern should look familiar based on the function definitions of the Managed Object Wrapper implementation.

Once the `CreateObject()` method is implemented, the `ComWrappers` subclass will be able to produce Native Object Wrappers for COM instances that implement both `IDemoGetType` and `IDemoStoreType`.

```

IntPtr iunk = ...; // Get a COM instance from native code.
object rcw = cw.GetOrCreateObjectForComInstance(iunk, CreateObjectFlags.UniqueInstance);

```

#### Step 4 – Handle Native Object Wrapper lifetime details

The `ComputeVtables()` and `CreateObject()` implementations covered some wrapper lifetime details, but there are further considerations. While this can be a short step, it can also significantly increase the complexity of the `ComWrappers` design.

Unlike the Managed Object Wrapper, which is controlled by calls to its `AddRef()` and `Release()` methods, the lifetime of a Native Object Wrapper is nondeterministically handled by the GC. The question here is, when does the Native Object Wrapper call `Release()` on the `IntPtr` that represents the COM instance? There are two general buckets:

1. The Native Object Wrapper's Finalizer is responsible for calling the COM instance's `Release()` method.  
This is the only time when it's safe to call this method. At this point, it's been correctly determined by the GC that there are no other references to the Native Object Wrapper in the .NET runtime. There can be complexity here if you're properly supporting COM Apartments; for more information, see the [Additional considerations](#) section.
2. The Native Object Wrapper implements `IDisposable` and calls `Release()` in `Dispose()`.

## NOTE

The `IDisposable` pattern should only be supported if, during the `CreateObject()` call, the `CreateObjectFlags.UniqueInstance` flag was passed in. If this requirement is not followed, it's possible for disposed Native Object Wrappers to be reused after being disposed.

## Using the `ComWrappers` subclass

You now have a `ComWrappers` subclass that can be tested. To avoid creating a native library that returns a COM instance that implements `IDemoGetType` and `IDemoStoreType`, you'll use the Managed Object Wrapper and treat it as a COM instance – this must be possible in order to pass it COM anyways.

Let's create a Managed Object Wrapper first. Instantiate a `DemoImpl` instance and display its current string state.

```
var demo = new DemoImpl();

string? value = demo.GetString();
Console.WriteLine($"Initial string: {value ?? "<null>"}");
```

Now you can create an instance of `DemoComWrappers` and a Managed Object Wrapper that you can then pass into a COM environment.

```
var cw = new DemoComWrappers();

IntPtr ccw = cw.GetOrCreateComInterfaceForObject(demo, CreateComInterfaceFlags.None);
```

Instead of passing the Managed Object Wrapper to a COM environment, pretend you just received this COM instance, so you'll create a Native Object Wrapper for it instead.

```
var rcw = cw.GetOrCreateObjectForComInstance(ccw, CreateObjectFlags.UniqueInstance);
```

With the Native Object Wrapper, you should be able to cast it to one of the desired interfaces and use it as a normal managed object. You can examine the `DemoImpl` instance and observe the impact of operations on the Native Object Wrapper that's wrapping a Managed Object Wrapper that's in turn wrapping the managed instance.

```
var getter = (IDemoGetType)rcw;
var store = (IDemoStoreType)rcw;

string msg = "hello world!";
store.StoreString(msg.Length, msg);
Console.WriteLine($"Setting string through wrapper: {msg}");

value = demo.GetString();
Console.WriteLine($"Get string through managed object: {value}");

msg = msg.ToUpper();
demo.StoreString(msg.Length, msg.ToUpper());
Console.WriteLine($"Setting string through managed object: {msg}");

value = getter.GetString();
Console.WriteLine($"Get string through wrapper: {value}");
```

Since your `ComWrapper` subclass was designed to support `CreateObjectFlags.UniqueInstance`, you can clean up

the Native Object Wrapper immediately instead of waiting for a GC to occur.

```
(rcw as IDisposable)?.Dispose();
```

## COM Activation with `ComWrappers`

The creation of COM objects is typically performed via COM Activation – a complex scenario outside the scope of this document. In order to provide a conceptual pattern to follow, we introduce the `CoCreateInstance()` API, used for COM Activation, and illustrate how it can be used with `ComWrappers`.

Assume you have the following C# code in your application. The below example uses `CoCreateInstance()` to activate a COM class and the built-in COM interop system to marshal the COM instance to the appropriate interface. Note the use of `typeof(I).GUID` is limited to an assert and is a case of using reflection which can impact if the code is AOT-friendly.

```
public static I ActivateClass<I>(Guid clsid, Guid iid)
{
    Debug.Assert(iid == typeof(I).GUID);
    int hr = CoCreateInstance(ref clsid, IntPtr.Zero, /*CLCTX_INPROC_SERVER*/ 1, ref iid, out object obj);
    if (hr < 0)
    {
        Marshal.ThrowExceptionForHR(hr);
    }
    return (I)obj;
}

[DllImport("Ole32")]
private static extern int CoCreateInstance(
    ref Guid rclsid,
    IntPtr pUnkOuter,
    int dwClsContext,
    ref Guid riid,
    [MarshalAs(UnmanagedType.Interface)] out object ppObj);
```

Converting the above to use `ComWrappers` involves removing the `[MarshalAs(UnmanagedType.Interface)]` from the `CoCreateInstance()` P/Invoke and performing the marshalling manually.

```
static ComWrappers s_ComWrappers = ...;

public static I ActivateClass<I>(Guid clsid, Guid iid)
{
    Debug.Assert(iid == typeof(I).GUID);
    int hr = CoCreateInstance(ref clsid, IntPtr.Zero, /*CLCTX_INPROC_SERVER*/ 1, ref iid, out IntPtr obj);
    if (hr < 0)
    {
        Marshal.ThrowExceptionForHR(hr);
    }
    return (I)s_ComWrappers.GetOrCreateObjectForComInstance(obj, CreateObjectFlags.None);
}

[DllImport("Ole32")]
private static extern int CoCreateInstance(
    ref Guid rclsid,
    IntPtr pUnkOuter,
    int dwClsContext,
    ref Guid riid,
    out IntPtr ppObj);
```

It is also possible to abstract away factory-style functions like `ActivateClass<I>` by including the activation logic

in the class constructor for a Native Object Wrapper. The constructor can use the [ComWrappers.GetOrRegisterObjectForComInstance\(\)](#) API to associate the newly constructed managed object with the activated COM instance.

## Additional considerations

**Native AOT** – Ahead-of-time (AOT) compilation provides improved startup cost as JIT compilation is avoided. Removing the need for JIT compilation is also often required on some platforms. Supporting AOT was a goal of the [ComWrappers](#) API, but any wrapper implementation must be careful not to inadvertently introduce cases where AOT breaks down, such as using reflection. The [Type.GUID](#) property is an example of where reflection is used, but in a non-obvious way. The [Type.GUID](#) property uses reflection to inspect the type's attributes and then potentially the type's name and containing assembly in order to generate its value.

**Source generation** – Most of the code that's needed for COM interop and a [ComWrappers](#) implementation can likely be autogenerated by some tooling. Source for both types of wrappers could be generated given the proper COM definitions – for example, Type Library (TLB), IDL, or a Primary Interop Assembly (PIA).

**Global registration** – Since the [ComWrappers](#) API was designed as a new phase of COM interop, it needed to have some way to partially integrate with the existing system. There are globally impacting static methods on the [ComWrappers](#) API that permit registration of a global instance for various support. These methods are designed for [ComWrappers](#) instances that are expecting to provide comprehensive COM interop support in all cases – akin to the built-in COM interop system.

**Reference Tracker support** – This support is primarily used for WinRT scenarios and represents an advanced scenario. For most [ComWrapper](#) implementations, either a [CreateComInterfaceFlags.TrackerSupport](#) or [CreateObjectFlags.TrackerObject](#) flag should throw a [NotSupportedException](#). If you'd like to enable this support, perhaps on a Windows or even non-Windows platform, it is highly recommended to reference the [C#/WinRT tool chain](#).

Aside from the lifetime, type system, and functional features that are discussed previously, a COM-compliant implementation of [ComWrappers](#) requires additional considerations. For any implementation that will be used on the Windows platform, there are the following considerations:

- **Apartments** – COM's organizational structure for threading is called "Apartments" and has strict rules that must be followed for stable operations. This tutorial does not implement apartment-aware Native Object Wrappers, but any production-ready implementation should be apartment-aware. To accomplish this, we recommend using the [RoGetAgileReference](#) API introduced in Windows 8. For versions prior to Windows 8, consider the [Global Interface Table](#).
- **Security** – COM provides a rich security model for class activation and proxied permission.

# Qualifying .NET Types for COM Interoperation

9/20/2022 • 2 minutes to read • [Edit Online](#)

If you intend to expose types in an assembly to COM applications, consider the requirements of COM interop at design time. Managed types (class, interface, structure, and enumeration) seamlessly integrate with COM types when you adhere to the following guidelines:

- Classes should implement interfaces explicitly.

Although COM interop provides a mechanism to automatically generate an interface containing all members of the class and the members of its base class, it is far better to provide explicit interfaces. The automatically generated interface is called the class interface. For guidelines, see [Introducing the class interface](#).

You can use Visual Basic, C#, and C++ to incorporate interface definitions in your code, instead of having to use Interface Definition Language (IDL) or its equivalent. For syntax details, see your language documentation.

- Managed types must be public.

Only public types in an assembly are registered and exported to the type library. As a result, only public types are visible to COM.

Managed types expose features to other managed code that might not be exposed to COM. For instance, parameterized constructors, static methods, and constant fields are not exposed to COM clients. Further, as the runtime marshals data in and out of a type, the data might be copied or transformed.

- Methods, properties, fields, and events must be public.

Members of public types must also be public if they are to be visible to COM. You can restrict the visibility of an assembly, a public type, or public members of a public type by applying the [ComVisibleAttribute](#). By default, all public types and members are visible.

- Types must have a public parameterless constructor to be activated from COM.

Managed, public types are visible to COM. However, without a public parameterless constructor (a constructor with no arguments), COM clients cannot create the type. COM clients can still use the type if it is activated by some other means.

- Types cannot be abstract.

Neither COM clients nor .NET clients can create abstract types.

When exported to COM, the inheritance hierarchy of a managed type is flattened. Versioning also differs between managed and unmanaged environments. Types exposed to COM do not have the same versioning characteristics as other managed types.

## See also

- [ComVisibleAttribute](#)
- [Exposing .NET Framework Components to COM](#)
- [Introducing the class interface](#)
- [Applying Interop Attributes](#)
- [Packaging a .NET Framework Assembly for COM](#)

# Applying Interop Attributes

9/20/2022 • 4 minutes to read • [Edit Online](#)

The [System.Runtime.InteropServices](#) namespace provides three categories of interop-specific attributes: those applied by you at design time, those applied by COM interop tools and APIs during the conversion process, and those applied either by you or COM interop.

If you are unfamiliar with the task of applying attributes to managed code, see [Extending Metadata Using Attributes](#). Like other custom attributes, you can apply interop-specific attributes to types, methods, properties, parameters, fields, and other members.

## Design-Time Attributes

You can adjust the outcome of the conversion process performed by COM interop tools and APIs by using design-time attributes. The following table describes the attributes that you can apply to your managed source code. COM interop tools, on occasion, might also apply the attributes described in this table.

ATTRIBUTE	DESCRIPTION
<a href="#">AutomationProxyAttribute</a>	Specifies whether the type should be marshalled using the Automation marshaller or a custom proxy and stub.
<a href="#">ClassInterfaceAttribute</a>	Controls the type of interface generated for a class.
<a href="#">CoClassAttribute</a>	Identifies the CLSID of the original coclass imported from a type library.  COM interop tools typically apply this attribute.
<a href="#">ComImportAttribute</a>	Indicates that a coclass or interface definition was imported from a COM type library. The runtime uses this flag to know how to activate and marshal the type. This attribute prohibits the type from being exported back to a type library.  COM interop tools typically apply this attribute.
<a href="#">ComRegisterFunctionAttribute</a>	Indicates that a method should be called when the assembly is registered for use from COM, so that user-written code can be executed during the registration process.
<a href="#">ComSourceInterfacesAttribute</a>	Identifies interfaces that are sources of events for the class.  COM interop tools can apply this attribute.
<a href="#">ComUnregisterFunctionAttribute</a>	Indicates that a method should be called when the assembly is unregistered from COM, so that user-written code can execute during the process.

ATTRIBUTE	DESCRIPTION
<a href="#">ComVisibleAttribute</a>	Renders types invisible to COM when the attribute value equals <b>false</b> . This attribute can be applied to an individual type or to an entire assembly to control COM visibility. By default, all managed, public types are visible; the attribute is not needed to make them visible.
<a href="#">DispIdAttribute</a>	Specifies the COM dispatch identifier (DISPID) of a method or field. This attribute contains the DISPID for the method, field, or property it describes.  COM interop tools can apply this attribute.
<a href="#">ComDefaultInterfaceAttribute</a>	Indicates the default interface for a COM class implemented in .NET.  COM interop tools can apply this attribute.
<a href="#">FieldOffsetAttribute</a>	Indicates the physical position of each field within a class when used with the <a href="#">StructLayoutAttribute</a> , and the <a href="#">LayoutKind</a> is set to Explicit.
<a href="#">GuidAttribute</a>	Specifies the globally unique identifier (GUID) of a class, interface, or an entire type library. The string passed to the attribute must be a format that is an acceptable constructor argument for the type <a href="#">System.Guid</a> .  COM interop tools can apply this attribute.
<a href="#">IDispatchImplAttribute</a>	Indicates which <a href="#">IDispatch</a> interface implementation the common language runtime uses when exposing dual interfaces and dispinterfaces to COM.
<a href="#">InAttribute</a>	Indicates that data should be marshalled in to the caller. Can be used to attribute parameters.
<a href="#">InterfaceTypeAttribute</a>	Controls how a managed interface is exposed to COM clients (Dual, IUnknown-derived, or IDispatch only).  COM interop tools can apply this attribute.
<a href="#">LCIDConversionAttribute</a>	Indicates that an unmanaged method signature expects an LCID parameter.  COM interop tools can apply this attribute.
<a href="#">MarshalAsAttribute</a>	Indicates how the data in fields or parameters should be marshalled between managed and unmanaged code. The attribute is always optional because each data type has default marshalling behavior.  COM interop tools can apply this attribute.
<a href="#">OptionalAttribute</a>	Indicates that a parameter is optional.  COM interop tools can apply this attribute.

ATTRIBUTE	DESCRIPTION
<a href="#">OutAttribute</a>	Indicates that the data in a field or parameter must be marshalled from a called object back to its caller.
<a href="#">PreserveSigAttribute</a>	Suppresses the HRESULT or retval signature transformation that normally takes place during interoperation calls. The attribute affects marshalling as well as type library exporting.  COM interop tools can apply this attribute.
<a href="#">ProgIdAttribute</a>	Specifies the ProgID of a .NET class. Can be used to attribute classes.
<a href="#">StructLayoutAttribute</a>	Controls the physical layout of the fields of a class.  COM interop tools can apply this attribute.

## Conversion-Tool Attributes

The following table describes attributes that COM interop tools apply during the conversion process. You do not apply these attributes at design time.

ATTRIBUTE	DESCRIPTION
<a href="#">ComAliasNameAttribute</a>	Indicates the COM alias for a parameter or field type. Can be used to attribute parameters, fields, or return values.
<a href="#">ComConversionLossAttribute</a>	Indicates that information about a class or interface was lost when it was imported from a type library to an assembly.
<a href="#">ComEventInterfaceAttribute</a>	Identifies the source interface and the class that implements the methods of the event interface.
<a href="#">ImportedFromTypeLibAttribute</a>	Indicates that the assembly was originally imported from a COM type library. This attribute contains the type library definition of the original type library.
<a href="#">TypeLibFuncAttribute</a>	Contains the FUNCFLAGS that were originally imported for this function from the COM type library.
<a href="#">TypeLibTypeAttribute</a>	Contains the TYPEFLAGS that were originally imported for this type from the COM type library.
<a href="#">TypeLibVarAttribute</a>	Contains the VARFLAGS that were originally imported for this variable from the COM type library.

## See also

- [System.Runtime.InteropServices](#)
- [Exposing .NET Framework Components to COM](#)
- [Attributes](#)
- [Qualifying .NET Types for Interoperation](#)
- [Packaging a .NET Framework Assembly for COM](#)

# Working with Interop Exceptions in Unmanaged Code

9/20/2022 • 2 minutes to read • [Edit Online](#)

Unmanaged code exception interop is supported on Windows platforms only. Portability issues arise on non-Windows platforms. Since the Unix ABI has no definition for exception handling, managed code can't know how exception mechanisms work under the covers. Therefore, exceptions can end up resulting in unpredictable behaviors and crashes.

## Setjmp/Longjmp Behaviors

Interop with `setjmp` and `longjmp` C functions is not supported. You can't use `longjmp` to skip over managed frames.

For more information, see [longjmp documentation](#).

## See also

- [Exceptions](#)
- [Interop with Native Libraries](#)

# .NET distribution packaging

9/20/2022 • 6 minutes to read • [Edit Online](#)

As .NET 5 (and .NET Core) and later versions become available on more and more platforms, it's useful to learn how to package, name, and version apps and libraries that use it. This way, package maintainers can help ensure a consistent experience no matter where users choose to run .NET. This article is useful for users that are:

- Attempting to build .NET from source.
- Wanting to make changes to the .NET CLI that could impact the resulting layout or packages produced.

## Disk layout

When installed, .NET consists of several components that are laid out as follows in the file system:

```

{dotnet_root}                                     (*)
  └── dotnet                               (1)
  └── LICENSE.txt                         (8)
  └── ThirdPartyNotices.txt            (8)
    └── host                                (*)
      └── fxr                                (*)
        └── <fxr version>          (2)
    └── sdk                                 (*)
      └── <sdk version>          (3)
    └── sdk-manifests                     (4)      (*)
      └── <sdk feature band version>
    └── library-packs                   (4)      (*)
    └── metadata                           (4)      (*)
      └── workloads
        └── <sdk feature band version>
    └── template-packs                 (4)      (*)
  └── packs                                (*)
    └── Microsoft.AspNetCore.App.Ref      (*)
      └── <aspnetcore ref version>    (11)
    └── Microsoft.NETCore.App.Ref       (*)
      └── <netcore ref version>        (12)
    └── Microsoft.NETCore.App.Host.<rid>  (*)
      └── <apphost version>           (13)
    └── Microsoft.WindowsDesktop.App.Ref  (*)
      └── <desktop ref version>        (14)
    └── NETStandard.Library.Ref         (*)
      └── <netstandard version>        (15)
  └── shared                                (*)
    └── Microsoft.NETCore.App           (*)
      └── <runtime version>           (5)
    └── Microsoft.AspNetCore.App       (*)
      └── <aspnetcore version>        (6)
    └── Microsoft.AspNetCore.All       (*)
      └── <aspnetcore version>        (6)
    └── Microsoft.WindowsDesktop.App   (*)
      └── <desktop app version>       (7)
  └── templates                             (*)
    └── <templates version>           (17)
/
└── etc/dotnet
  └── install_location          (16)
└── usr/share/man/man1
  └── dotnet.1.gz                (9)
└── usr/bin
  └── dotnet                      (10)

```

- (1) **dotnet** The host (also known as the "muxer") has two distinct roles: activate a runtime to launch an application, and activate an SDK to dispatch commands to it. The host is a native executable (`dotnet.exe`).

While there's a single host, most of the other components are in versioned directories (2,3,5,6). This means multiple versions can be present on the system since they're installed side by side.

- (2) **host/fxr/<fxr version>** contains the framework resolution logic used by the host. The host uses the latest hostfxr that is installed. The hostfxr is responsible for selecting the appropriate runtime when executing a .NET application. For example, an application built for .NET Core 2.0.0 uses the 2.0.5 runtime when it's available. Similarly, hostfxr selects the appropriate SDK during development.
- (3) **sdk/<sdk version>** The SDK (also known as "the tooling") is a set of managed tools that are used to write and build .NET libraries and applications. The SDK includes the .NET CLI, the managed languages compilers, MSBuild, and associated build tasks and targets, NuGet, new project templates, and so on.
- (4) **sdk-manifests/<sdk feature band version>** The names and versions of the assets that an optional workload installation requires are maintained in workload manifests stored in this folder. The

folder name is the feature band version of the SDK. So for an SDK version such as 6.0.102, this folder would still be named 6.0.100. When a workload is installed, the following folders are created as needed for the workload's assets: *library-packs*, *metadata*, and *template-packs*. A distribution can create an empty */metadata/workloads/< sdkfeatureband >/userlocal* file if workloads should be installed under a user path rather than in the *dotnet* folder. For more information, see GitHub issue [dotnet/installer#12104](#).

The **shared** folder contains frameworks. A shared framework provides a set of libraries at a central location so they can be used by different applications.

- (5) **shared/Microsoft.NETCore.App/<runtime version>** This framework contains the .NET runtime and supporting managed libraries.
- (6) **shared/Microsoft.AspNetCore.{App,All}/<aspnetcore version>** contains the ASP.NET Core libraries. The libraries under `Microsoft.AspNetCore.App` are developed and supported as part of the .NET project. The libraries under `Microsoft.AspNetCore.All` are a superset that also contains third-party libraries.
- (7) **shared/Microsoft.Desktop.App/<desktop app version>** contains the Windows desktop libraries. This isn't included on non-Windows platforms.
- (8) **LICENSE.txt,ThirdPartyNotices.txt** are the .NET license and licenses of third-party libraries used in .NET, respectively.
- (9,10) **dotnet.1.gz, dotnet** `dotnet.1.gz` is the dotnet manual page. `dotnet` is a symlink to the dotnet host(1). These files are installed at well-known locations for system integration.
- (11,12) **Microsoft.NETCore.App.Ref,Microsoft.AspNetCore.App.Ref** describe the API of an `x.y` version of .NET and ASP.NET Core respectively. These packs are used when compiling for those target versions.
- (13) **Microsoft.NETCore.App.Host.<rid>** contains a native binary for platform `rid`. This binary is a template when compiling a .NET application into a native binary for that platform.
- (14) **Microsoft.WindowsDesktop.App.Ref** describes the API of `x.y` version of Windows Desktop applications. These files are used when compiling for that target. This isn't provided on non-Windows platforms.
- (15) **NETStandard.Library.Ref** describes the netstandard `x.y` API. These files are used when compiling for that target.
- (16) **/etc/dotnet/install_location** is a file that contains the full path for `{dotnet_root}`. The path may end with a newline. It's not necessary to add this file when the root is `/usr/share/dotnet`.
- (17) **templates** contains the templates used by the SDK. For example, `dotnet new` finds project templates here.

The folders marked with `(*)` are used by multiple packages. Some package formats (for example, `rpm`) require special handling of such folders. The package maintainer must take care of this.

## Recommended packages

.NET versioning is based on the runtime component `[major].[minor]` version numbers. The SDK version uses the same `[major].[minor]` and has an independent `[patch]` that combines feature and patch semantics for the SDK. For example: SDK version 2.2.302 is the second patch release of the third feature release of the SDK that supports the 2.2 runtime. For more information about how versioning works, see [.NET versioning overview](#).

Some of the packages include part of the version number in their name. This allows you to install a specific version. The rest of the version isn't included in the version name. This allows the OS package manager to

update the packages (for example, automatically installing security fixes). Supported package managers are Linux specific.

The following lists the recommended packages:

- `dotnet-sdk-[major].[minor]` - Installs the latest sdk for specific runtime
  - **Version:** <sdk version>
  - **Example:** dotnet-sdk-2.1
  - **Contains:** (3),(4)
  - **Dependencies:** `dotnet-runtime-[major].[minor]`, `aspnetcore-runtime-[major].[minor]`,  
`dotnet-targeting-pack-[major].[minor]`, `aspnetcore-targeting-pack-[major].[minor]`,  
`netstandard-targeting-pack-[netstandard_major].[netstandard_minor]`,  
`dotnet-apphost-pack-[major].[minor]`, `dotnet-templates-[major].[minor]`
- `aspnetcore-runtime-[major].[minor]` - Installs a specific ASP.NET Core runtime
  - **Version:** <aspnetcore runtime version>
  - **Example:** aspnetcore-runtime-2.1
  - **Contains:** (6)
  - **Dependencies:** `dotnet-runtime-[major].[minor]`
- `dotnet-runtime-deps-[major].[minor]` (*Optional*) - Installs the dependencies for running self-contained applications
  - **Version:** <runtime version>
  - **Example:** dotnet-runtime-deps-2.1
  - **Dependencies:** *distribution-specific dependencies*
- `dotnet-runtime-[major].[minor]` - Installs a specific runtime
  - **Version:** <runtime version>
  - **Example:** dotnet-runtime-2.1
  - **Contains:** (5)
  - **Dependencies:** `dotnet-hostfxr-[major].[minor]`, `dotnet-runtime-deps-[major].[minor]`
- `dotnet-hostfxr-[major].[minor]` - dependency
  - **Version:** <runtime version>
  - **Example:** dotnet-hostfxr-3.0
  - **Contains:** (2)
  - **Dependencies:** `dotnet-host`
- `dotnet-host` - dependency
  - **Version:** <runtime version>
  - **Example:** dotnet-host
  - **Contains:** (1),(8),(9),(10),(16)
- `dotnet-apphost-pack-[major].[minor]` - dependency
  - **Version:** <runtime version>
  - **Contains:** (13)
- `dotnet-targeting-pack-[major].[minor]` - Allows targeting a non-latest runtime
  - **Version:** <runtime version>
  - **Contains:** (12)
- `aspnetcore-targeting-pack-[major].[minor]` - Allows targeting a non-latest runtime

- **Version:** <aspnetcore runtime version>
- **Contains:** (11)
- `netstandard-targeting-pack-[netstandard_major].[netstandard_minor]` - Allows targeting a netstandard version
  - **Version:** <sdk version>
  - **Contains:** (15)
- `dotnet-templates-[major].[minor]`
  - **Version:** <sdk version>
  - **Contains:** (15)

The `dotnet-runtime-deps-[major].[minor]` requires understanding the *distro-specific dependencies*. Because the distro build system may be able to derive this automatically, the package is optional, in which case these dependencies are added directly to the `dotnet-runtime-[major].[minor]` package.

When package content is under a versioned folder, the package name `[major].[minor]` match the versioned folder name. For all packages, except the `netstandard-targeting-pack-[netstandard_major].[netstandard_minor]`, this also matches with the .NET version.

Dependencies between packages should use an *equal or greater than* version requirement. For example, `dotnet-sdk-2.2:2.2.401` requires `aspnetcore-runtime-2.2 >= 2.2.6`. This makes it possible for the user to upgrade their installation via a root package (for example, `dnf update dotnet-sdk-2.2`).

Most distributions require all artifacts to be built from source. This has some impact on the packages:

- The third-party libraries under `shared/Microsoft.AspNetCore.All` can't be easily built from source. So that folder is omitted from the `aspnetcore-runtime` package.
- The `NuGetFallbackFolder` is populated using binary artifacts from `nuget.org`. It should remain empty.

Multiple `dotnet-sdk` packages may provide the same files for the `NuGetFallbackFolder`. To avoid issues with the package manager, these files should be identical (checksum, modification date, and so on).

## Building packages

The [dotnet/source-build](#) repository provides instructions on how to build a source tarball of the .NET SDK and all its components. The output of the source-build repository matches the layout described in the first section of this article.

# Library guidance

9/20/2022 • 2 minutes to read • [Edit Online](#)

This guidance provides recommendations for developers to create high-quality .NET libraries. This documentation focuses on the *what* and the *why* when building a .NET library, not the *how*.

Aspects of high-quality .NET libraries:

- **Inclusive** - Good .NET libraries strive to support many platforms, programming languages, and applications.
- **Stable** - Good .NET libraries coexist in the .NET ecosystem, running in applications built with many libraries.
- **Designed to evolve** - .NET libraries should improve and evolve over time, while supporting existing users.
- **Debuggable** - .NET libraries should use the latest tools to create a great debugging experience for users.
- **Trusted** - .NET libraries have developers' trust by publishing to NuGet using security best practices.

[Get Started](#)

## Types of recommendations

Each article presents four types of recommendations: **Do**, **Consider**, **Avoid**, and **Do not**. The type of recommendation indicates how strongly it should be followed.

You should almost always follow a **Do** recommendation. For example:

- ✓ **DO** distribute your library using a NuGet package.

On the other hand, **Consider** recommendations should generally be followed, but there are legitimate exceptions to the rule and you shouldn't feel bad about not following the guidance:

- ✓ **CONSIDER** using [SemVer 2.0.0](#) to version your NuGet package.

**Avoid** recommendations mention things that are generally not a good idea, but breaking the rule sometimes makes sense:

- ✗ **AVOID** NuGet package references that demand an exact version.

And finally, **Do not** recommendations indicate something you should almost never do:

- ✗ **DO NOT** publish strong-named and non-strong-named versions of your library. For example, `Contoso.Api` and `Contoso.Api.StrongNamed`.

[NEXT](#)

# Framework Design Guidelines

9/20/2022 • 2 minutes to read

This section provides guidelines for designing libraries that extend and interact with the .NET Framework. The goal is to help library designers ensure API consistency and ease of use by providing a unified programming model that is independent of the programming language used for development. We recommend that you follow these design guidelines when developing classes and components that extend the .NET Framework. Inconsistent library design adversely affects developer productivity and discourages adoption.

The guidelines are organized as simple recommendations prefixed with the terms **Do**, **Consider**, **Avoid**, and **Do not**. These guidelines are intended to help class library designers understand the trade-offs between different solutions. There might be situations where good library design requires that you violate these design guidelines. Such cases should be rare, and it is important that you have a clear and compelling reason for your decision.

These guidelines are excerpted from the book *Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries, 2nd Edition*, by Krzysztof Cwalina and Brad Abrams.

## In This Section

### [Naming Guidelines](#)

Provides guidelines for naming assemblies, namespaces, types, and members in class libraries.

### [Type Design Guidelines](#)

Provides guidelines for using static and abstract classes, interfaces, enumerations, structures, and other types.

### [Member Design Guidelines](#)

Provides guidelines for designing and using properties, methods, constructors, fields, events, operators, and parameters.

### [Designing for Extensibility](#)

Discusses extensibility mechanisms such as subclassing, using events, virtual members, and callbacks, and explains how to choose the mechanisms that best meet your framework's requirements.

### [Design Guidelines for Exceptions](#)

Describes design guidelines for designing, throwing, and catching exceptions.

### [Usage Guidelines](#)

Describes guidelines for using common types such as arrays, attributes, and collections, supporting serialization, and overloading equality operators.

### [Common Design Patterns](#)

Provides guidelines for choosing and implementing dependency properties.

*Portions © 2005, 2009 Microsoft Corporation. All rights reserved.*

*Reprinted by permission of Pearson Education, Inc. from [Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries, 2nd Edition](#) by Krzysztof Cwalina and Brad Abrams, published Oct 22, 2008 by Addison-Wesley Professional as part of the Microsoft Windows Development Series.*

## See also

- [Overview](#)

- [Development Guide](#)

# Naming Guidelines

9/20/2022 • 2 minutes to read

Following a consistent set of naming conventions in the development of a framework can be a major contribution to the framework's usability. It allows the framework to be used by many developers on widely separated projects. Beyond consistency of form, names of framework elements must be easily understood and must convey the function of each element.

The goal of this chapter is to provide a consistent set of naming conventions that results in names that make immediate sense to developers.

Although adopting these naming conventions as general code development guidelines would result in more consistent naming throughout your code, you are required only to apply them to APIs that are publicly exposed (public or protected types and members, and explicitly implemented interfaces).

## In This Section

[Capitalization Conventions](#)

[General Naming Conventions](#)

[Names of Assemblies and DLLs](#)

[Names of Namespaces](#)

[Names of Classes, Structs, and Interfaces](#)

[Names of Type Members](#)

[Naming Parameters](#)

[Naming Resources](#)

*Portions © 2005, 2009 Microsoft Corporation. All rights reserved.*

*Reprinted by permission of Pearson Education, Inc. from [Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries, 2nd Edition](#) by Krzysztof Cwalina and Brad Abrams, published Oct 22, 2008 by Addison-Wesley Professional as part of the Microsoft Windows Development Series.*

## See also

- [Framework Design Guidelines](#)

# Capitalization Conventions

9/20/2022 • 2 minutes to read

The guidelines in this chapter lay out a simple method for using case that, when applied consistently, make identifiers for types, members, and parameters easy to read.

## Capitalization Rules for Identifiers

To differentiate words in an identifier, capitalize the first letter of each word in the identifier. Do not use underscores to differentiate words, or for that matter, anywhere in identifiers. There are two appropriate ways to capitalize identifiers, depending on the use of the identifier:

- PascalCasing
- camelCasing

The PascalCasing convention, used for all identifiers except parameter names, capitalizes the first character of each word (including acronyms over two letters in length), as shown in the following examples:

`PropertyDescriptor` `HtmlTag`

A special case is made for two-letter acronyms in which both letters are capitalized, as shown in the following identifier:

`IOStream`

The camelCasing convention, used only for parameter names, capitalizes the first character of each word except the first word, as shown in the following examples. As the example also shows, two-letter acronyms that begin a camel-cased identifier are both lowercase.

`propertyDescriptor` `ioStream` `htmlTag`

- ✓ DO use PascalCasing for all public member, type, and namespace names consisting of multiple words.
- ✓ DO use camelCasing for parameter names.

The following table describes the capitalization rules for different types of identifiers.

IDENTIFIER	CASING	EXAMPLE
Namespace	Pascal	<code>namespace System.Security { ... }</code>
Type	Pascal	<code>public class StreamReader { ... }</code>
Interface	Pascal	<code>public interface IEnumerable { ... }</code>
Method	Pascal	<code>public class Object {</code> <code>    public virtual string</code> <code>        ToString();</code> <code>}</code>

IDENTIFIER	CASING	EXAMPLE
Property	Pascal	<pre>public class String {     public int Length { get; } }</pre>
Event	Pascal	<pre>public class Process {     public event EventHandler         Exited; }</pre>
Field	Pascal	<pre>public class MessageQueue {     public static readonly TimeSpan         InfiniteTimeout; }  public struct UInt32 {     public const Min = 0; }</pre>
Enum value	Pascal	<pre>public enum FileMode {     Append,     ... }</pre>
Parameter	Camel	<pre>public class Convert {     public static intToInt32(string         value); }</pre>

## Capitalizing Compound Words and Common Terms

Most compound terms are treated as single words for purposes of capitalization.

✗ DO NOT capitalize each word in so-called closed-form compound words.

These are compound words written as a single word, such as endpoint. For the purpose of casing guidelines, treat a closed-form compound word as a single word. Use a current dictionary to determine if a compound word is written in closed form.

PASCAL	CAMEL	NOT
BitFlag	bitFlag	Bitflag
Callback	callback	CallBack
Canceled	canceled	Cancelled
DoNot	doNot	Don't
Email	email	EMail
Endpoint	endpoint	EndPoint
FileName	fileName	Filename

PASCAL	CAMEL	NOT
Gridline	gridline	GridLine
Hashtable	hashtable	HashTable
Id	id	ID
Indexes	indexes	Indices
LogOff	logOff	LogOut
LogOn	logOn	LogIn
Metadata	metadata	MetaData, metaData
Multipanel	multipanel	MultiPanel
Multiview	multiview	MultiView
Namespace	namespace	NameSpace
Ok	ok	OK
Pi	pi	PI
Placeholder	placeholder	PlaceHolder
SignIn	signIn	SignOn
SignOut	signOut	SignOff
UserName	userName	Username
WhiteSpace	whiteSpace	Whitespace
Writable	writable	Writeable

## Case Sensitivity

Languages that can run on the CLR are not required to support case-sensitivity, although some do. Even if your language supports it, other languages that might access your framework do not. Any APIs that are externally accessible, therefore, cannot rely on case alone to distinguish between two names in the same context.

✗ DO NOT assume that all programming languages are case sensitive. They are not. Names cannot differ by case alone.

*Portions © 2005, 2009 Microsoft Corporation. All rights reserved.*

*Reprinted by permission of Pearson Education, Inc. from [Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries, 2nd Edition](#) by Krzysztof Cwalina and Brad Abrams, published Oct 22, 2008 by Addison-Wesley Professional as part of the Microsoft Windows Development Series.*

## See also

- [Framework Design Guidelines](#)
- [Naming Guidelines](#)

# General Naming Conventions

9/20/2022 • 3 minutes to read

This section describes general naming conventions that relate to word choice, guidelines on using abbreviations and acronyms, and recommendations on how to avoid using language-specific names.

## Word Choice

- ✓ DO choose easily readable identifier names.

For example, a property named `HorizontalAlignment` is more English-readable than `AlignmentHorizontal`.

- ✓ DO favor readability over brevity.

The property name `CanScrollHorizontally` is better than `ScrollableX` (an obscure reference to the X-axis).

- ✗ DO NOT use underscores, hyphens, or any other nonalphanumeric characters.

- ✗ DO NOT use Hungarian notation.

- ✗ AVOID using identifiers that conflict with keywords of widely used programming languages.

According to Rule 4 of the Common Language Specification (CLS), all compliant languages must provide a mechanism that allows access to named items that use a keyword of that language as an identifier. C#, for example, uses the @ sign as an escape mechanism in this case. However, it is still a good idea to avoid common keywords because it is much more difficult to use a method with the escape sequence than one without it.

## Using Abbreviations and Acronyms

- ✗ DO NOT use abbreviations or contractions as part of identifier names.

For example, use `GetWindow` rather than `GetWin`.

- ✗ DO NOT use any acronyms that are not widely accepted, and even if they are, only when necessary.

## Avoiding Language-Specific Names

- ✓ DO use semantically interesting names rather than language-specific keywords for type names.

For example, `GetLength` is a better name than `GetInt`.

- ✓ DO use a generic CLR type name, rather than a language-specific name, in the rare cases when an identifier has no semantic meaning beyond its type.

For example, a method converting to `Int64` should be named `ToInt64`, not `ToLong` (because `Int64` is a CLR name for the C#-specific alias `long`). The following table presents several base data types using the CLR type names (as well as the corresponding type names for C#, Visual Basic, and C++).

C#	VISUAL BASIC	C++	CLR
<code>sbyte</code>	<code>SByte</code>	<code>char</code>	<code>SByte</code>
<code>byte</code>	<code>Byte</code>	<code>unsigned char</code>	<code>Byte</code>

C#	VISUAL BASIC	C++	CLR
short	Short	short	Int16
ushort	UInt16	unsigned short	UInt16
int	Integer	int	Int32
uint	UInt32	unsigned int	UInt32
long	Long	_int64	Int64
ulong	UInt64	unsigned _int64	UInt64
float	Single	float	Single
double	Double	double	Double
bool	Boolean	bool	Boolean
char	Char	wchar_t	Char
string	String	String	String
object	Object	Object	Object

- ✓ DO use a common name, such as `value` or `item`, rather than repeating the type name, in the rare cases when an identifier has no semantic meaning and the type of the parameter is not important.

## Naming New Versions of Existing APIs

- ✓ DO use a name similar to the old API when creating new versions of an existing API.

This helps to highlight the relationship between the APIs.

- ✓ DO prefer adding a suffix rather than a prefix to indicate a new version of an existing API.

This will assist discovery when browsing documentation, or using IntelliSense. The old version of the API will be organized close to the new APIs, because most browsers and IntelliSense show identifiers in alphabetical order.

- ✓ CONSIDER using a brand new, but meaningful identifier, instead of adding a suffix or a prefix.

- ✓ DO use a numeric suffix to indicate a new version of an existing API, particularly if the existing name of the API is the only name that makes sense (i.e., if it is an industry standard) and if adding any meaningful suffix (or changing the name) is not an appropriate option.

- ✗ DO NOT use the "Ex" (or a similar) suffix for an identifier to distinguish it from an earlier version of the same API.

- ✓ DO use the "64" suffix when introducing versions of APIs that operate on a 64-bit integer (a long integer) instead of a 32-bit integer. You only need to take this approach when the existing 32-bit API exists; don't do it for brand new APIs with only a 64-bit version.

*Portions © 2005, 2009 Microsoft Corporation. All rights reserved.*

*Reprinted by permission of Pearson Education, Inc. from Framework Design Guidelines: Conventions, Idioms,*

*and Patterns for Reusable .NET Libraries, 2nd Edition* by Krzysztof Cwalina and Brad Abrams, published Oct 22, 2008 by Addison-Wesley Professional as part of the Microsoft Windows Development Series.

## See also

- [Framework design guidelines](#)
- [Naming guidelines](#)
- [.NET naming conventions for EditorConfig](#)

# Names of Assemblies and DLLs

9/20/2022 • 2 minutes to read

An assembly is the unit of deployment and identity for managed code programs. Although assemblies can span one or more files, typically an assembly maps one-to-one with a DLL. Therefore, this section describes only DLL naming conventions, which then can be mapped to assembly naming conventions.

- ✓ DO choose names for your assembly DLLs that suggest large chunks of functionality, such as System.Data.

Assembly and DLL names don't have to correspond to namespace names, but it is reasonable to follow the namespace name when naming assemblies. A good rule of thumb is to name the DLL based on the common prefix of the namespaces contained in the assembly. For example, an assembly with two namespaces,

`MyCompany.MyTechnology.FirstFeature` and `MyCompany.MyTechnology.SecondFeature`, could be called `MyCompany.MyTechnology.dll`.

- ✓ CONSIDER naming DLLs according to the following pattern:

`<Company>.<Component>.dll`

where `<Component>` contains one or more dot-separated clauses. For example:

`Litware.Controls.dll`.

*Portions © 2005, 2009 Microsoft Corporation. All rights reserved.*

*Reprinted by permission of Pearson Education, Inc. from [Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries, 2nd Edition](#) by Krzysztof Cwalina and Brad Abrams, published Oct 22, 2008 by Addison-Wesley Professional as part of the Microsoft Windows Development Series.*

## See also

- [Framework Design Guidelines](#)
- [Naming Guidelines](#)

# Names of Namespaces

9/20/2022 • 3 minutes to read

As with other naming guidelines, the goal when naming namespaces is creating sufficient clarity for the programmer using the framework to immediately know what the content of the namespace is likely to be. The following template specifies the general rule for naming namespaces:

```
<Company>.(<Product>|<Technology>) [.<Feature>] [ .<Subnamespace>]
```

The following are examples:

```
Fabrikam.Math Litware.Security
```

- ✓ DO prefix namespace names with a company name to prevent namespaces from different companies from having the same name.
- ✓ DO use a stable, version-independent product name at the second level of a namespace name.
- ✗ DO NOT use organizational hierarchies as the basis for names in namespace hierarchies, because group names within corporations tend to be short-lived. Organize the hierarchy of namespaces around groups of related technologies.
- ✓ DO use PascalCasing, and separate namespace components with periods (e.g., `Microsoft.Office.PowerPoint`). If your brand employs nontraditional casing, you should follow the casing defined by your brand, even if it deviates from normal namespace casing.
- ✓ CONSIDER using plural namespace names where appropriate.

For example, use `System.Collections` instead of `System.Collection`. Brand names and acronyms are exceptions to this rule, however. For example, use `System.IO` instead of `System.IOs`.

✗ DO NOT use the same name for a namespace and a type in that namespace.

For example, do not use `Debug` as a namespace name and then also provide a class named `Debug` in the same namespace. Several compilers require such types to be fully qualified.

## Namespaces and Type Name Conflicts

✗ DO NOT introduce generic type names such as `Element`, `Node`, `Log`, and `Message`.

There is a very high probability that doing so will lead to type name conflicts in common scenarios. You should qualify the generic type names (`FormElement`, `XmlNode`, `EventLog`, `SoapMessage`).

There are specific guidelines for avoiding type name conflicts for different categories of namespaces.

- **Application model namespaces**

Namespaces belonging to a single application model are very often used together, but they are almost never used with namespaces of other application models. For example, the `System.Windows.Forms` namespace is very rarely used together with the `System.Web.UI` namespace. The following is a list of well-known application model namespace groups:

```
System.Windows* System.Web.UI*
```

✗ DO NOT give the same name to types in namespaces within a single application model.

For example, do not add a type named `Page` to the `System.Web.UI.Adapters` namespace, because the

`System.Web.UI` namespace already contains a type named `Page`.

- **Infrastructure namespaces**

This group contains namespaces that are rarely imported during development of common applications. For example, `.Design` namespaces are mainly used when developing programming tools. Avoiding conflicts with types in these namespaces is not critical.

- **Core namespaces**

Core namespaces include all `System` namespaces, excluding namespaces of the application models and the Infrastructure namespaces. Core namespaces include, among others, `System`, `System.IO`, `System.Xml`, and `System.Net`.

✗ DO NOT give types names that would conflict with any type in the Core namespaces.

For example, never use `Stream` as a type name. It would conflict with `System.IO.Stream`, a very commonly used type.

- **Technology namespace groups**

This category includes all namespaces with the same first two namespace nodes (`<Company>.<Technology>*`), such as `Microsoft.Build.Utilities` and `Microsoft.Build.Tasks`. It is important that types belonging to a single technology do not conflict with each other.

✗ DO NOT assign type names that would conflict with other types within a single technology.

✗ DO NOT introduce type name conflicts between types in technology namespaces and an application model namespace (unless the technology is not intended to be used with the application model).

*Portions © 2005, 2009 Microsoft Corporation. All rights reserved.*

*Reprinted by permission of Pearson Education, Inc. from [Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries, 2nd Edition](#) by Krzysztof Cwalina and Brad Abrams, published Oct 22, 2008 by Addison-Wesley Professional as part of the Microsoft Windows Development Series.*

## See also

- [Framework Design Guidelines](#)
- [Naming Guidelines](#)

# Names of Classes, Structs, and Interfaces

9/20/2022 • 3 minutes to read

The naming guidelines that follow apply to general type naming.

- ✓ DO name classes and structs with nouns or noun phrases, using PascalCasing.

This distinguishes type names from methods, which are named with verb phrases.

- ✓ DO name interfaces with adjective phrases, or occasionally with nouns or noun phrases.

Nouns and noun phrases should be used rarely and they might indicate that the type should be an abstract class, and not an interface.

- ✗ DO NOT give class names a prefix (e.g., "C").

- ✓ CONSIDER ending the name of derived classes with the name of the base class.

This is very readable and explains the relationship clearly. Some examples of this in code are:

`ArgumentOutOfRangeException`, which is a kind of `Exception`, and `SerializableAttribute`, which is a kind of `Attribute`. However, it is important to use reasonable judgment in applying this guideline; for example, the `Button` class is a kind of `Control` event, although `Control` doesn't appear in its name.

- ✓ DO prefix interface names with the letter I, to indicate that the type is an interface.

For example, `IComponent` (descriptive noun), `ICustomAttributeProvider` (noun phrase), and `IPersistable` (adjective) are appropriate interface names. As with other type names, avoid abbreviations.

- ✓ DO ensure that the names differ only by the "I" prefix on the interface name when you are defining a class-interface pair where the class is a standard implementation of the interface.

## Names of Generic Type Parameters

Generics were added to .NET Framework 2.0. The feature introduced a new kind of identifier called *type parameter*.

- ✓ DO name generic type parameters with descriptive names unless a single-letter name is completely self-explanatory and a descriptive name would not add value.

- ✓ CONSIDER using `T` as the type parameter name for types with one single-letter type parameter.

```
public int IComparer<T> { ... }
public delegate bool Predicate<T>(T item);
public struct Nullable<T> where T:struct { ... }
```

- ✓ DO prefix descriptive type parameter names with `T`.

```
public interface ISessionChannel<TSession> where TSession : ISession {
    TSession Session { get; }
}
```

- ✓ CONSIDER indicating constraints placed on a type parameter in the name of the parameter.

For example, a parameter constrained to `ISession` might be called `TSession`.

## Names of Common Types

- ✓ DO follow the guidelines described in the following table when naming types derived from or implementing certain .NET Framework types.

BASE TYPE	DERIVED/IMPLEMENTING TYPE GUIDELINE
<code>System.Attribute</code>	✓ DO add the suffix "Attribute" to names of custom attribute classes.
<code>System.Delegate</code>	<ul style="list-style-type: none"><li>✓ DO add the suffix "EventHandler" to names of delegates that are used in events.</li><li>✓ DO add the suffix "Callback" to names of delegates other than those used as event handlers.</li><li>✗ DO NOT add the suffix "Delegate" to a delegate.</li></ul>
<code>System.EventArgs</code>	✓ DO add the suffix "EventArgs."
<code>System.Enum</code>	<ul style="list-style-type: none"><li>✗ DO NOT derive from this class; use the keyword supported by your language instead; for example, in C#, use the <code>enum</code> keyword.</li><li>✗ DO NOT add the suffix "Enum" or "Flag."</li></ul>
<code>System.Exception</code>	✓ DO add the suffix "Exception."
<code>IDictionary</code> <code>IDictionary&lt;TKey, TValue&gt;</code>	✓ DO add the suffix "Dictionary." Note that <code>IDictionary</code> is a specific type of collection, but this guideline takes precedence over the more general collections guideline that follows.
<code>IEnumerable</code> <code>ICollection</code> <code>IList</code> <code>IEnumerable&lt;T&gt;</code> <code>ICollection&lt;T&gt;</code> <code>IList&lt;T&gt;</code>	✓ DO add the suffix "Collection."
<code>System.IO.Stream</code>	✓ DO add the suffix "Stream."
<code>CodeAccessPermission</code> <code>IPermission</code>	✓ DO add the suffix "Permission."

## Naming Enumerations

Names of enumeration types (also called enums) in general should follow the standard type-naming rules (PascalCasing, etc.). However, there are additional guidelines that apply specifically to enums.

- ✓ DO use a singular type name for an enumeration unless its values are bit fields.
- ✓ DO use a plural type name for an enumeration with bit fields as values, also called flags enum.
- ✗ DO NOT use an "Enum" suffix in enum type names.
- ✗ DO NOT use "Flag" or "Flags" suffixes in enum type names.

✖ DO NOT use a prefix on enumeration value names (e.g., "ad" for ADO enums, "rtf" for rich text enums, etc.).

*Portions © 2005, 2009 Microsoft Corporation. All rights reserved.*

*Reprinted by permission of Pearson Education, Inc. from [Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries, 2nd Edition](#) by Krzysztof Cwalina and Brad Abrams, published Oct 22, 2008 by Addison-Wesley Professional as part of the Microsoft Windows Development Series.*

## See also

- [Framework Design Guidelines](#)
- [Naming Guidelines](#)

# Names of Type Members

9/20/2022 • 3 minutes to read

Types are made of members: methods, properties, events, constructors, and fields. The following sections describe guidelines for naming type members.

## Names of Methods

Because methods are the means of taking action, the design guidelines require that method names be verbs or verb phrases. Following this guideline also serves to distinguish method names from property and type names, which are noun or adjective phrases.

- ✓ DO give methods names that are verbs or verb phrases.

```
public class String {  
    public int CompareTo(...);  
    public string[] Split(...);  
    public string Trim();  
}
```

## Names of Properties

Unlike other members, properties should be given noun phrase or adjective names. That is because a property refers to data, and the name of the property reflects that. PascalCasing is always used for property names.

- ✓ DO name properties using a noun, noun phrase, or adjective.
- ✗ DO NOT have properties that match the name of "Get" methods as in the following example:

```
public string TextWriter { get {...} set {...} } public string GetTextWriter(int value) { ... }
```

This pattern typically indicates that the property should really be a method.

- ✓ DO name collection properties with a plural phrase describing the items in the collection instead of using a singular phrase followed by "List" or "Collection".
- ✓ DO name Boolean properties with an affirmative phrase (`CanSeek` instead of `CantSeek`). Optionally, you can also prefix Boolean properties with "Is", "Can", or "Has", but only where it adds value.
- ✓ CONSIDER giving a property the same name as its type.

For example, the following property correctly gets and sets an enum value named `Color`, so the property is named `Color`:

```
public enum Color {...}  
public class Control {  
    public Color Color { get {...} set {...} }  
}
```

## Names of Events

Events always refer to some action, either one that is happening or one that has occurred. Therefore, as with methods, events are named with verbs, and verb tense is used to indicate the time when the event is raised.

- ✓ DO name events with a verb or a verb phrase.

Examples include `Clicked`, `Painting`, `DroppedDown`, and so on.

- ✓ DO give events names with a concept of before and after, using the present and past tenses.

For example, a close event that is raised before a window is closed would be called `closing`, and one that is raised after the window is closed would be called `Closed`.

✗ DO NOT use "Before" or "After" prefixes or postfixes to indicate pre- and post-events. Use present and past tenses as just described.

- ✓ DO name event handlers (delegates used as types of events) with the "EventHandler" suffix, as shown in the following example:

```
public delegate void ClickedEventHandler(object sender, ClickedEventArgs e);
```

- ✓ DO use two parameters named `sender` and `e` in event handlers.

The `sender` parameter represents the object that raised the event. The `sender` parameter is typically of type `object`, even if it is possible to employ a more specific type.

- ✓ DO name event argument classes with the "EventArgs" suffix.

## Names of Fields

The field-naming guidelines apply to static public and protected fields. Internal and private fields are not covered by guidelines, and public or protected instance fields are not allowed by the [member design guidelines](#).

- ✓ DO use PascalCasing in field names.

- ✓ DO name fields using a noun, noun phrase, or adjective.

- ✗ DO NOT use a prefix for field names.

For example, do not use `"g_"` or `"s_"` to indicate static fields.

*Portions © 2005, 2009 Microsoft Corporation. All rights reserved.*

*Reprinted by permission of Pearson Education, Inc. from [Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries, 2nd Edition](#) by Krzysztof Cwalina and Brad Abrams, published Oct 22, 2008 by Addison-Wesley Professional as part of the Microsoft Windows Development Series.*

## See also

- [Framework Design Guidelines](#)
- [Naming Guidelines](#)

# Naming Parameters

9/20/2022 • 2 minutes to read

Beyond the obvious reason of readability, it is important to follow the guidelines for parameter names because parameters are displayed in documentation and in the designer when visual design tools provide Intellisense and class browsing functionality.

- ✓ DO use camelCasing in parameter names.
- ✓ DO use descriptive parameter names.
- ✓ CONSIDER using names based on a parameter's meaning rather than the parameter's type.

## Naming Operator Overload Parameters

- ✓ DO use `left` and `right` for binary operator overload parameter names if there is no meaning to the parameters.
- ✓ DO use `value` for unary operator overload parameter names if there is no meaning to the parameters.
- ✓ CONSIDER meaningful names for operator overload parameters if doing so adds significant value.
- ✗ DO NOT use abbreviations or numeric indices for operator overload parameter names.

*Portions © 2005, 2009 Microsoft Corporation. All rights reserved.*

*Reprinted by permission of Pearson Education, Inc. from [Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries, 2nd Edition](#) by Krzysztof Cwalina and Brad Abrams, published Oct 22, 2008 by Addison-Wesley Professional as part of the Microsoft Windows Development Series.*

## See also

- [Framework Design Guidelines](#)
- [Naming Guidelines](#)

# Naming Resources

9/20/2022 • 2 minutes to read

Because localizable resources can be referenced via certain objects as if they were properties, the naming guidelines for resources are similar to property guidelines.

- ✓ DO use PascalCasing in resource keys.
- ✓ DO provide descriptive rather than short identifiers.
- ✗ DO NOT use language-specific keywords of the main CLR languages.
- ✓ DO use only alphanumeric characters and underscores in naming resources.
- ✓ DO use the following naming convention for exception message resources.

The resource identifier should be the exception type name plus a short identifier of the exception:

[ArgumentExceptionIllegalCharacters](#) [ArgumentExceptionInvalidName](#) [ArgumentExceptionFileNameIsMalformed](#)

*Portions © 2005, 2009 Microsoft Corporation. All rights reserved.*

*Reprinted by permission of Pearson Education, Inc. from [Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries, 2nd Edition](#) by Krzysztof Cwalina and Brad Abrams, published Oct 22, 2008 by Addison-Wesley Professional as part of the Microsoft Windows Development Series.*

## See also

- [Framework Design Guidelines](#)
- [Naming Guidelines](#)

# Type design guidelines

9/20/2022 • 2 minutes to read

From the CLR perspective, there are only two categories of types—reference types and value types—but for the purpose of a discussion about framework design, we divide types into more logical groups, each with its own specific design rules.

Classes are the general case of reference types. They make up the bulk of types in the majority of frameworks. Classes owe their popularity to the rich set of object-oriented features they support and to their general applicability. Base classes and abstract classes are special logical groups related to extensibility.

Interfaces are types that can be implemented by both reference types and value types. They can thus serve as roots of polymorphic hierarchies of reference types and value types. In addition, interfaces can be used to simulate multiple inheritance, which is not natively supported by the CLR.

Structs are the general case of value types and should be reserved for small, simple types, similar to language primitives.

Enums are a special case of value types used to define short sets of values, such as days of the week, console colors, and so on.

Static classes are types intended to be containers for static members. They are commonly used to provide shortcuts to other operations.

Delegates, exceptions, attributes, arrays, and collections are all special cases of reference types intended for specific uses, and guidelines for their design and usage are discussed elsewhere in this book.

✓ DO ensure that each type is a well-defined set of related members, not just a random collection of unrelated functionality.

## In this section

[Choosing Between Class and Struct](#)

[Abstract Class Design](#)

[Static Class Design](#)

[Interface Design](#)

[Struct Design](#)

[Enum Design](#)

[Nested Types](#)

*Portions © 2005, 2009 Microsoft Corporation. All rights reserved.*

*Reprinted by permission of Pearson Education, Inc. from [Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries, 2nd Edition](#) by Krzysztof Cwalina and Brad Abrams, published Oct 22, 2008 by Addison-Wesley Professional as part of the Microsoft Windows Development Series.*

## See also

- [Framework Design Guidelines](#)

# Choosing Between Class and Struct

9/20/2022 • 2 minutes to read

One of the basic design decisions every framework designer faces is whether to design a type as a class (a reference type) or as a struct (a value type). Good understanding of the differences in the behavior of reference types and value types is crucial in making this choice.

The first difference between reference types and value types we will consider is that reference types are allocated on the heap and garbage-collected, whereas value types are allocated either on the stack or inline in containing types and deallocated when the stack unwinds or when their containing type gets deallocated. Therefore, allocations and deallocations of value types are in general cheaper than allocations and deallocations of reference types.

Next, arrays of reference types are allocated out-of-line, meaning the array elements are just references to instances of the reference type residing on the heap. Value type arrays are allocated inline, meaning that the array elements are the actual instances of the value type. Therefore, allocations and deallocations of value type arrays are much cheaper than allocations and deallocations of reference type arrays. In addition, in a majority of cases value type arrays exhibit much better locality of reference.

The next difference is related to memory usage. Value types get boxed when cast to a reference type or one of the interfaces they implement. They get unboxed when cast back to the value type. Because boxes are objects that are allocated on the heap and are garbage-collected, too much boxing and unboxing can have a negative impact on the heap, the garbage collector, and ultimately the performance of the application. In contrast, no such boxing occurs as reference types are cast. (For more information, see [Boxing and Unboxing](#)).

Next, reference type assignments copy the reference, whereas value type assignments copy the entire value. Therefore, assignments of large reference types are cheaper than assignments of large value types.

Finally, reference types are passed by reference, whereas value types are passed by value. Changes to an instance of a reference type affect all references pointing to the instance. Value type instances are copied when they are passed by value. When an instance of a value type is changed, it of course does not affect any of its copies. Because the copies are not created explicitly by the user but are implicitly created when arguments are passed or return values are returned, value types that can be changed can be confusing to many users. Therefore, value types should be immutable.

As a rule of thumb, the majority of types in a framework should be classes. There are, however, some situations in which the characteristics of a value type make it more appropriate to use structs.

✓ CONSIDER defining a struct instead of a class if instances of the type are small and commonly short-lived or are commonly embedded in other objects.

✗ AVOID defining a struct unless the type has all of the following characteristics:

- It logically represents a single value, similar to primitive types (`int`, `double`, etc.).
- It has an instance size under 16 bytes.
- It is immutable.
- It will not have to be boxed frequently.

In all other cases, you should define your types as classes.

*Reprinted by permission of Pearson Education, Inc. from Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries, 2nd Edition* by Krzysztof Cwalina and Brad Abrams, published Oct 22, 2008 by Addison-Wesley Professional as part of the Microsoft Windows Development Series.

## See also

- [Type Design Guidelines](#)
- [Framework Design Guidelines](#)

# Abstract Class Design

9/20/2022 • 2 minutes to read

- ✗ DO NOT define public or protected internal constructors in abstract types.

Constructors should be public only if users will need to create instances of the type. Because you cannot create instances of an abstract type, an abstract type with a public constructor is incorrectly designed and misleading to the users.

- ✓ DO define a protected or an internal constructor in abstract classes.

A protected constructor is more common and simply allows the base class to do its own initialization when subtypes are created.

An internal constructor can be used to limit concrete implementations of the abstract class to the assembly defining the class.

- ✓ DO provide at least one concrete type that inherits from each abstract class that you ship.

Doing this helps to validate the design of the abstract class. For example, [System.IO.FileStream](#) is an implementation of the [System.IO.Stream](#) abstract class.

*Portions © 2005, 2009 Microsoft Corporation. All rights reserved.*

*Reprinted by permission of Pearson Education, Inc. from [Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries, 2nd Edition](#) by Krzysztof Cwalina and Brad Abrams, published Oct 22, 2008 by Addison-Wesley Professional as part of the Microsoft Windows Development Series.*

## See also

- [Type Design Guidelines](#)
- [Framework Design Guidelines](#)

# Static Class Design

9/20/2022 • 2 minutes to read

A static class is defined as a class that contains only static members (of course besides the instance members inherited from [System.Object](#) and possibly a private constructor). Some languages provide built-in support for static classes. In C# 2.0 and later, when a class is declared to be static, it is sealed, abstract, and no instance members can be overridden or declared.

Static classes are a compromise between pure object-oriented design and simplicity. They are commonly used to provide shortcuts to other operations (such as [System.IO.File](#)), holders of extension methods, or functionality for which a full object-oriented wrapper is unwarranted (such as [System.Environment](#)).

- ✓ DO use static classes sparingly.

Static classes should be used only as supporting classes for the object-oriented core of the framework.

- ✗ DO NOT treat static classes as a miscellaneous bucket.

- ✗ DO NOT declare or override instance members in static classes.

- ✓ DO declare static classes as sealed, abstract, and add a private instance constructor if your programming language does not have built-in support for static classes.

*Portions © 2005, 2009 Microsoft Corporation. All rights reserved.*

*Reprinted by permission of Pearson Education, Inc. from [Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries, 2nd Edition](#) by Krzysztof Cwalina and Brad Abrams, published Oct 22, 2008 by Addison-Wesley Professional as part of the Microsoft Windows Development Series.*

## See also

- [Type Design Guidelines](#)
- [Framework Design Guidelines](#)

# Interface Design

9/20/2022 • 2 minutes to read

Although most APIs are best modeled using classes and structs, there are cases in which interfaces are more appropriate or are the only option.

The CLR does not support multiple inheritance (i.e., CLR classes cannot inherit from more than one base class), but it does allow types to implement one or more interfaces in addition to inheriting from a base class.

Therefore, interfaces are often used to achieve the effect of multiple inheritance. For example, [IDisposable](#) is an interface that allows types to support disposability independent of any other inheritance hierarchy in which they want to participate.

The other situation in which defining an interface is appropriate is in creating a common interface that can be supported by several types, including some value types. Value types cannot inherit from types other than [ValueType](#), but they can implement interfaces, so using an interface is the only option in order to provide a common base type.

- ✓ DO define an interface if you need some common API to be supported by a set of types that includes value types.
- ✓ CONSIDER defining an interface if you need to support its functionality on types that already inherit from some other type.
- ✗ AVOID using marker interfaces (interfaces with no members).

If you need to mark a class as having a specific characteristic (marker), in general, use a custom attribute rather than an interface.

- ✓ DO provide at least one type that is an implementation of an interface.

Doing this helps to validate the design of the interface. For example, [List<T>](#) is an implementation of the [IList<T>](#) interface.

- ✓ DO provide at least one API that consumes each interface you define (a method taking the interface as a parameter or a property typed as the interface).

Doing this helps to validate the interface design. For example, [List<T>.Sort](#) consumes the [System.Collections.Generic.IComparer<T>](#) interface.

- ✗ DO NOT add members to an interface that has previously shipped.

Doing so would break implementations of the interface. You should create a new interface in order to avoid versioning problems.

Except for the situations described in these guidelines, you should, in general, choose classes rather than interfaces in designing managed code reusable libraries.

*Portions © 2005, 2009 Microsoft Corporation. All rights reserved.*

*Reprinted by permission of Pearson Education, Inc. from [Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries, 2nd Edition](#) by Krzysztof Cwalina and Brad Abrams, published Oct 22, 2008 by Addison-Wesley Professional as part of the Microsoft Windows Development Series.*

## See also

- Type Design Guidelines
- Framework Design Guidelines

# Struct Design

9/20/2022 • 2 minutes to read

The general-purpose value type is most often referred to as a struct, its C# keyword. This section provides guidelines for general struct design.

- ✖ DO NOT provide a parameterless constructor for a struct.

Following this guideline allows arrays of structs to be created without having to run the constructor on each item of the array. Notice that C# does not allow structs to have parameterless constructors.

- ✖ DO NOT define mutable value types.

Mutable value types have several problems. For example, when a property getter returns a value type, the caller receives a copy. Because the copy is created implicitly, developers might not be aware that they are mutating the copy, and not the original value. Also, some languages (dynamic languages, in particular) have problems using mutable value types because even local variables, when dereferenced, cause a copy to be made.

- ✓ DO ensure that a state where all instance data is set to zero, false, or null (as appropriate) is valid.

This prevents accidental creation of invalid instances when an array of the structs is created.

- ✓ DO implement [IEquatable<T>](#) on value types.

The [Object.Equals](#) method on value types causes boxing, and its default implementation is not very efficient, because it uses reflection. [Equals](#) can have much better performance and can be implemented so that it will not cause boxing.

- ✖ DO NOT explicitly extend [ValueType](#). In fact, most languages prevent this.

In general, structs can be very useful but should only be used for small, single, immutable values that will not be boxed frequently.

*Portions © 2005, 2009 Microsoft Corporation. All rights reserved.*

*Reprinted by permission of Pearson Education, Inc. from [Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries, 2nd Edition](#) by Krzysztof Cwalina and Brad Abrams, published Oct 22, 2008 by Addison-Wesley Professional as part of the Microsoft Windows Development Series.*

## See also

- [Type Design Guidelines](#)
- [Framework Design Guidelines](#)
- [Choosing Between Class and Struct](#)

# Enum Design

9/20/2022 • 4 minutes to read

Enums are a special kind of value type. There are two kinds of enums: simple enums and flag enums.

Simple enums represent small closed sets of choices. A common example of the simple enum is a set of colors.

Flag enums are designed to support bitwise operations on the enum values. A common example of the flags enum is a list of options.

- ✓ DO use an enum to strongly type parameters, properties, and return values that represent sets of values.
- ✓ DO favor using an enum instead of static constants.
- ✗ DO NOT use an enum for open sets (such as the operating system version, names of your friends, etc.).
- ✗ DO NOT provide reserved enum values that are intended for future use.

You can always simply add values to the existing enum at a later stage. See [Adding Values to Enums](#) for more details on adding values to enums. Reserved values just pollute the set of real values and tend to lead to user errors.

✗ AVOID publicly exposing enums with only one value.

A common practice for ensuring future extensibility of C APIs is to add reserved parameters to method signatures. Such reserved parameters can be expressed as enums with a single default value. This should not be done in managed APIs. Method overloading allows adding parameters in future releases.

✗ DO NOT include sentinel values in enums.

Although they are sometimes helpful to framework developers, sentinel values are confusing to users of the framework. They are used to track the state of the enum rather than being one of the values from the set represented by the enum.

✓ DO provide a value of zero on simple enums.

Consider calling the value something like "None." If such a value is not appropriate for this particular enum, the most common default value for the enum should be assigned the underlying value of zero.

✓ CONSIDER using [Int32](#) (the default in most programming languages) as the underlying type of an enum unless any of the following is true:

- The enum is a flags enum and you have more than 32 flags, or expect to have more in the future.
- The underlying type needs to be different than [Int32](#) for easier interoperability with unmanaged code expecting different-size enums.
- A smaller underlying type would result in substantial savings in space. If you expect the enum to be used mainly as an argument for flow of control, the size makes little difference. The size savings might be significant if:
  - You expect the enum to be used as a field in a very frequently instantiated structure or class.
  - You expect users to create large arrays or collections of the enum instances.
  - You expect a large number of instances of the enum to be serialized.

For in-memory usage, be aware that managed objects are always `DWORD`-aligned, so you effectively need multiple enums or other small structures in an instance to pack a smaller enum with in order to make a difference, because the total instance size is always going to be rounded up to a `DWORD`.

✓ DO name flag enums with plural nouns or noun phrases and simple enums with singular nouns or noun phrases.

✗ DO NOT extend `System.Enum` directly.

`System.Enum` is a special type used by the CLR to create user-defined enumerations. Most programming languages provide a programming element that gives you access to this functionality. For example, in C# the `enum` keyword is used to define an enumeration.

### Designing Flag Enums

✓ DO apply the `System.FlagsAttribute` to flag enums. Do not apply this attribute to simple enums.

✓ DO use powers of two for the flag enum values so they can be freely combined using the bitwise OR operation.

✓ CONSIDER providing special enum values for commonly used combinations of flags.

Bitwise operations are an advanced concept and should not be required for simple tasks. `ReadWrite` is an example of such a special value.

✗ AVOID creating flag enums where certain combinations of values are invalid.

✗ AVOID using flag enum values of zero unless the value represents "all flags are cleared" and is named appropriately, as prescribed by the next guideline.

✓ DO name the zero value of flag enums `None`. For a flag enum, the value must always mean "all flags are cleared."

### Adding Value to Enums

It is very common to discover that you need to add values to an enum after you have already shipped it. There is a potential application compatibility problem when the newly added value is returned from an existing API, because poorly written applications might not handle the new value correctly.

✓ CONSIDER adding values to enums, despite a small compatibility risk.

If you have real data about application incompatibilities caused by additions to an enum, consider adding a new API that returns the new and old values, and deprecate the old API, which should continue returning just the old values. This will ensure that your existing applications remain compatible.

*Portions © 2005, 2009 Microsoft Corporation. All rights reserved.*

*Reprinted by permission of Pearson Education, Inc. from [Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries, 2nd Edition](#) by Krzysztof Cwalina and Brad Abrams, published Oct 22, 2008 by Addison-Wesley Professional as part of the Microsoft Windows Development Series.*

## See also

- [Type Design Guidelines](#)
- [Framework Design Guidelines](#)

# Nested Types

9/20/2022 • 2 minutes to read

A nested type is a type defined within the scope of another type, which is called the enclosing type. A nested type has access to all members of its enclosing type. For example, it has access to private fields defined in the enclosing type and to protected fields defined in all descendants of the enclosing type.

In general, nested types should be used sparingly. There are several reasons for this. Some developers are not fully familiar with the concept. These developers might, for example, have problems with the syntax of declaring variables of nested types. Nested types are also very tightly coupled with their enclosing types, and as such are not suited to be general-purpose types.

Nested types are best suited for modeling implementation details of their enclosing types. The end user should rarely have to declare variables of a nested type and almost never should have to explicitly instantiate nested types. For example, the enumerator of a collection can be a nested type of that collection. Enumerators are usually instantiated by their enclosing type, and because many languages support the foreach statement, enumerator variables rarely have to be declared by the end user.

✓ DO use nested types when the relationship between the nested type and its outer type is such that member-accessibility semantics are desirable.

✗ DO NOT use public nested types as a logical grouping construct; use namespaces for this.

✗ AVOID publicly exposed nested types. The only exception to this is if variables of the nested type need to be declared only in rare scenarios such as subclassing or other advanced customization scenarios.

✗ DO NOT use nested types if the type is likely to be referenced outside of the containing type.

For example, an enum passed to a method defined on a class should not be defined as a nested type in the class.

✗ DO NOT use nested types if they need to be instantiated by client code. If a type has a public constructor, it should probably not be nested.

If a type can be instantiated, that seems to indicate the type has a place in the framework on its own (you can create it, work with it, and destroy it without ever using the outer type), and thus should not be nested. Inner types should not be widely reused outside of the outer type without any relationship whatsoever to the outer type.

✗ DO NOT define a nested type as a member of an interface. Many languages do not support such a construct.

*Portions © 2005, 2009 Microsoft Corporation. All rights reserved.*

*Reprinted by permission of Pearson Education, Inc. from [Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries, 2nd Edition](#) by Krzysztof Cwalina and Brad Abrams, published Oct 22, 2008 by Addison-Wesley Professional as part of the Microsoft Windows Development Series.*

## See also

- [Type Design Guidelines](#)
- [Framework Design Guidelines](#)

# Member Design Guidelines

9/20/2022 • 2 minutes to read

Methods, properties, events, constructors, and fields are collectively referred to as members. Members are ultimately the means by which framework functionality is exposed to the end users of a framework.

Members can be virtual or nonvirtual, concrete or abstract, static or instance, and can have several different scopes of accessibility. All this variety provides incredible expressiveness but at the same time requires care on the part of the framework designer.

This chapter offers basic guidelines that should be followed when designing members of any type.

## In This Section

[Member Overloading](#)

[Property Design](#)

[Constructor Design](#)

[Event Design](#)

[Field Design](#)

[Extension Methods](#)

[Operator Overloads](#)

[Parameter Design](#)

*Portions © 2005, 2009 Microsoft Corporation. All rights reserved.*

*Reprinted by permission of Pearson Education, Inc. from [Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries, 2nd Edition](#) by Krzysztof Cwalina and Brad Abrams, published Oct 22, 2008 by Addison-Wesley Professional as part of the Microsoft Windows Development Series.*

## See also

- [Framework Design Guidelines](#)

# Member Overloading

9/20/2022 • 2 minutes to read

Member overloading means creating two or more members on the same type that differ only in the number or type of parameters but have the same name. For example, in the following, the `WriteLine` method is overloaded:

```
public static class Console {  
    public void WriteLine();  
    public void WriteLine(string value);  
    public void WriteLine(bool value);  
    ...  
}
```

Because only methods, constructors, and indexed properties can have parameters, only those members can be overloaded.

Overloading is one of the most important techniques for improving usability, productivity, and readability of reusable libraries. Overloading on the number of parameters makes it possible to provide simpler versions of constructors and methods. Overloading on the parameter type makes it possible to use the same member name for members performing identical operations on a selected set of different types.

- ✓ DO try to use descriptive parameter names to indicate the default used by shorter overloads.
- ✗ AVOID arbitrarily varying parameter names in overloads. If a parameter in one overload represents the same input as a parameter in another overload, the parameters should have the same name.
- ✗ AVOID being inconsistent in the ordering of parameters in overloaded members. Parameters with the same name should appear in the same position in all overloads.
- ✓ DO make only the longest overload virtual (if extensibility is required). Shorter overloads should simply call through to a longer overload.
- ✗ DO NOT use `ref` or `out` modifiers to overload members.

Some languages cannot resolve calls to overloads like this. In addition, such overloads usually have completely different semantics and probably should not be overloads but two separate methods instead.

- ✗ DO NOT have overloads with parameters at the same position and similar types yet with different semantics.
- ✓ DO allow `null` to be passed for optional arguments.
- ✓ DO use member overloading rather than defining members with default arguments.

Default arguments are not CLS compliant.

*Portions © 2005, 2009 Microsoft Corporation. All rights reserved.*

*Reprinted by permission of Pearson Education, Inc. from [Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries, 2nd Edition](#) by Krzysztof Cwalina and Brad Abrams, published Oct 22, 2008 by Addison-Wesley Professional as part of the Microsoft Windows Development Series.*

## See also

- [Member Design Guidelines](#)

- Framework Design Guidelines

# Property Design

9/20/2022 • 4 minutes to read

Although properties are technically very similar to methods, they are quite different in terms of their usage scenarios. They should be seen as smart fields. They have the calling syntax of fields, and the flexibility of methods.

- ✓ DO create get-only properties if the caller should not be able to change the value of the property.

Keep in mind that if the type of the property is a mutable reference type, the property value can be changed even if the property is get-only.

- ✗ DO NOT provide set-only properties or properties with the setter having broader accessibility than the getter.

For example, do not use properties with a public setter and a protected getter.

If the property getter cannot be provided, implement the functionality as a method instead. Consider starting the method name with `Set` and follow with what you would have named the property. For example, `AppDomain` has a method called `SetCachePath` instead of having a set-only property called `CachePath`.

- ✓ DO provide sensible default values for all properties, ensuring that the defaults do not result in a security hole or terribly inefficient code.

- ✓ DO allow properties to be set in any order even if this results in a temporary invalid state of the object.

It is common for two or more properties to be interrelated to a point where some values of one property might be invalid given the values of other properties on the same object. In such cases, exceptions resulting from the invalid state should be postponed until the interrelated properties are actually used together by the object.

- ✓ DO preserve the previous value if a property setter throws an exception.

- ✗ AVOID throwing exceptions from property getters.

Property getters should be simple operations and should not have any preconditions. If a getter can throw an exception, it should probably be redesigned to be a method. Notice that this rule does not apply to indexers, where we do expect exceptions as a result of validating the arguments.

## Indexed Property Design

An indexed property is a special property that can have parameters and can be called with special syntax similar to array indexing.

Indexed properties are commonly referred to as indexers. Indexers should be used only in APIs that provide access to items in a logical collection. For example, a string is a collection of characters, and the indexer on `System.String` was added to access its characters.

- ✓ CONSIDER using indexers to provide access to data stored in an internal array.

- ✓ CONSIDER providing indexers on types representing collections of items.

- ✗ AVOID using indexed properties with more than one parameter.

If the design requires multiple parameters, reconsider whether the property really represents an accessor to a logical collection. If it does not, use methods instead. Consider starting the method name with `Get` or `Set`.

- ✗ AVOID indexers with parameter types other than `System.Int32`, `System.Int64`, `System.String`, `System.Object`,

or an enum.

If the design requires other types of parameters, strongly reevaluate whether the API really represents an accessor to a logical collection. If it does not, use a method. Consider starting the method name with `Get` or `Set`.

- ✓ DO use the name `Item` for indexed properties unless there is an obviously better name (e.g., see the `Chars[]` property on `System.String`).

In C#, indexers are by default named `Item`. The `IndexerNameAttribute` can be used to customize this name.

✗ DO NOT provide both an indexer and methods that are semantically equivalent.

✗ DO NOT provide more than one family of overloaded indexers in one type.

This is enforced by the C# compiler.

✗ DO NOT use nondefault indexed properties.

This is enforced by the C# compiler.

### Property Change Notification Events

Sometimes it is useful to provide an event notifying the user of changes in a property value. For example, `System.Windows.Forms.Control` raises a `TextChanged` event after the value of its `Text` property has changed.

- ✓ CONSIDER raising change notification events when property values in high-level APIs (usually designer components) are modified.

If there is a good scenario for a user to know when a property of an object is changing, the object should raise a change notification event for the property.

However, it is unlikely to be worth the overhead to raise such events for low-level APIs such as base types or collections. For example, `List<T>` would not raise such events when a new item is added to the list and the `Count` property changes.

- ✓ CONSIDER raising change notification events when the value of a property changes via external forces.

If a property value changes via some external force (in a way other than by calling methods on the object), raise events indicate to the developer that the value is changing and has changed. A good example is the `Text` property of a text box control. When the user types text in a `TextBox`, the property value automatically changes.

*Portions © 2005, 2009 Microsoft Corporation. All rights reserved.*

*Reprinted by permission of Pearson Education, Inc. from [Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries, 2nd Edition](#) by Krzysztof Cwalina and Brad Abrams, published Oct 22, 2008 by Addison-Wesley Professional as part of the Microsoft Windows Development Series.*

## See also

- [Member Design Guidelines](#)
- [Framework Design Guidelines](#)

# Constructor Design

9/20/2022 • 3 minutes to read

There are two kinds of constructors: type constructors and instance constructors.

Type constructors are static and are run by the CLR before the type is used. Instance constructors run when an instance of a type is created.

Type constructors cannot take any parameters. Instance constructors can. Instance constructors that don't take any parameters are often called parameterless constructors.

Constructors are the most natural way to create instances of a type. Most developers will search and try to use a constructor before they consider alternative ways of creating instances (such as factory methods).

✓ CONSIDER providing simple, ideally default, constructors.

A simple constructor has a very small number of parameters, and all parameters are primitives or enums. Such simple constructors increase usability of the framework.

✓ CONSIDER using a static factory method instead of a constructor if the semantics of the desired operation do not map directly to the construction of a new instance, or if following the constructor design guidelines feels unnatural.

✓ DO use constructor parameters as shortcuts for setting main properties.

There should be no difference in semantics between using the empty constructor followed by some property sets and using a constructor with multiple arguments.

✓ DO use the same name for constructor parameters and a property if the constructor parameters are used to simply set the property.

The only difference between such parameters and the properties should be casing.

✓ DO minimal work in the constructor.

Constructors should not do much work other than capture the constructor parameters. The cost of any other processing should be delayed until required.

✓ DO throw exceptions from instance constructors, if appropriate.

✓ DO explicitly declare the public parameterless constructor in classes, if such a constructor is required.

If you don't explicitly declare any constructors on a type, many languages (such as C#) will automatically add a public parameterless constructor. (Abstract classes get a protected constructor.)

Adding a parameterized constructor to a class prevents the compiler from adding the parameterless constructor. This often causes accidental breaking changes.

✗ AVOID explicitly defining parameterless constructors on structs.

This makes array creation faster, because if the parameterless constructor is not defined, it does not have to be run on every slot in the array. Note that many compilers, including C#, don't allow structs to have parameterless constructors for this reason.

✗ AVOID calling virtual members on an object inside its constructor.

Calling a virtual member will cause the most derived override to be called, even if the constructor of the most

derived type has not been fully run yet.

## Type Constructor Guidelines

- ✓ DO make static constructors private.

A static constructor, also called a class constructor, is used to initialize a type. The CLR calls the static constructor before the first instance of the type is created or any static members on that type are called. The user has no control over when the static constructor is called. If a static constructor is not private, it can be called by code other than the CLR. Depending on the operations performed in the constructor, this can cause unexpected behavior. The C# compiler forces static constructors to be private.

- ✗ DO NOT throw exceptions from static constructors.

If an exception is thrown from a type constructor, the type is not usable in the current application domain.

- ✓ CONSIDER initializing static fields inline rather than explicitly using static constructors, because the runtime is able to optimize the performance of types that don't have an explicitly defined static constructor.

*Portions © 2005, 2009 Microsoft Corporation. All rights reserved.*

*Reprinted by permission of Pearson Education, Inc. from [Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries, 2nd Edition](#) by Krzysztof Cwalina and Brad Abrams, published Oct 22, 2008 by Addison-Wesley Professional as part of the Microsoft Windows Development Series.*

## See also

- [Member Design Guidelines](#)
- [Framework Design Guidelines](#)

# Event Design

9/20/2022 • 3 minutes to read

Events are the most commonly used form of callbacks (constructs that allow the framework to call into user code). Other callback mechanisms include members taking delegates, virtual members, and interface-based plug-ins. Data from usability studies indicate that the majority of developers are more comfortable using events than they are using the other callback mechanisms. Events are nicely integrated with Visual Studio and many languages.

It is important to note that there are two groups of events: events raised before a state of the system changes, called pre-events, and events raised after a state changes, called post-events. An example of a pre-event would be `Form.Closing`, which is raised before a form is closed. An example of a post-event would be `Form.Closed`, which is raised after a form is closed.

- ✓ DO use the term "raise" for events rather than "fire" or "trigger."
- ✓ DO use `System.EventHandler<EventArgs>` instead of manually creating new delegates to be used as event handlers.
- ✓ CONSIDER using a subclass of `EventArgs` as the event argument, unless you are absolutely sure the event will never need to carry any data to the event handling method, in which case you can use the `EventArgs` type directly.

If you ship an API using `EventArgs` directly, you will never be able to add any data to be carried with the event without breaking compatibility. If you use a subclass, even if initially completely empty, you will be able to add properties to the subclass when needed.

- ✓ DO use a protected virtual method to raise each event. This is only applicable to nonstatic events on unsealed classes, not to structs, sealed classes, or static events.

The purpose of the method is to provide a way for a derived class to handle the event using an override. Overriding is a more flexible, faster, and more natural way to handle base class events in derived classes. By convention, the name of the method should start with "On" and be followed with the name of the event.

The derived class can choose not to call the base implementation of the method in its override. Be prepared for this by not including any processing in the method that is required for the base class to work correctly.

- ✓ DO take one parameter to the protected method that raises an event.

The parameter should be named `e` and should be typed as the event argument class.

- ✗ DO NOT pass null as the sender when raising a nonstatic event.

- ✓ DO pass null as the sender when raising a static event.

- ✗ DO NOT pass null as the event data parameter when raising an event.

You should pass `EventArgs.Empty` if you don't want to pass any data to the event handling method. Developers expect this parameter not to be null.

- ✓ CONSIDER raising events that the end user can cancel. This only applies to pre-events.

Use `System.ComponentModel.CancelEventArgs` or its subclass as the event argument to allow the end user to cancel events.

## Custom Event Handler Design

There are cases in which `EventHandler<T>` cannot be used, such as when the framework needs to work with earlier versions of the CLR, which did not support Generics. In such cases, you might need to design and develop a custom event handler delegate.

- ✓ DO use a return type of void for event handlers.

An event handler can invoke multiple event handling methods, possibly on multiple objects. If event handling methods were allowed to return a value, there would be multiple return values for each event invocation.

- ✓ DO use `object` as the type of the first parameter of the event handler, and call it `sender`.
- ✓ DO use `System.EventArgs` or its subclass as the type of the second parameter of the event handler, and call it `e`.

- ✗ DO NOT have more than two parameters on event handlers.

*Portions © 2005, 2009 Microsoft Corporation. All rights reserved.*

*Reprinted by permission of Pearson Education, Inc. from [Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries, 2nd Edition](#) by Krzysztof Cwalina and Brad Abrams, published Oct 22, 2008 by Addison-Wesley Professional as part of the Microsoft Windows Development Series.*

## See also

- [Member Design Guidelines](#)
- [Framework Design Guidelines](#)

# Field Design

9/20/2022 • 2 minutes to read

The principle of encapsulation is one of the most important notions in object-oriented design. This principle states that data stored inside an object should be accessible only to that object.

A useful way to interpret the principle is to say that a type should be designed so that changes to fields of that type (name or type changes) can be made without breaking code other than for members of the type. This interpretation immediately implies that all fields must be private.

We exclude constant and static read-only fields from this strict restriction, because such fields, almost by definition, are never required to change.

✗ DO NOT provide instance fields that are public or protected.

You should provide properties for accessing fields instead of making them public or protected.

✓ DO use constant fields for constants that will never change.

The compiler burns the values of const fields directly into calling code. Therefore, const values can never be changed without the risk of breaking compatibility.

✓ DO use public static `readonly` fields for predefined object instances.

If there are predefined instances of the type, declare them as public read-only static fields of the type itself.

✗ DO NOT assign instances of mutable types to `readonly` fields.

A mutable type is a type with instances that can be modified after they are instantiated. For example, arrays, most collections, and streams are mutable types, but `System.Int32`, `System.Uri`, and `System.String` are all immutable. The read-only modifier on a reference type field prevents the instance stored in the field from being replaced, but it does not prevent the field's instance data from being modified by calling members changing the instance.

*Portions © 2005, 2009 Microsoft Corporation. All rights reserved.*

*Reprinted by permission of Pearson Education, Inc. from [Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries, 2nd Edition](#) by Krzysztof Cwalina and Brad Abrams, published Oct 22, 2008 by Addison-Wesley Professional as part of the Microsoft Windows Development Series.*

## See also

- [Member Design Guidelines](#)
- [Framework Design Guidelines](#)

# Extension Methods

9/20/2022 • 2 minutes to read

Extension methods are a language feature that allows static methods to be called using instance method call syntax. These methods must take at least one parameter, which represents the instance the method is to operate on.

The class that defines such extension methods is referred to as the "sponsor" class, and it must be declared as static. To use extension methods, one must import the namespace defining the sponsor class.

✗ AVOID frivolously defining extension methods, especially on types you don't own.

If you do own source code of a type, consider using regular instance methods instead. If you don't own, and you want to add a method, be very careful. Liberal use of extension methods has the potential of cluttering APIs of types that were not designed to have these methods.

✓ CONSIDER using extension methods in any of the following scenarios:

- To provide helper functionality relevant to every implementation of an interface, if said functionality can be written in terms of the core interface. This is because concrete implementations cannot otherwise be assigned to interfaces. For example, the `LINQ to Objects` operators are implemented as extension methods for all `IEnumerable<T>` types. Thus, any `IEnumerable<>` implementation is automatically LINQ-enabled.
- When an instance method would introduce a dependency on some type, but such a dependency would break dependency management rules. For example, a dependency from `String` to `System.Uri` is probably not desirable, and so `String.Uri()` instance method returning `System.Uri` would be the wrong design from a dependency management perspective. A static extension method `Uri.Uri(this String str)` returning `System.Uri` would be a much better design.

✗ AVOID defining extension methods on `System.Object`.

VB users will not be able to call such methods on object references using the extension method syntax. VB does not support calling such methods because, in VB, declaring a reference as Object forces all method invocations on it to be late bound (actual member called is determined at run time), while bindings to extension methods are determined at compile-time (early bound).

Note that the guideline applies to other languages where the same binding behavior is present, or where extension methods are not supported.

✗ DO NOT put extension methods in the same namespace as the extended type unless it is for adding methods to interfaces or for dependency management.

✗ AVOID defining two or more extension methods with the same signature, even if they reside in different namespaces.

✓ CONSIDER defining extension methods in the same namespace as the extended type if the type is an interface and if the extension methods are meant to be used in most or all cases.

✗ DO NOT define extension methods implementing a feature in namespaces normally associated with other features. Instead, define them in the namespace associated with the feature they belong to.

✗ AVOID generic naming of namespaces dedicated to extension methods (e.g., "Extensions"). Use a descriptive name (e.g., "Routing") instead.

*Portions © 2005, 2009 Microsoft Corporation. All rights reserved.*

*Reprinted by permission of Pearson Education, Inc. from [Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries, 2nd Edition](#) by Krzysztof Cwalina and Brad Abrams, published Oct 22, 2008 by Addison-Wesley Professional as part of the Microsoft Windows Development Series.*

## See also

- [Member Design Guidelines](#)
- [Framework Design Guidelines](#)

# Operator Overloads

9/20/2022 • 3 minutes to read

Operator overloads allow framework types to appear as if they were built-in language primitives.

Although allowed and useful in some situations, operator overloads should be used cautiously. There are many cases in which operator overloading has been abused, such as when framework designers started to use operators for operations that should be simple methods. The following guidelines should help you decide when and how to use operator overloading.

✗ AVOID defining operator overloads, except in types that should feel like primitive (built-in) types.

✓ CONSIDER defining operator overloads in a type that should feel like a primitive type.

For example, `System.String` has `operator==` and `operator!=` defined.

✓ DO define operator overloads in structs that represent numbers (such as `System.Decimal`).

✗ DO NOT be cute when defining operator overloads.

Operator overloading is useful in cases in which it is immediately obvious what the result of the operation will be. For example, it makes sense to be able to subtract one `DateTime` from another `DateTime` and get a `TimeSpan`. However, it is not appropriate to use the logical union operator to union two database queries, or to use the shift operator to write to a stream.

✗ DO NOT provide operator overloads unless at least one of the operands is of the type defining the overload.

✓ DO overload operators in a symmetric fashion.

For example, if you overload the `operator==`, you should also overload the `operator!=`. Similarly, if you overload the `operator<`, you should also overload the `operator>`, and so on.

✓ CONSIDER providing methods with friendly names that correspond to each overloaded operator.

Many languages do not support operator overloading. For this reason, it is recommended that types that overload operators include a secondary method with an appropriate domain-specific name that provides equivalent functionality.

The following table contains a list of operators and the corresponding friendly method names.

C# OPERATOR SYMBOL	METADATA NAME	FRIENDLY NAME
N/A	<code>op_Implicit</code>	<code>To&lt;TypeName&gt;/From&lt;TypeName&gt;</code>
N/A	<code>op_Explicit</code>	<code>To&lt;TypeName&gt;/From&lt;TypeName&gt;</code>
+ (binary)	<code>op_Addition</code>	<code>Add</code>
- (binary)	<code>op_Subtraction</code>	<code>Subtract</code>
* (binary)	<code>op_Multiply</code>	<code>Multiply</code>
/	<code>op_Division</code>	<code>Divide</code>

C# OPERATOR SYMBOL	METADATA NAME	FRIENDLY NAME
%	op_Modulus	Mod or Remainder
^	op_ExclusiveOr	Xor
& (binary)	op_BitwiseAnd	BitwiseAnd
	op_BitwiseOr	BitwiseOr
&&	op_LogicalAnd	And
	op_LogicalOr	Or
=	op_Assign	Assign
<<	op_LeftShift	LeftShift
>>	op_RightShift	RightShift
N/A	op_SignedRightShift	SignedRightShift
N/A	op_UnsignedRightShift	UnsignedRightShift
==	op_Equality	Equals
!=	op_Inequality	Equals
>	op_GreaterThan	CompareTo
<	op_LessThan	CompareTo
>=	op_GreaterThanOrEqual	CompareTo
<=	op_LessThanOrEqual	CompareTo
*=	op_MultiplicationAssignment	Multiply
-=	op_SubtractionAssignment	Subtract
^=	op_ExclusiveOrAssignment	Xor
<<=	op_LeftShiftAssignment	LeftShift
%=	op_ModulusAssignment	Mod
+=	op>AdditionAssignment	Add
&=	op_BitwiseAndAssignment	BitwiseAnd

C# OPERATOR SYMBOL	METADATA NAME	FRIENDLY NAME
=	op_BitwiseOrAssignment	BitwiseOr
,	op_Comma	Comma
/=	op_DivisionAssignment	Divide
--	op_Decrement	Decrement
++	op_Increment	Increment
- (unary)	op_UnaryNegation	Negate
+ (unary)	op_UnaryPlus	Plus
~	op_OnesComplement	OnesComplement

## Overloading Operator ==

Overloading `operator ==` is quite complicated. The semantics of the operator need to be compatible with several other members, such as `Object.Equals`.

## Conversion Operators

Conversion operators are unary operators that allow conversion from one type to another. The operators must be defined as static members on either the operand or the return type. There are two types of conversion operators: implicit and explicit.

✗ DO NOT provide a conversion operator if such conversion is not clearly expected by the end users.

✗ DO NOT define conversion operators outside of a type's domain.

For example, `Int32`, `Double`, and `Decimal` are all numeric types, whereas `DateTime` is not. Therefore, there should be no conversion operator to convert a `Double(long)` to a `DateTime`. A constructor is preferred in such a case.

✗ DO NOT provide an implicit conversion operator if the conversion is potentially lossy.

For example, there should not be an implicit conversion from `Double` to `Int32` because `Double` has a wider range than `Int32`. An explicit conversion operator can be provided even if the conversion is potentially lossy.

✗ DO NOT throw exceptions from implicit casts.

It is very difficult for end users to understand what is happening, because they might not be aware that a conversion is taking place.

✓ DO throw `System.InvalidCastException` if a call to a cast operator results in a lossy conversion and the contract of the operator does not allow lossy conversions.

*Portions © 2005, 2009 Microsoft Corporation. All rights reserved.*

*Reprinted by permission of Pearson Education, Inc. from [Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries, 2nd Edition](#) by Krzysztof Cwalina and Brad Abrams, published Oct 22, 2008 by Addison-Wesley Professional as part of the Microsoft Windows Development Series.*

## See also

- [Member Design Guidelines](#)

- Framework Design Guidelines

# Parameter Design

9/20/2022 • 7 minutes to read

This section provides broad guidelines on parameter design, including sections with guidelines for checking arguments. In addition, you should refer to the guidelines described in [Naming Parameters](#).

- ✓ DO use the least derived parameter type that provides the functionality required by the member.

For example, suppose you want to design a method that enumerates a collection and prints each item to the console. Such a method should take `IEnumerable` as the parameter, not `ArrayList` or `IList`, for example.

- ✗ DO NOT use reserved parameters.

If more input to a member is needed in some future version, a new overload can be added.

- ✗ DO NOT have publicly exposed methods that take pointers, arrays of pointers, or multidimensional arrays as parameters.

Pointers and multidimensional arrays are relatively difficult to use properly. In almost all cases, APIs can be redesigned to avoid taking these types as parameters.

- ✓ DO place all `out` parameters following all of the by-value and `ref` parameters (excluding parameter arrays), even if it results in an inconsistency in parameter ordering between overloads (see [Member Overloading](#)).

The `out` parameters can be seen as extra return values, and grouping them together makes the method signature easier to understand.

- ✓ DO be consistent in naming parameters when overriding members or implementing interface members.

This better communicates the relationship between the methods.

## Choosing Between Enum and Boolean Parameters

- ✓ DO use enums if a member would otherwise have two or more Boolean parameters.

- ✗ DO NOT use Booleans unless you are absolutely sure there will never be a need for more than two values.

Enums give you some room for future addition of values, but you should be aware of all the implications of adding values to enums, which are described in [Enum Design](#).

- ✓ CONSIDER using Booleans for constructor parameters that are truly two-state values and are simply used to initialize Boolean properties.

## Validating Arguments

- ✓ DO validate arguments passed to public, protected, or explicitly implemented members. Throw `System.ArgumentException`, or one of its subclasses, if the validation fails.

Note that the actual validation does not necessarily have to happen in the public or protected member itself. It could happen at a lower level in some private or internal routine. The main point is that the entire surface area that is exposed to the end users checks the arguments.

- ✓ DO throw `ArgumentNullException` if a null argument is passed and the member does not support null arguments.

- ✓ DO validate enum parameters.

Do not assume enum arguments will be in the range defined by the enum. The CLR allows casting any integer value into an enum value even if the value is not defined in the enum.

- ✗ DO NOT use `Enum.IsDefined` for enum range checks.
- ✓ DO be aware that mutable arguments might have changed after they were validated.

If the member is security sensitive, you are encouraged to make a copy and then validate and process the argument.

### Parameter Passing

From the perspective of a framework designer, there are three main groups of parameters: by-value parameters, `ref` parameters, and `out` parameters.

When an argument is passed through a by-value parameter, the member receives a copy of the actual argument passed in. If the argument is a value type, a copy of the argument is put on the stack. If the argument is a reference type, a copy of the reference is put on the stack. Most popular CLR languages, such as C#, VB.NET, and C++, default to passing parameters by value.

When an argument is passed through a `ref` parameter, the member receives a reference to the actual argument passed in. If the argument is a value type, a reference to the argument is put on the stack. If the argument is a reference type, a reference to the reference is put on the stack. `Ref` parameters can be used to allow the member to modify arguments passed by the caller.

`out` parameters are similar to `ref` parameters, with some small differences. The parameter is initially considered unassigned and cannot be read in the member body before it is assigned some value. Also, the parameter has to be assigned some value before the member returns.

- ✗ AVOID using `out` or `ref` parameters.

Using `out` or `ref` parameters requires experience with pointers, understanding how value types and reference types differ, and handling methods with multiple return values. Also, the difference between `out` and `ref` parameters is not widely understood. Framework architects designing for a general audience should not expect users to become proficient in working with `out` or `ref` parameters.

- ✗ DO NOT pass reference types by reference.

There are some limited exceptions to the rule, such as a method that can be used to swap references.

### Members with Variable Number of Parameters

Members that can take a variable number of arguments are expressed by providing an array parameter. For example, `String` provides the following method:

```
public class String {  
    public static string Format(string format, object[] parameters);  
}
```

A user can then call the `String.Format` method, as follows:

```
String.Format("File {0} not found in {1}", new object[]{filename,directory});
```

Adding the C# `params` keyword to an array parameter changes the parameter to a so-called `params` array parameter and provides a shortcut to creating a temporary array.

```
public class String {  
    public static string Format(string format, params object[] parameters);  
}
```

Doing this allows the user to call the method by passing the array elements directly in the argument list.

```
String.Format("File {0} not found in {1}",filename,directory);
```

Note that the params keyword can be added only to the last parameter in the parameter list.

✓ CONSIDER adding the params keyword to array parameters if you expect the end users to pass arrays with a small number of elements. If it's expected that lots of elements will be passed in common scenarios, users will probably not pass these elements inline anyway, and so the params keyword is not necessary.

✗ AVOID using params arrays if the caller would almost always have the input already in an array.

For example, members with byte array parameters would almost never be called by passing individual bytes. For this reason, byte array parameters in the .NET Framework do not use the params keyword.

✗ DO NOT use params arrays if the array is modified by the member taking the params array parameter.

Because of the fact that many compilers turn the arguments to the member into a temporary array at the call site, the array might be a temporary object, and therefore any modifications to the array will be lost.

✓ CONSIDER using the params keyword in a simple overload, even if a more complex overload could not use it.

Ask yourself if users would value having the params array in one overload even if it wasn't in all overloads.

✓ DO try to order parameters to make it possible to use the params keyword.

✓ CONSIDER providing special overloads and code paths for calls with a small number of arguments in extremely performance-sensitive APIs.

This makes it possible to avoid creating array objects when the API is called with a small number of arguments. Form the names of the parameters by taking a singular form of the array parameter and adding a numeric suffix.

You should only do this if you are going to special-case the entire code path, not just create an array and call the more general method.

✓ DO be aware that null could be passed as a params array argument.

You should validate that the array is not null before processing.

✗ DO NOT use the `varargs` methods, otherwise known as the ellipsis.

Some CLR languages, such as C++, support an alternative convention for passing variable parameter lists called `varargs` methods. The convention should not be used in frameworks, because it is not CLS compliant.

## Pointer Parameters

In general, pointers should not appear in the public surface area of a well-designed managed code framework. Most of the time, pointers should be encapsulated. However, in some cases pointers are required for interoperability reasons, and using pointers in such cases is appropriate.

✓ DO provide an alternative for any member that takes a pointer argument, because pointers are not CLS-compliant.

✗ AVOID doing expensive argument checking of pointer arguments.

✓ DO follow common pointer-related conventions when designing members with pointers.

For example, there is no need to pass the start index, because simple pointer arithmetic can be used to accomplish the same result.

*Reprinted by permission of Pearson Education, Inc. from Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries, 2nd Edition* by Krzysztof Cwalina and Brad Abrams, published Oct 22, 2008 by Addison-Wesley Professional as part of the Microsoft Windows Development Series.

## See also

- [Member Design Guidelines](#)
- [Framework Design Guidelines](#)

# Designing for Extensibility

9/20/2022 • 2 minutes to read

One important aspect of designing a framework is making sure the extensibility of the framework has been carefully considered. This requires that you understand the costs and benefits associated with various extensibility mechanisms. This chapter helps you decide which of the extensibility mechanisms—subclassing, events, virtual members, callbacks, and so on—can best meet the requirements of your framework.

There are many ways to allow extensibility in frameworks. They range from less powerful but less costly to very powerful but expensive. For any given extensibility requirement, you should choose the least costly extensibility mechanism that meets the requirements. Keep in mind that it's usually possible to add more extensibility later, but you can never take it away without introducing breaking changes.

## In This Section

[Unsealed Classes](#)

[Protected Members](#)

[Events and Callbacks](#)

[Virtual Members](#)

[Abstractions \(Abstract Types and Interfaces\)](#)

[Base Classes for Implementing Abstractions](#)

[Sealing](#)

*Portions © 2005, 2009 Microsoft Corporation. All rights reserved.*

*Reprinted by permission of Pearson Education, Inc. from [Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries, 2nd Edition](#) by Krzysztof Cwalina and Brad Abrams, published Oct 22, 2008 by Addison-Wesley Professional as part of the Microsoft Windows Development Series.*

## See also

- [Framework Design Guidelines](#)

# Unsealed Classes

9/20/2022 • 2 minutes to read

Sealed classes cannot be inherited from, and they prevent extensibility. In contrast, classes that can be inherited from are called unsealed classes.

✓ CONSIDER using unsealed classes with no added virtual or protected members as a great way to provide inexpensive yet much appreciated extensibility to a framework.

Developers often want to inherit from unsealed classes so as to add convenience members such as custom constructors, new methods, or method overloads. For example, `System.Messaging.MessageQueue` is unsealed and thus allows users to create custom queues that default to a particular queue path or to add custom methods that simplify the API for specific scenarios.

Classes are unsealed by default in most programming languages, and this is also the recommended default for most classes in frameworks. The extensibility afforded by unsealed types is much appreciated by framework users and quite inexpensive to provide because of relatively low test costs associated with unsealed types.

*Portions © 2005, 2009 Microsoft Corporation. All rights reserved.*

*Reprinted by permission of Pearson Education, Inc. from [Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries, 2nd Edition](#) by Krzysztof Cwalina and Brad Abrams, published Oct 22, 2008 by Addison-Wesley Professional as part of the Microsoft Windows Development Series.*

## See also

- [Framework Design Guidelines](#)
- [Designing for Extensibility](#)
- [Sealing](#)

# Protected Members

9/20/2022 • 2 minutes to read

Protected members by themselves do not provide any extensibility, but they can make extensibility through subclassing more powerful. They can be used to expose advanced customization options without unnecessarily complicating the main public interface.

Framework designers need to be careful with protected members because the name "protected" can give a false sense of security. Anyone is able to subclass an unsealed class and access protected members, and so all the same defensive coding practices used for public members apply to protected members.

- ✓ CONSIDER using protected members for advanced customization.
- ✓ DO treat protected members in unsealed classes as public for the purpose of security, documentation, and compatibility analysis.

Anyone can inherit from a class and access the protected members.

*Portions © 2005, 2009 Microsoft Corporation. All rights reserved.*

*Reprinted by permission of Pearson Education, Inc. from [Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries, 2nd Edition](#) by Krzysztof Cwalina and Brad Abrams, published Oct 22, 2008 by Addison-Wesley Professional as part of the Microsoft Windows Development Series.*

## See also

- [Framework Design Guidelines](#)
- [Designing for Extensibility](#)

# Events and Callbacks

9/20/2022 • 2 minutes to read

Callbacks are extensibility points that allow a framework to call back into user code through a delegate. These delegates are usually passed to the framework through a parameter of a method.

Events are a special case of callbacks that supports convenient and consistent syntax for supplying the delegate (an event handler). In addition, Visual Studio's statement completion and designers provide help in using event-based APIs. (See [Event Design](#).)

- ✓ CONSIDER using callbacks to allow users to provide custom code to be executed by the framework.
- ✓ CONSIDER using events to allow users to customize the behavior of a framework without the need for understanding object-oriented design.
- ✓ DO prefer events over plain callbacks, because they are more familiar to a broader range of developers and are integrated with Visual Studio statement completion.
- ✗ AVOID using callbacks in performance-sensitive APIs.

✓ DO use the new `Func<...>`, `Action<...>`, OR `Expression<...>` types instead of custom delegates, when defining APIs with callbacks.

`Func<...>` and `Action<...>` represent generic delegates. `Expression<...>` represents function definitions that can be compiled and subsequently invoked at run time but can also be serialized and passed to remote processes.

✓ DO measure and understand performance implications of using `Expression<...>`, instead of using `Func<...>` and `Action<...>` delegates.

`Expression<...>` types are in most cases logically equivalent to `Func<...>` and `Action<...>` delegates. The main difference between them is that the delegates are intended to be used in local process scenarios; expressions are intended for cases where it's beneficial and possible to evaluate the expression in a remote process or machine.

✓ DO understand that by calling a delegate, you are executing arbitrary code and that could have security, correctness, and compatibility repercussions.

*Portions © 2005, 2009 Microsoft Corporation. All rights reserved.*

*Reprinted by permission of Pearson Education, Inc. from [Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries, 2nd Edition](#) by Krzysztof Cwalina and Brad Abrams, published Oct 22, 2008 by Addison-Wesley Professional as part of the Microsoft Windows Development Series.*

## See also

- [Designing for Extensibility](#)
- [Framework Design Guidelines](#)

# Virtual Members

9/20/2022 • 2 minutes to read

Virtual members can be overridden, thus changing the behavior of the subclass. They are quite similar to callbacks in terms of the extensibility they provide, but they are better in terms of execution performance and memory consumption. Also, virtual members feel more natural in scenarios that require creating a special kind of an existing type (specialization).

Virtual members perform better than callbacks and events, but do not perform better than non-virtual methods.

The main disadvantage of virtual members is that the behavior of a virtual member can only be modified at the time of compilation. The behavior of a callback can be modified at run time.

Virtual members, like callbacks (and maybe more than callbacks), are costly to design, test, and maintain because any call to a virtual member can be overridden in unpredictable ways and can execute arbitrary code. Also, much more effort is usually required to clearly define the contract of virtual members, so the cost of designing and documenting them is higher.

✗ DO NOT make members virtual unless you have a good reason to do so and you are aware of all the costs related to designing, testing, and maintaining virtual members.

Virtual members are less forgiving in terms of changes that can be made to them without breaking compatibility. Also, they are slower than non-virtual members, mostly because calls to virtual members are not inlined.

✓ CONSIDER limiting extensibility to only what is absolutely necessary.

✓ DO prefer protected accessibility over public accessibility for virtual members. Public members should provide extensibility (if required) by calling into a protected virtual member.

The public members of a class should provide the right set of functionality for direct consumers of that class. Virtual members are designed to be overridden in subclasses, and protected accessibility is a great way to scope all virtual extensibility points to where they can be used.

*Portions © 2005, 2009 Microsoft Corporation. All rights reserved.*

*Reprinted by permission of Pearson Education, Inc. from [Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries, 2nd Edition](#) by Krzysztof Cwalina and Brad Abrams, published Oct 22, 2008 by Addison-Wesley Professional as part of the Microsoft Windows Development Series.*

## See also

- [Framework Design Guidelines](#)
- [Designing for Extensibility](#)

# Abstractions (Abstract Types and Interfaces)

9/20/2022 • 2 minutes to read

An abstraction is a type that describes a contract but does not provide a full implementation of the contract. Abstractions are usually implemented as abstract classes or interfaces, and they come with a well-defined set of reference documentation describing the required semantics of the types implementing the contract. Some of the most important abstractions in the .NET Framework include [Stream](#), [IEnumerable<T>](#), and [Object](#).

You can extend frameworks by implementing a concrete type that supports the contract of an abstraction and using this concrete type with framework APIs consuming (operating on) the abstraction.

A meaningful and useful abstraction that is able to withstand the test of time is very difficult to design. The main difficulty is getting the right set of members, no more and no fewer. If an abstraction has too many members, it becomes difficult or even impossible to implement. If it has too few members for the promised functionality, it becomes useless in many interesting scenarios.

Too many abstractions in a framework also negatively affect usability of the framework. It is often quite difficult to understand an abstraction without understanding how it fits into the larger picture of the concrete implementations and the APIs operating on the abstraction. Also, names of abstractions and their members are necessarily abstract, which often makes them cryptic and unapproachable without first understanding the broader context of their usage.

However, abstractions provide extremely powerful extensibility that the other extensibility mechanisms cannot often match. They are at the core of many architectural patterns, such as plug-ins, inversion of control (IoC), pipelines, and so on. They are also extremely important for testability of frameworks. Good abstractions make it possible to stub out heavy dependencies for the purpose of unit testing. In summary, abstractions are responsible for the sought-after richness of the modern object-oriented frameworks.

- ✗ DO NOT provide abstractions unless they are tested by developing several concrete implementations and APIs consuming the abstractions.
- ✓ DO choose carefully between an abstract class and an interface when designing an abstraction.
- ✓ CONSIDER providing reference tests for concrete implementations of abstractions. Such tests should allow users to test whether their implementations correctly implement the contract.

*Portions © 2005, 2009 Microsoft Corporation. All rights reserved.*

*Reprinted by permission of Pearson Education, Inc. from [Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries, 2nd Edition](#) by Krzysztof Cwalina and Brad Abrams, published Oct 22, 2008 by Addison-Wesley Professional as part of the Microsoft Windows Development Series.*

## See also

- [Framework Design Guidelines](#)
- [Designing for Extensibility](#)

# Base Classes for Implementing Abstractions

9/20/2022 • 2 minutes to read

Strictly speaking, a class becomes a base class when another class is derived from it. For the purpose of this section, however, a base class is a class designed mainly to provide a common abstraction or for other classes to reuse some default implementation through inheritance. Base classes usually sit in the middle of inheritance hierarchies, between an abstraction at the root of a hierarchy and several custom implementations at the bottom.

They serve as implementation helpers for implementing abstractions. For example, one of the Framework's abstractions for ordered collections of items is the `IList<T>` interface. Implementing `IList<T>` is not trivial, and therefore the Framework provides several base classes, such as `Collection<T>` and `KeyedCollection< TKey,TItem >`, which serve as helpers for implementing custom collections.

Base classes are usually not suited to serve as abstractions by themselves, because they tend to contain too much implementation. For example, the `Collection<T>` base class contains lots of implementation related to the fact that it implements the nongeneric `IList` interface (to integrate better with nongeneric collections) and to the fact that it is a collection of items stored in memory in one of its fields.

As previously discussed, base classes can provide invaluable help for users who need to implement abstractions, but at the same time they can be a significant liability. They add surface area and increase the depth of inheritance hierarchies and so conceptually complicate the framework. Therefore, base classes should be used only if they provide significant value to the users of the framework. They should be avoided if they provide value only to the implementers of the framework, in which case delegation to an internal implementation instead of inheritance from a base class should be strongly considered.

- ✓ CONSIDER making base classes abstract even if they don't contain any abstract members. This clearly communicates to the users that the class is designed solely to be inherited from.
- ✓ CONSIDER placing base classes in a separate namespace from the mainline scenario types. By definition, base classes are intended for advanced extensibility scenarios and therefore are not interesting to the majority of users.
- ✗ AVOID naming base classes with a "Base" suffix if the class is intended for use in public APIs.

*Portions © 2005, 2009 Microsoft Corporation. All rights reserved.*

*Reprinted by permission of Pearson Education, Inc. from [Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries, 2nd Edition](#) by Krzysztof Cwalina and Brad Abrams, published Oct 22, 2008 by Addison-Wesley Professional as part of the Microsoft Windows Development Series.*

## See also

- [Framework Design Guidelines](#)
- [Designing for Extensibility](#)

# Sealing

9/20/2022 • 2 minutes to read

One of the features of object-oriented frameworks is that developers can extend and customize them in ways unanticipated by the framework designers. This is both the power and danger of extensible design. When you design your framework, it is, therefore, very important to carefully design for extensibility when it is desired, and to limit extensibility when it is dangerous.

A powerful mechanism that prevents extensibility is sealing. You can seal either the class or individual members. Sealing a class prevents users from inheriting from the class. Sealing a member prevents users from overriding a particular member.

✗ DO NOT seal classes without having a good reason to do so.

Sealing a class because you cannot think of an extensibility scenario is not a good reason. Framework users like to inherit from classes for various non-obvious reasons, like adding convenience members. See [Unsealed Classes](#) for examples of non-obvious reasons users want to inherit from a type.

Good reasons for sealing a class include the following:

- The class is a static class. See [Static Class Design](#).
- The class stores security-sensitive secrets in inherited protected members.
- The class inherits many virtual members and the cost of sealing them individually would outweigh the benefits of leaving the class unsealed.
- The class is an attribute that requires very fast runtime look-up. Sealed attributes have slightly higher performance levels than unsealed ones. See [Attributes](#).

✗ DO NOT declare protected or virtual members on sealed types.

By definition, sealed types cannot be inherited from. This means that protected members on sealed types cannot be called, and virtual methods on sealed types cannot be overridden.

✓ CONSIDER sealing members that you override.

Problems that can result from introducing virtual members (discussed in [Virtual Members](#)) apply to overrides as well, although to a slightly lesser degree. Sealing an override shields you from these problems starting from that point in the inheritance hierarchy.

*Portions © 2005, 2009 Microsoft Corporation. All rights reserved.*

*Reprinted by permission of Pearson Education, Inc. from [Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries, 2nd Edition](#) by Krzysztof Cwalina and Brad Abrams, published Oct 22, 2008 by Addison-Wesley Professional as part of the Microsoft Windows Development Series.*

## See also

- [Framework Design Guidelines](#)
- [Designing for Extensibility](#)
- [Unsealed Classes](#)

# Design Guidelines for Exceptions

9/20/2022 • 2 minutes to read

Exception handling has many advantages over return-value-based error reporting. Good framework design helps the application developer realize the benefits of exceptions. This section discusses the benefits of exceptions and presents guidelines for using them effectively.

## In This Section

[Exception Throwing](#)

[Using Standard Exception Types](#)

[Exceptions and Performance](#)

*Portions © 2005, 2009 Microsoft Corporation. All rights reserved.*

*Reprinted by permission of Pearson Education, Inc. from [Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries, 2nd Edition](#) by Krzysztof Cwalina and Brad Abrams, published Oct 22, 2008 by Addison-Wesley Professional as part of the Microsoft Windows Development Series.*

## See also

- [Framework Design Guidelines](#)

# Exception Throwing

9/20/2022 • 2 minutes to read

Exception-throwing guidelines described in this section require a good definition of the meaning of execution failure. Execution failure occurs whenever a member cannot do what it was designed to do (what the member name implies). For example, if the `OpenFile` method cannot return an opened file handle to the caller, it would be considered an execution failure.

Most developers have become comfortable with using exceptions for usage errors such as division by zero or null references. In the Framework, exceptions are used for all error conditions, including execution errors.

- ✗ DO NOT return error codes.

Exceptions are the primary means of reporting errors in frameworks.

- ✓ DO report execution failures by throwing exceptions.
- ✓ CONSIDER terminating the process by calling `System.Environment.FailFast` (.NET Framework 2.0 feature) instead of throwing an exception if your code encounters a situation where it is unsafe for further execution.

- ✗ DO NOT use exceptions for the normal flow of control, if possible.

Except for system failures and operations with potential race conditions, framework designers should design APIs so users can write code that does not throw exceptions. For example, you can provide a way to check preconditions before calling a member so users can write code that does not throw exceptions.

The member used to check preconditions of another member is often referred to as a tester, and the member that actually does the work is called a doer.

There are cases when the Tester-Doer Pattern can have an unacceptable performance overhead. In such cases, the so-called Try-Parse Pattern should be considered (see [Exceptions and Performance](#) for more information).

- ✓ CONSIDER the performance implications of throwing exceptions. Throw rates above 100 per second are likely to noticeably impact the performance of most applications.
- ✓ DO document all exceptions thrown by publicly callable members because of a violation of the member contract (rather than a system failure) and treat them as part of your contract.

Exceptions that are a part of the contract should not change from one version to the next (i.e., exception type should not change, and new exceptions should not be added).

- ✗ DO NOT have public members that can either throw or not based on some option.
- ✗ DO NOT have public members that return exceptions as the return value or an `out` parameter.

Returning exceptions from public APIs instead of throwing them defeats many of the benefits of exception-based error reporting.

- ✓ CONSIDER using exception builder methods.

It is common to throw the same exception from different places. To avoid code bloat, use helper methods that create exceptions and initialize their properties.

Also, members that throw exceptions are not getting inlined. Moving the throw statement inside the builder might allow the member to be inlined.

- ✗ DO NOT throw exceptions from exception filter blocks.

When an exception filter raises an exception, the exception is caught by the CLR, and the filter returns false. This behavior is indistinguishable from the filter executing and returning false explicitly and is therefore very difficult to debug.

✖ AVOID explicitly throwing exceptions from finally blocks. Implicitly thrown exceptions resulting from calling methods that throw are acceptable.

*Portions © 2005, 2009 Microsoft Corporation. All rights reserved.*

*Reprinted by permission of Pearson Education, Inc. from [Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries, 2nd Edition](#) by Krzysztof Cwalina and Brad Abrams, published Oct 22, 2008 by Addison-Wesley Professional as part of the Microsoft Windows Development Series.*

## See also

- [Framework Design Guidelines](#)
- [Design Guidelines for Exceptions](#)

# Using Standard Exception Types

9/20/2022 • 2 minutes to read

This section describes the standard exceptions provided by the Framework and the details of their usage. The list is by no means exhaustive. Please refer to the .NET Framework reference documentation for usage of other Framework exception types.

## Exception and SystemException

- ✗ DO NOT throw `System.Exception` or `System.SystemException`.
- ✗ DO NOT catch `System.Exception` or `System.SystemException` in framework code, unless you intend to rethrow.
- ✗ AVOID catching `System.Exception` or `System.SystemException`, except in top-level exception handlers.

## ApplicationException

- ✗ DO NOT throw or derive from `ApplicationException`.

## InvalidOperationException

- ✓ DO throw an `InvalidOperationException` if the object is in an inappropriate state.

## ArgumentException, ArgumentNullException, and ArgumentOutOfRangeException

- ✓ DO throw `ArgumentException` or one of its subtypes if bad arguments are passed to a member. Prefer the most derived exception type, if applicable.
- ✓ DO set the `ParamName` property when throwing one of the subclasses of `ArgumentException`.

This property represents the name of the parameter that caused the exception to be thrown. Note that the property can be set using one of the constructor overloads.

- ✓ DO use `value` for the name of the implicit value parameter of property setters.

## NullReferenceException, IndexOutOfRangeException, and AccessViolationException

- ✗ DO NOT allow publicly callable APIs to explicitly or implicitly throw `NullReferenceException`, `AccessViolationException`, or `IndexOutOfRangeException`. These exceptions are reserved and thrown by the execution engine and in most cases indicate a bug.

Do argument checking to avoid throwing these exceptions. Throwing these exceptions exposes implementation details of your method that might change over time.

## StackOverflowException

- ✗ DO NOT explicitly throw `StackOverflowException`. The exception should be explicitly thrown only by the CLR.
- ✗ DO NOT catch `StackOverflowException`.

It is almost impossible to write managed code that remains consistent in the presence of arbitrary stack overflows. The unmanaged parts of the CLR remain consistent by using probes to move stack overflows to well-defined places rather than by backing out from arbitrary stack overflows.

## OutOfMemoryException

✗ DO NOT explicitly throw [OutOfMemoryException](#). This exception is to be thrown only by the CLR infrastructure.

## ComException, SEHException, and ExecutionEngineException

✗ DO NOT explicitly throw [COMException](#), [ExecutionEngineException](#), and [SEHException](#). These exceptions are to be thrown only by the CLR infrastructure.

*Portions © 2005, 2009 Microsoft Corporation. All rights reserved.*

*Reprinted by permission of Pearson Education, Inc. from [Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries, 2nd Edition](#) by Krzysztof Cwalina and Brad Abrams, published Oct 22, 2008 by Addison-Wesley Professional as part of the Microsoft Windows Development Series.*

## See also

- [Framework Design Guidelines](#)
- [Design Guidelines for Exceptions](#)

# Exceptions and Performance

9/20/2022 • 2 minutes to read

One common concern related to exceptions is that if exceptions are used for code that routinely fails, the performance of the implementation will be unacceptable. This is a valid concern. When a member throws an exception, its performance can be orders of magnitude slower. However, it is possible to achieve good performance while strictly adhering to the exception guidelines that disallow using error codes. Two patterns described in this section suggest ways to do this.

✗ DO NOT use error codes because of concerns that exceptions might affect performance negatively.

To improve performance, it is possible to use either the Tester-Doer Pattern or the Try-Parse Pattern, described in the next two sections.

## Tester-Doer Pattern

Sometimes performance of an exception-throwing member can be improved by breaking the member into two. Let's look at the `Add` method of the `ICollection<T>` interface.

```
ICollection<int> numbers = ...  
numbers.Add(1);
```

The method `Add` throws if the collection is read-only. This can be a performance problem in scenarios where the method call is expected to fail often. One of the ways to mitigate the problem is to test whether the collection is writable before trying to add a value.

```
ICollection<int> numbers = ...  
...  
if (!numbers.IsReadOnly)  
{  
    numbers.Add(1);  
}
```

The member used to test a condition, which in our example is the property `IsReadOnly`, is referred to as the tester. The member used to perform a potentially throwing operation, the `Add` method in our example, is referred to as the doer.

✓ CONSIDER the Tester-Doer Pattern for members that might throw exceptions in common scenarios to avoid performance problems related to exceptions.

## Try-Parse Pattern

For extremely performance-sensitive APIs, an even faster pattern than the Tester-Doer Pattern described in the previous section should be used. The pattern calls for adjusting the member name to make a well-defined test case a part of the member semantics. For example, `DateTime` defines a `Parse` method that throws an exception if parsing of a string fails. It also defines a corresponding `TryParse` method that attempts to parse, but returns false if parsing is unsuccessful and returns the result of a successful parsing using an `out` parameter.

```
public struct DateTime
{
    public static DateTime Parse(string dateTime)
    {
        ...
    }
    public static bool TryParse(string dateTime, out DateTime result)
    {
        ...
    }
}
```

When using this pattern, it is important to define the try functionality in strict terms. If the member fails for any reason other than the well-defined try, the member must still throw a corresponding exception.

- ✓ CONSIDER the Try-Parse Pattern for members that might throw exceptions in common scenarios to avoid performance problems related to exceptions.
- ✓ DO use the prefix "Try" and Boolean return type for methods implementing this pattern.
- ✓ DO provide an exception-throwing member for each member using the Try-Parse Pattern.

*Portions © 2005, 2009 Microsoft Corporation. All rights reserved.*

*Reprinted by permission of Pearson Education, Inc. from [Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries, 2nd Edition](#) by Krzysztof Cwalina and Brad Abrams, published Oct 22, 2008 by Addison-Wesley Professional as part of the Microsoft Windows Development Series.*

## See also

- [Framework Design Guidelines](#)
- [Design Guidelines for Exceptions](#)

# Usage guidelines

9/20/2022 • 2 minutes to read

This section contains guidelines for using common types in publicly accessible APIs. It deals with direct usage of built-in Framework types (e.g., serialization attributes) and overloading common operators.

The [System.IDisposable](#) interface is not covered in this section, but is discussed in the [Dispose Pattern](#) section.

## NOTE

For guidelines and additional information about other common, built-in .NET Framework types, see the reference topics for the following: [System.DateTime](#), [System DateTimeOffset](#), [System.ICloneable](#), [System.IComparable<T>](#), [System.IEquatable<T>](#), [System.Nullable<T>](#), [System.Object](#), [System.Uri](#).

## In this section

- [Arrays](#)
- [Attributes](#)
- [Collections](#)
- [Serialization](#)
- [System.Xml Usage](#)
- [Equality Operators\](#)

*Portions © 2005, 2009 Microsoft Corporation. All rights reserved.*

*Reprinted by permission of Pearson Education, Inc. from [Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries, 2nd Edition](#) by Krzysztof Cwalina and Brad Abrams, published Oct 22, 2008 by Addison-Wesley Professional as part of the Microsoft Windows Development Series.*

## See also

- [Framework Design Guidelines](#)

# Arrays (.NET Framework design guidelines)

9/20/2022 • 2 minutes to read

✓ DO prefer using collections over arrays in public APIs. The [Collections](#) section provides details about how to choose between collections and arrays.

✗ DO NOT use read-only array fields. The field itself is read-only and can't be changed, but elements in the array can be changed.

✓ CONSIDER using jagged arrays instead of multidimensional arrays.

A jagged array is an array with elements that are also arrays. The arrays that make up the elements can be of different sizes, leading to less wasted space for some sets of data (e.g., sparse matrix) compared to multidimensional arrays. Furthermore, the CLR optimizes index operations on jagged arrays, so they might exhibit better runtime performance in some scenarios.

*Portions © 2005, 2009 Microsoft Corporation. All rights reserved.*

*Reprinted by permission of Pearson Education, Inc. from [Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries, 2nd Edition](#) by Krzysztof Cwalina and Brad Abrams, published Oct 22, 2008 by Addison-Wesley Professional as part of the Microsoft Windows Development Series.*

## See also

- [Array](#)
- [Framework Design Guidelines](#)
- [Usage Guidelines](#)

# Attributes (.NET Framework design guidelines)

9/20/2022 • 2 minutes to read

[System.Attribute](#) is a base class used to define custom attributes.

Attributes are annotations that can be added to programming elements such as assemblies, types, members, and parameters. They are stored in the metadata of the assembly and can be accessed at run time using the reflection APIs. For example, the Framework defines the [ObsoleteAttribute](#), which can be applied to a type or a member to indicate that the type or member has been deprecated.

Attributes can have one or more properties that carry additional data related to the attribute. For example, [ObsoleteAttribute](#) could carry additional information about the release in which a type or a member got deprecated and the description of the new APIs replacing the obsolete API.

Some properties of an attribute must be specified when the attribute is applied. These are referred to as the required properties or required arguments, because they are represented as positional constructor parameters. For example, the [ConditionString](#) property of the [ConditionalAttribute](#) is a required property.

Properties that do not necessarily have to be specified when the attribute is applied are called optional properties (or optional arguments). They are represented by settable properties. Compilers provide special syntax to set these properties when an attribute is applied. For example, the [AttributeUsageAttribute.Inherited](#) property represents an optional argument.

- ✓ DO name custom attribute classes with the suffix "Attribute."
- ✓ DO apply the [AttributeUsageAttribute](#) to custom attributes.
- ✓ DO provide settable properties for optional arguments.
- ✓ DO provide get-only properties for required arguments.
- ✓ DO provide constructor parameters to initialize properties corresponding to required arguments. Each parameter should have the same name (although with different casing) as the corresponding property.
- ✗ AVOID providing constructor parameters to initialize properties corresponding to the optional arguments.

In other words, do not have properties that can be set with both a constructor and a setter. This guideline makes very explicit which arguments are optional and which are required, and avoids having two ways of doing the same thing.

- ✗ AVOID overloading custom attribute constructors.

Having only one constructor clearly communicates to the user which arguments are required and which are optional.

- ✓ DO seal custom attribute classes, if possible. This makes the look-up for the attribute faster.

*Portions © 2005, 2009 Microsoft Corporation. All rights reserved.*

*Reprinted by permission of Pearson Education, Inc. from [Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries, 2nd Edition](#) by Krzysztof Cwalina and Brad Abrams, published Oct 22, 2008 by Addison-Wesley Professional as part of the Microsoft Windows Development Series.*

## See also

- [Framework Design Guidelines](#)

- Usage Guidelines

# Guidelines for Collections

9/20/2022 • 7 minutes to read

Any type designed specifically to manipulate a group of objects having some common characteristic can be considered a collection. It is almost always appropriate for such types to implement `IEnumerable` or `IEnumerable<T>`, so in this section we only consider types implementing one or both of those interfaces to be collections.

✗ DO NOT use weakly typed collections in public APIs.

The type of all return values and parameters representing collection items should be the exact item type, not any of its base types (this applies only to public members of the collection).

✗ DO NOT use `ArrayList` or `List<T>` in public APIs.

These types are data structures designed to be used in internal implementation, not in public APIs. `List<T>` is optimized for performance and power at the cost of cleanliness of the APIs and flexibility. For example, if you return `List<T>`, you will not ever be able to receive notifications when client code modifies the collection. Also, `List<T>` exposes many members, such as `BinarySearch`, that are not useful or applicable in many scenarios. The following two sections describe types (abstractions) designed specifically for use in public APIs.

✗ DO NOT use `Hashtable` or `Dictionary< TKey, TValue >` in public APIs.

These types are data structures designed to be used in internal implementation. Public APIs should use `IDictionary`, `IDictionary < TKey, TValue >`, or a custom type implementing one or both of the interfaces.

✗ DO NOT use `IEnumerator<T>`, `IEnumerator`, or any other type that implements either of these interfaces, except as the return type of a `GetEnumerator` method.

Types returning enumerators from methods other than `GetEnumerator` cannot be used with the `foreach` statement.

✗ DO NOT implement both `IEnumerator<T>` and `IEnumerable<T>` on the same type. The same applies to the nongeneric interfaces `IEnumerator` and `IEnumerable`.

## Collection Parameters

✓ DO use the least-specialized type possible as a parameter type. Most members taking collections as parameters use the `IEnumerable<T>` interface.

✗ AVOID using `ICollection<T>` or `ICollection` as a parameter just to access the `Count` property.

Instead, consider using `IEnumerable<T>` or `IEnumerable` and dynamically checking whether the object implements `ICollection<T>` or `ICollection`.

## Collection Properties and Return Values

✗ DO NOT provide settable collection properties.

Users can replace the contents of the collection by clearing the collection first and then adding the new contents. If replacing the whole collection is a common scenario, consider providing the `AddRange` method on the collection.

✓ DO use `Collection<T>` or a subclass of `Collection<T>` for properties or return values representing

read/write collections.

If `Collection<T>` does not meet some requirement (e.g., the collection must not implement `IList`), use a custom collection by implementing `IEnumerable<T>`, `ICollection<T>`, or `IList<T>`.

✓ DO use `ReadOnlyCollection<T>`, a subclass of `ReadOnlyCollection<T>`, or in rare cases `IEnumerable<T>` for properties or return values representing read-only collections.

In general, prefer `ReadOnlyCollection<T>`. If it does not meet some requirement (e.g., the collection must not implement `IList`), use a custom collection by implementing `IEnumerable<T>`, `ICollection<T>`, or `IList<T>`. If you do implement a custom read-only collection, implement `ICollection<T>.IsReadOnly` to return `true`.

In cases where you are sure that the only scenario you will ever want to support is forward-only iteration, you can simply use `IEnumerable<T>`.

✓ CONSIDER using subclasses of generic base collections instead of using the collections directly.

This allows for a better name and for adding helper members that are not present on the base collection types. This is especially applicable to high-level APIs.

✓ CONSIDER returning a subclass of `Collection<T>` or `ReadOnlyCollection<T>` from very commonly used methods and properties.

This will make it possible for you to add helper methods or change the collection implementation in the future.

✓ CONSIDER using a keyed collection if the items stored in the collection have unique keys (names, IDs, etc.). Keyed collections are collections that can be indexed by both an integer and a key and are usually implemented by inheriting from `KeyedCollection< TKey, TItem >`.

Keyed collections usually have larger memory footprints and should not be used if the memory overhead outweighs the benefits of having the keys.

✗ DO NOT return null values from collection properties or from methods returning collections. Return an empty collection or an empty array instead.

The general rule is that null and empty (0 item) collections or arrays should be treated the same.

### Snapshots Versus Live Collections

Collections representing a state at some point in time are called snapshot collections. For example, a collection containing rows returned from a database query would be a snapshot. Collections that always represent the current state are called live collections. For example, a collection of `ComboBox` items is a live collection.

✗ DO NOT return snapshot collections from properties. Properties should return live collections.

Property getters should be very lightweight operations. Returning a snapshot requires creating a copy of an internal collection in an O(n) operation.

✓ DO use either a snapshot collection or a live `IEnumerable<T>` (or its subtype) to represent collections that are volatile (i.e., that can change without explicitly modifying the collection).

In general, all collections representing a shared resource (e.g., files in a directory) are volatile. Such collections are very difficult or impossible to implement as live collections unless the implementation is simply a forward-only enumerator.

## Choosing Between Arrays and Collections

✓ DO prefer collections over arrays.

Collections provide more control over contents, can evolve over time, and are more usable. In addition, using

arrays for read-only scenarios is discouraged because the cost of cloning the array is prohibitive. Usability studies have shown that some developers feel more comfortable using collection-based APIs.

However, if you are developing low-level APIs, it might be better to use arrays for read-write scenarios. Arrays have a smaller memory footprint, which helps reduce the working set, and access to elements in an array is faster because it is optimized by the runtime.

- ✓ CONSIDER using arrays in low-level APIs to minimize memory consumption and maximize performance.
- ✓ DO use byte arrays instead of collections of bytes.
- ✗ DO NOT use arrays for properties if the property would have to return a new array (e.g., a copy of an internal array) every time the property getter is called.

## Implementing Custom Collections

- ✓ CONSIDER inheriting from `Collection<T>`, `ReadOnlyCollection<T>`, or `KeyedCollection< TKey, TItem >` when designing new collections.
- ✓ DO implement `IEnumerable<T>` when designing new collections. Consider implementing `ICollection<T>` or even `IList<T>` where it makes sense.

When implementing such custom collection, follow the API pattern established by `Collection<T>` and `ReadOnlyCollection<T>` as closely as possible. That is, implement the same members explicitly, name the parameters like these two collections name them, and so on.

- ✓ CONSIDER implementing nongeneric collection interfaces (`IList` and `ICollection`) if the collection will often be passed to APIs taking these interfaces as input.
- ✗ AVOID implementing collection interfaces on types with complex APIs unrelated to the concept of a collection.
- ✗ DO NOT inherit from nongeneric base collections such as `CollectionBase`. Use `Collection<T>`, `ReadOnlyCollection<T>`, and `KeyedCollection< TKey, TItem >` instead.

### Naming Custom Collections

Collections (types that implement `IEnumerable`) are created mainly for two reasons: (1) to create a new data structure with structure-specific operations and often different performance characteristics than existing data structures (e.g., `List<T>`, `LinkedList<T>`, `Stack<T>`), and (2) to create a specialized collection for holding a specific set of items (e.g., `StringCollection`). Data structures are most often used in the internal implementation of applications and libraries. Specialized collections are mainly to be exposed in APIs (as property and parameter types).

- ✓ DO use the "Dictionary" suffix in names of abstractions implementing `IDictionary` or `IDictionary< TKey, TValue >`.
- ✓ DO use the "Collection" suffix in names of types implementing `IEnumerable` (or any of its descendants) and representing a list of items.
- ✓ DO use the appropriate data structure name for custom data structures.
- ✗ AVOID using any suffixes implying particular implementation, such as "LinkedList" or "Hashtable," in names of collection abstractions.
- ✓ CONSIDER prefixing collection names with the name of the item type. For example, a collection storing items of type `Address` (implementing `IEnumerable<Address>`) should be named `AddressCollection`. If the item type is an interface, the "I" prefix of the item type can be omitted. Thus, a collection of `IDisposable` items can be called `DisposableCollection`.

- ✓ CONSIDER using the "ReadOnly" prefix in names of read-only collections if a corresponding writeable collection might be added or already exists in the framework.

For example, a read-only collection of strings should be called `ReadOnlyStringCollection`.

*Portions © 2005, 2009 Microsoft Corporation. All rights reserved.*

*Reprinted by permission of Pearson Education, Inc. from [Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries, 2nd Edition](#) by Krzysztof Cwalina and Brad Abrams, published Oct 22, 2008 by Addison-Wesley Professional as part of the Microsoft Windows Development Series.*

## See also

- [Framework Design Guidelines](#)
- [Usage Guidelines](#)

# Serialization

9/20/2022 • 5 minutes to read

Serialization is the process of converting an object into a format that can be readily persisted or transported. For example, you can serialize an object, transport it over the Internet using HTTP, and deserialize it at the destination machine.

The .NET Framework offers three main serialization technologies optimized for various serialization scenarios. The following table lists these technologies and the main Framework types related to these technologies.

TECHNOLOGY NAME	MAIN TYPES	SCENARIOS
Data Contract Serialization	<a href="#">DataContractAttribute</a> <a href="#">DataMemberAttribute</a> <a href="#">DataContractSerializer</a> <a href="#">NetDataContractSerializer</a> <a href="#">DataContractJsonSerializer</a> <a href="#">ISerializable</a>	General persistence Web Services JSON
XML Serialization	<a href="#">XmlSerializer</a>	XML format with full control over the shape of the XML
Runtime Serialization (Binary and SOAP)	<a href="#">SerializableAttribute</a> <a href="#">ISerializable</a> <a href="#">BinaryFormatter</a> <a href="#">SoapFormatter</a>	.NET Remoting

- ✓ DO think about serialization when you design new types.

## Choosing the Right Serialization Technology to Support

- ✓ CONSIDER supporting Data Contract Serialization if instances of your type might need to be persisted or used in Web Services.
- ✓ CONSIDER supporting the XML Serialization instead of or in addition to Data Contract Serialization if you need more control over the XML format that is produced when the type is serialized.

This may be necessary in some interoperability scenarios where you need to use an XML construct that is not supported by Data Contract Serialization, for example, to produce XML attributes.

- ✓ CONSIDER supporting the Runtime Serialization if instances of your type need to travel across .NET Remoting boundaries.

- ✗ AVOID supporting Runtime Serialization or XML Serialization just for general persistence reasons. Prefer Data Contract Serialization instead.

## Supporting Data Contract Serialization

Types can support Data Contract Serialization by applying the [DataContractAttribute](#) to the type and the [DataMemberAttribute](#) to the members (fields and properties) of the type.

- ✓ CONSIDER marking data members of your type public if the type can be used in partial trust.

In full trust, Data Contract serializers can serialize and deserialize nonpublic types and members, but only public

members can be serialized and deserialized in partial trust.

✓ DO implement a getter and setter on all properties that have [DataMemberAttribute](#). Data Contract serializers require both the getter and the setter for the type to be considered serializable. (In .NET Framework 3.5 SP1, some collection properties can be get-only.) If the type won't be used in partial trust, one or both of the property accessors can be nonpublic.

✓ CONSIDER using the serialization callbacks for initialization of deserialized instances.

Constructors are not called when objects are deserialized. (There are exceptions to the rule. Constructors of collections marked with [CollectionDataContractAttribute](#) are called during deserialization.) Therefore, any logic that executes during normal construction needs to be implemented as one of the serialization callbacks.

[OnDeserializedAttribute](#) is the most commonly used callback attribute. The other attributes in the family are [OnDeserializingAttribute](#), [OnSerializingAttribute](#), and [OnSerializedAttribute](#). They can be used to mark callbacks that get executed before deserialization, before serialization, and finally, after serialization, respectively.

✓ CONSIDER using the [KnownTypeAttribute](#) to indicate concrete types that should be used when deserializing a complex object graph.

✓ DO consider backward and forward compatibility when creating or changing serializable types.

Keep in mind that serialized streams of future versions of your type can be deserialized into the current version of the type, and vice versa.

Make sure you understand that data members, even private and internal, cannot change their names, types, or even their order in future versions of the type unless special care is taken to preserve the contract using explicit parameters to the data contract attributes.

Test compatibility of serialization when making changes to serializable types. Try deserializing the new version into an old version, and vice versa.

✓ CONSIDER implementing [IExtensibleDataObject](#) to allow round-tripping between different versions of the type.

The interface allows the serializer to ensure that no data is lost during round-tripping. The [IExtensibleDataObject.ExtensionData](#) property is used to store any data from the future version of the type that is unknown to the current version, and so it cannot store it in its data members. When the current version is subsequently serialized and deserialized into a future version, the additional data will be available in the serialized stream.

## Supporting XML Serialization

Data Contract Serialization is the main (default) serialization technology in the .NET Framework, but there are serialization scenarios that Data Contract Serialization does not support. For example, it does not give you full control over the shape of XML produced or consumed by the serializer. If such fine control is required, XML Serialization has to be used, and you need to design your types to support this serialization technology.

✗ AVOID designing your types specifically for XML Serialization, unless you have a very strong reason to control the shape of the XML produced. This serialization technology has been superseded by the Data Contract Serialization discussed in the previous section.

✓ CONSIDER implementing the [IXmlSerializable](#) interface if you want even more control over the shape of the serialized XML than what's offered by applying the XML Serialization attributes. Two methods of the interface, [ReadXml](#) and [WriteXml](#), allow you to fully control the serialized XML stream. You can also control the XML schema that gets generated for the type by applying the [XmlAttributeProviderAttribute](#).

# Supporting Runtime Serialization

Runtime Serialization is a technology used by .NET Remoting. If you think your types will be transported using .NET Remoting, you need to make sure they support Runtime Serialization.

The basic support for Runtime Serialization can be provided by applying the [SerializableAttribute](#), and more advanced scenarios involve implementing a simple Runtime Serializable Pattern (implement [ISerializable](#) and provide serialization constructor).

✓ CONSIDER supporting Runtime Serialization if your types will be used with .NET Remoting. For example, the [System.AddIn](#) namespace uses .NET Remoting, and so all types exchanged between [System.AddIn](#) add-ins need to support Runtime Serialization.

✓ CONSIDER implementing the Runtime Serializable Pattern if you want complete control over the serialization process. For example, if you want to transform data as it gets serialized or deserialized.

The pattern is very simple. All you need to do is implement the [ISerializable](#) interface and provide a special constructor that is used when the object is deserialized.

✓ DO make the serialization constructor protected and provide two parameters typed and named exactly as shown in the sample here.

```
[Serializable]
public class Person : ISerializable
{
    protected Person(SerializationInfo info, StreamingContext context)
    {
        // ...
    }
}
```

✓ DO implement the [ISerializable](#) members explicitly.

✓ DO apply a link demand to [ISerializable.GetObjectData](#) implementation. This ensures that only fully trusted code and the Runtime Serializer have access to the member.

*Portions © 2005, 2009 Microsoft Corporation. All rights reserved.*

*Reprinted by permission of Pearson Education, Inc. from [Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries, 2nd Edition](#) by Krzysztof Cwalina and Brad Abrams, published Oct 22, 2008 by Addison-Wesley Professional as part of the Microsoft Windows Development Series.*

## See also

- [Framework Design Guidelines](#)
- [Usage Guidelines](#)

# System.Xml Usage

9/20/2022 • 2 minutes to read

This section talks about usage of several types residing in [System.Xml](#) namespaces that can be used to represent XML data.

✗ DO NOT use `XmlNode` or  `XmlDocument` to represent XML data. Favor using instances of `IXPathNavigable`, `XmlReader`, `XmlWriter`, or subtypes of `XNode` instead. `XmlNode` and  `XmlDocument` are not designed for exposing in public APIs.

✓ DO use `XmlReader`, `IXPathNavigable`, or subtypes of `XNode` as input or output of members that accept or return XML.

Use these abstractions instead of  `XmlDocument`,  `XmlNode`, or  `XPathDocument`, because this decouples the methods from specific implementations of an in-memory XML document and allows them to work with virtual XML data sources that expose `XNode`,  `XmlReader`, or  `XPathNavigator`.

✗ DO NOT subclass  `XmlDocument` if you want to create a type representing an XML view of an underlying object model or data source.

*Portions © 2005, 2009 Microsoft Corporation. All rights reserved.*

*Reprinted by permission of Pearson Education, Inc. from [Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries, 2nd Edition](#) by Krzysztof Cwalina and Brad Abrams, published Oct 22, 2008 by Addison-Wesley Professional as part of the Microsoft Windows Development Series.*

## See also

- [Framework Design Guidelines](#)
- [Usage Guidelines](#)

# Equality Operators

9/20/2022 • 2 minutes to read

This section discusses overloading equality operators and refers to `operator==` and `operator!=` as equality operators.

- ✗ DO NOT overload one of the equality operators and not the other.
- ✓ DO ensure that `Object.Equals` and the equality operators have exactly the same semantics and similar performance characteristics.

This often means that `Object.Equals` needs to be overridden when the equality operators are overloaded.

- ✗ AVOID throwing exceptions from equality operators.

For example, return false if one of the arguments is null instead of throwing `NullReferenceException`.

## Equality Operators on Value Types

- ✓ DO overload the equality operators on value types, if equality is meaningful.

In most programming languages, there is no default implementation of `operator==` for value types.

## Equality Operators on Reference Types

- ✗ AVOID overloading equality operators on mutable reference types.

Many languages have built-in equality operators for reference types. The built-in operators usually implement the reference equality, and many developers are surprised when the default behavior is changed to the value equality.

This problem is mitigated for immutable reference types because immutability makes it much harder to notice the difference between reference equality and value equality.

- ✗ AVOID overloading equality operators on reference types if the implementation would be significantly slower than that of reference equality.

*Portions © 2005, 2009 Microsoft Corporation. All rights reserved.*

*Reprinted by permission of Pearson Education, Inc. from [Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries, 2nd Edition](#) by Krzysztof Cwalina and Brad Abrams, published Oct 22, 2008 by Addison-Wesley Professional as part of the Microsoft Windows Development Series.*

## See also

- [Framework Design Guidelines](#)
- [Usage Guidelines](#)

# Common Design Patterns

9/20/2022 • 2 minutes to read

There are numerous books on software patterns, pattern languages, and antipatterns that address the very broad subject of patterns. Thus, this chapter provides guidelines and discussion related to a very limited set of patterns that are used frequently in the design of the .NET Framework APIs.

## In This Section

[Dependency Properties](#)

[Dispose Pattern](#)

*Portions © 2005, 2009 Microsoft Corporation. All rights reserved.*

*Reprinted by permission of Pearson Education, Inc. from [Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries, 2nd Edition](#) by Krzysztof Cwalina and Brad Abrams, published Oct 22, 2008 by Addison-Wesley Professional as part of the Microsoft Windows Development Series.*

## See also

- [Framework Design Guidelines](#)

# Dependency Properties

9/20/2022 • 3 minutes to read

A dependency property (DP) is a regular property that stores its value in a property store instead of storing it in a type variable (field), for example.

An attached dependency property is a kind of dependency property modeled as static Get and Set methods representing "properties" describing relationships between objects and their containers (e.g., the position of a `Button` object on a `Panel` container).

- ✓ DO provide the dependency properties, if you need the properties to support WPF features such as styling, triggers, data binding, animations, dynamic resources, and inheritance.

## Dependency Property Design

- ✓ DO inherit from `DependencyObject`, or one of its subtypes, when implementing dependency properties. The type provides a very efficient implementation of a property store and automatically supports WPF data binding.
- ✓ DO provide a regular CLR property and public static read-only field storing an instance of `System.Windows.DependencyProperty` for each dependency property.
- ✓ DO implement dependency properties by calling instance methods `DependencyObject.GetValue` and `DependencyObject.SetValue`.
- ✓ DO name the dependency property static field by suffixing the name of the property with "Property."

✗ DO NOT set default values of dependency properties explicitly in code; set them in metadata instead.

If you set a property default explicitly, you might prevent that property from being set by some implicit means, such as a styling.

✗ DO NOT put code in the property accessors other than the standard code to access the static field.

That code won't execute if the property is set by implicit means, such as a styling, because styling uses the static field directly.

✗ DO NOT use dependency properties to store secure data. Even private dependency properties can be accessed publicly.

## Attached Dependency Property Design

Dependency properties described in the preceding section represent intrinsic properties of the declaring type; for example, the `Text` property is a property of `TextButton`, which declares it. A special kind of dependency property is the attached dependency property.

A classic example of an attached property is the `Grid.Column` property. The property represents Button's (not Grid's) column position, but it is only relevant if the Button is contained in a Grid, and so it's "attached" to Buttons by Grids.

```

<Grid>
    <Grid.ColumnDefinitions>
        <ColumnDefinition />
        <ColumnDefinition />
    </Grid.ColumnDefinitions>

    <Button Grid.Column="0">Click</Button>
    <Button Grid.Column="1">Clack</Button>
</Grid>

```

The definition of an attached property looks mostly like that of a regular dependency property, except that the accessors are represented by static Get and Set methods:

```

public class Grid {

    public static int GetColumn(DependencyObject obj) {
        return (int)obj.GetValue(ColumnProperty);
    }

    public static void SetColumn(DependencyObject obj, int value) {
        obj.SetValue(ColumnProperty,value);
    }

    public static readonly DependencyProperty ColumnProperty =
        DependencyProperty.RegisterAttached(
            "Column",
            typeof(int),
            typeof(Grid)
        );
}

```

## Dependency Property Validation

Properties often implement what is called validation. Validation logic executes when an attempt is made to change the value of a property.

Unfortunately dependency property accessors cannot contain arbitrary validation code. Instead, dependency property validation logic needs to be specified during property registration.

**✗ DO NOT put dependency property validation logic in the property's accessors. Instead, pass a validation callback to `DependencyProperty.Register` method.**

## Dependency Property Change Notifications

**✗ DO NOT implement change notification logic in dependency property accessors. Dependency properties have a built-in change notifications feature that must be used by supplying a change notification callback to the `PropertyMetadata`.**

## Dependency Property Value Coercion

Property coercion takes place when the value given to a property setter is modified by the setter before the property store is actually modified.

**✗ DO NOT implement coercion logic in dependency property accessors.**

Dependency properties have a built-in coercion feature, and it can be used by supplying a coercion callback to the `PropertyMetadata`.

*Reprinted by permission of Pearson Education, Inc. from Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries, 2nd Edition* by Krzysztof Cwalina and Brad Abrams, published Oct 22, 2008 by Addison-Wesley Professional as part of the Microsoft Windows Development Series.

## See also

- [Framework Design Guidelines](#)
- [Common Design Patterns](#)

# Overview of porting from .NET Framework to .NET

9/20/2022 • 7 minutes to read • [Edit Online](#)

This article provides an overview of what you should consider when porting your code from .NET Framework to .NET (formerly named .NET Core). Porting to .NET from .NET Framework for many projects is relatively straightforward. The complexity of your projects dictates how much work you'll do after the initial migration of the project files.

Projects where the app-model is available in .NET (such as libraries, console apps, and desktop apps) usually require little change. Projects that require a new app model, such as moving to ASP.NET Core from ASP.NET, require more work. Many patterns from the old app model have equivalents that can be used during the conversion.

## Windows desktop technologies

Many applications created for .NET Framework use a desktop technology such as Windows Forms or Windows Presentation Foundation (WPF). Both Windows Forms and WPF have been ported to .NET, but these remain Windows-only technologies.

Consider the following dependencies before you migrate a Windows Forms or WPF application:

1. Project files for .NET use a different format than .NET Framework.
2. Your project may use an API that isn't available in .NET.
3. 3rd-party controls and libraries may not have been ported to .NET and remain only available to .NET Framework.
4. Your project uses a [technology that is no longer available](#) in .NET.

.NET uses the open-source versions of Windows Forms and WPF and includes enhancements over .NET Framework.

For tutorials on migrating your desktop application to .NET 6, see one of the following articles:

- [Migrate .NET Framework WPF apps to .NET](#)
- [Migrate .NET Framework Windows Forms apps to .NET](#)

## Windows-specific APIs

Applications can still P/Invoke native libraries on platforms supported by .NET. This technology isn't limited to Windows. However, if the library you're referencing is Windows-specific, such as a *user32.dll* or *kernel32.dll*, then the code only works on Windows. For each platform you want your app to run on, you'll have to either find platform-specific versions, or make your code generic enough to run on all platforms.

When porting an application from .NET Framework to .NET, your application probably used a library provided distributed with the .NET Framework. Many APIs that were available in .NET Framework weren't ported to .NET because they relied on Windows-specific technology, such as the Windows Registry or the GDI+ drawing model.

The **Windows Compatibility Pack** provides a large portion of the .NET Framework API surface to .NET and is provided via the [Microsoft.Windows.Compatibility NuGet package](#).

For more information, see [Use the Windows Compatibility Pack to port code to .NET](#).

## .NET Framework compatibility mode

The .NET Framework compatibility mode was introduced in .NET Standard 2.0. This compatibility mode allows .NET Standard and .NET (since .NET Core 3.1) projects to reference .NET Framework libraries on Windows-only. Referencing .NET Framework libraries doesn't work for all projects, such as if the library uses Windows Presentation Foundation (WPF) APIs, but it does unblock many porting scenarios. For more information, see the [Analyze your dependencies to port code from .NET Framework to .NET](#).

## Unavailable technologies

There are a few technologies in .NET Framework that don't exist in .NET:

- [Application domains](#)

Creating additional application domains isn't supported. For code isolation, use separate processes or containers as an alternative.

- [Remoting](#)

Remoting is used for communicating across application domains, which are no longer supported. For simple communication across processes, consider inter-process communication (IPC) mechanisms as an alternative to remoting, such as the [System.IO.Pipes](#) class or the [MemoryMappedFile](#) class. For more complex scenarios, consider frameworks such as [StreamJsonRpc](#) or [ASP.NET Core](#) (either using [gRPC](#) or [RESTful Web API services](#)).

- [Code access security \(CAS\)](#)

CAS was a sandboxing technique supported by .NET Framework but deprecated in .NET Framework 4.0. It was replaced by Security Transparency and it's not supported in .NET. Instead, use security boundaries provided by the operating system, such as virtualization, containers, or user accounts.

- [Security transparency](#)

Similar to CAS, this sandboxing technique is no longer recommended for .NET Framework applications and it's not supported in .NET. Instead, use security boundaries provided by the operating system, such as virtualization, containers, or user accounts.

- [System.EnterpriseServices](#)

[System.EnterpriseServices](#) (COM+) isn't supported in .NET.

- Windows Workflow Foundation (WF)

WF isn't supported in .NET. For an alternative, see [CoreWF](#).

For more information about these unsupported technologies, see [.NET Framework technologies unavailable on .NET Core and .NET 5+](#).

## Cross-platform

.NET (formerly known as .NET Core) is designed to be cross-platform. If your code doesn't depend on Windows-specific technologies, it may run on other platforms such as macOS, Linux, and Android. This includes project types like:

- Libraries
- Console-based tools
- Automation
- ASP.NET sites

.NET Framework is a Windows-only component. When your code uses Windows-specific technologies or APIs,

such as Windows Forms and Windows Presentation Foundation (WPF), the code can still run on .NET but it won't run on other operating systems.

It's possible that your library or console-based application can be used cross-platform without changing much. When porting to .NET, you may want to take this into consideration and test your application on other platforms.

## The future of .NET Standard

.NET Standard is a formal specification of .NET APIs that are available on multiple .NET implementations. The motivation behind .NET Standard was to establish greater uniformity in the .NET ecosystem. Starting with .NET 5, a different approach to establishing uniformity has been adopted, and this new approach eliminates the need for .NET Standard in many scenarios. For more information, see [.NET 5 and .NET Standard](#).

.NET Standard 2.0 was the last version to support .NET Framework.

## Tools to assist porting

Instead of manually porting an application from .NET Framework to .NET, you can use different tools to help automate some aspects of the migration. Porting a complex project is, in itself, a complex process. These tools may help in that journey.

Even if you use a tool to help port your application, you should review the [Considerations when porting](#) section in this article.

### **.NET Upgrade Assistant**

The [.NET Upgrade Assistant](#) is a command-line tool that can be run on different kinds of .NET Framework apps. It's designed to assist with upgrading .NET Framework apps to .NET. After running the tool, **in most cases the app will require more effort to complete the migration**. The tool includes the installation of analyzers that can assist with completing the migration. This tool works on the following types of .NET Framework applications:

- Windows Forms
- WPF
- ASP.NET MVC
- Console
- Class libraries

This tool uses the other tools listed in this article and guides the migration process. For more information about the tool, see [Overview of the .NET Upgrade Assistant](#).

### **try-convert**

The try-convert tool is a .NET global tool that can convert a project or entire solution to the .NET SDK, including moving desktop apps to .NET. However, this tool isn't recommended if your project has a complicated build process such as custom tasks, targets, or imports.

For more information, see the [try-convert GitHub repository](#).

### **.NET Portability Analyzer**

The .NET Portability Analyzer is a tool that analyzes assemblies and provides a detailed report on .NET APIs that are missing for the applications or libraries to be portable on your specified targeted .NET platforms.

To use the .NET Portability Analyzer in Visual Studio, install the [extension from the marketplace](#).

For more information, see [The .NET Portability Analyzer](#).

### **Platform compatibility analyzer**

The [Platform compatibility analyzer](#) analyzes whether or not you're using an API that will throw a [PlatformNotSupportedException](#) at run time. Although this isn't common if you're moving from .NET Framework 4.7.2 or higher, it's good to check. For more information about APIs that throw exceptions on .NET, see [APIs that always throw exceptions on .NET Core](#).

For more information, see [Platform compatibility analyzer](#).

## Considerations when porting

When porting your application to .NET, consider the following suggestions in order.

- ✓ CONSIDER using the [.NET Upgrade Assistant](#) to migrate your projects. Even though this tool is in preview, it automates most of the manual steps detailed in this article and gives you a great starting point for continuing your migration path.
- ✓ CONSIDER examining your dependencies first. Your dependencies must target .NET, .NET Standard, or .NET Core.
- ✓ DO migrate from a NuGet *packages.config* file to [PackageReference](#) settings in the project file. Use Visual Studio to [convert the package.config file](#).
- ✓ CONSIDER upgrading to the latest project file format even if you can't yet port your app. .NET Framework projects use an outdated project format. Even though the latest project format, known as SDK-style projects, was created for .NET Core and beyond, they work with .NET Framework. Having your project file in the latest format gives you a good basis for porting your app in the future.
- ✓ DO retarget your .NET Framework project to at least .NET Framework 4.7.2. This ensures the availability of the latest API alternatives for cases where .NET Standard doesn't support existing APIs.
- ✓ CONSIDER targeting .NET 6 which is under long-term support (LTS).
- ✓ DO target .NET 6 for **Windows Forms and WPF** projects. .NET 6 contains many improvements for Desktop apps.
- ✓ CONSIDER targeting .NET Standard 2.0 if you're migrating a library that may also be used with .NET Framework projects. You can also multitarget your library, targeting both .NET Framework and .NET Standard.
- ✓ DO add reference to the [Microsoft.Windows.Compatibility NuGet package](#) if, after migrating, you get errors of missing APIs. A large portion of the .NET Framework API surface is available to .NET via the NuGet package.

## See also

- [Overview of the .NET Upgrade Assistant](#)
- [ASP.NET to ASP.NET Core migration](#)
- [Migrate .NET Framework WPF apps to .NET](#)
- [Migrate .NET Framework Windows Forms apps to .NET](#)
- [.NET 5 vs. .NET Framework for server apps](#)

# Overview of .NET, MSBuild, and Visual Studio versioning

9/20/2022 • 3 minutes to read • [Edit Online](#)

Understanding the versioning of the .NET SDK and how it relates to Visual Studio and MSBuild can be confusing. MSBuild versions with Visual Studio, but is also included in the .NET SDK. The SDK has a minimum version of MSBuild and Visual Studio that it works with, and it won't load in a version of Visual Studio that's older than that minimum version.

## Versioning

The first part of the .NET SDK version matches the .NET version that it includes, runs on, and targets by default. The feature band starts at 1 and increases for each quarterly Visual Studio minor release. The patch version increments with each month's servicing updates.

For example, version 5.0.203 ships with .NET 5, is the second minor Visual Studio release since 5.0.100 first came out, and is the third patch since 5.0.200 released.

## Lifecycle

The support timeframe for the SDK typically matches that of the Visual Studio version it's included in.

SDK VERSION	MSBUILD/VISUAL STUDIO VERSION	SHIP DATE	LIFECYCLE
2.1.5xx	15.9	Nov '18	Aug '21 ¹
2.1.8xx	16.2 (No VS)	July '19	Aug '21
3.1.1xx	16.4	Dec '19	Oct '21
3.1.4xx	16.7	Aug '20	Dec '22
5.0.1xx	16.8	Nov '20	Mar '21
5.0.2xx	16.9	March '21	May '22 ¹
5.0.3xx	16.10	May '21	Aug '21
5.0.4xx	16.11	Aug '21	May '22 ¹
6.0.100	17.0 ²	Nov '21	Jul '23
6.0.200	17.1	Feb '22	May '22
6.0.300	17.2 ³	May '22	Oct '22
6.0.400	17.3	Aug '22	Nov '24 ²

SDK VERSION	MSBUILD/VISUAL STUDIO VERSION	SHIP DATE	LIFECYCLE
7.0.100	17.4	Nov '22	TBD
7.0.200	17.5	TBD	TBD

#### NOTE

Targeting `net6.0` is officially supported in Visual Studio 17.0+ only.

¹ The .NET 5 SDK will be supported in Visual Studio scenarios until December 2022 when 3.1 goes out of support. MSBuild/Visual Studio supported for longer.

² Visual Studio 17.3 will be in support until 17.4 ships

[Visual Studio 2019 Lifecycle](#)

[Visual Studio 2022 Lifecycle](#)

² With .NET 6, the .NET 6.0.100 SDK can be used in version 16.11 for **downlevel** targeting. This means that you're not forced to update your SDK and Visual Studio versions simultaneously. However, you won't be able to target .NET 6 because of limitations in 6.0 features and C# 10 features in version 16.11. This compatibility is specifically for targeting 5.0 and below.

³ 6.0.300 and newer SDKs require a minimum Visual Studio version of 17.0.

## Targeting and support rules

Starting with .NET SDK 7.0.100 and .NET SDK 6.0.300, a policy has been put into place regarding which versions of MSBuild and Visual Studio a given version of the .NET SDK will run in. The policy is:

- Each new TargetFramework **requires** a new Visual Studio version or a new `dotnet` version.
- The first version of Visual Studio that supports a new TargetFramework becomes a floor for the feature bands of that SDK for Roslyn API surface, msbuild targets, source generators, analyzers, and so on.
- The first version of a new .NET SDK that supports a new TargetFramework can still be used with the prior version of Visual Studio to allow one quarter for tooling and infrastructure (for example, actions and pipelines) to migrate.

SDK	VISUAL STUDIO VERSION THE SDK SHIPS WITH	MINIMUM VISUAL STUDIO VERSION	MAX TARGETFRAMEWORK IN MINIMUM VISUAL STUDIO VERSION	MAX TARGETFRAMEWORK IN <code>DOTNET</code>
6.0.100	17.0	16.11	Net5.0	Net6.0
6.0.200	17.1	17.0	Net6.0	Net6.0
6.0.300	17.2	17.0	Net6.0	Net6.0
6.0.400	17.3	17.0	Net6.0	Net6.0
7.0.100	17.4	17.3	Net6.0	Net7.0

SDK	VISUAL STUDIO VERSION THE SDK SHIPS WITH	MINIMUM VISUAL STUDIO VERSION	MAX TARGETFRAMEWORK IN MINIMUM VISUAL STUDIO VERSION	MAX TARGETFRAMEWORK IN DOTNET
7.0.200	17.5	17.4	Net7.0	Net7.0
7.0.300	17.6	17.4	Net7.0	Net7.0
7.0.400	17.7	17.4	Net7.0	Net7.0

#### NOTE

The table depicts how these versioning rules will be applied going forward, starting with .NET SDK 7.0.100 and .NET SDK 6.0.300. It also depicts how the policy would have applied to previously shipped versions of the .NET SDK, had it been in place then. However, the requirements for previous versions of the SDK don't change — that is, the minimum required version of Visual Studio for .NET SDK 6.0.100 or 6.0.200 remains 16.10.

To ensure consistent tooling, you should use `dotnet build` rather than `msbuild` to build your application when possible.

## Reference

- [Overview of how .NET is versioned](#)
- [.NET and .NET Core official support policy](#)
- [Microsoft .NET and .NET Core](#)
- [.NET Downloads \(Windows, Linux, and macOS\)](#)
- [Visual Studio 2019 Product Lifecycle and Servicing](#)

# .NET vs. .NET Framework for server apps

9/20/2022 • 5 minutes to read • [Edit Online](#)

There are two supported [.NET implementations](#) for building server-side apps.

IMPLEMENTATION	INCLUDED VERSIONS
.NET	.NET Core 1.0 - 3.1, .NET 5, and later versions of .NET.
.NET Framework	.NET Framework 1.0 - 4.8

Both share many of the same components, and you can share code across the two. However, there are fundamental differences between the two and your choice depends on what you want to accomplish. This article provides guidance on when to use each.

Use .NET for your server application when:

- You have cross-platform needs.
- You're targeting microservices.
- You're using Docker containers.
- You need high-performance and scalable systems.
- You need side-by-side .NET versions per application.

Use .NET Framework for your server application when:

- Your app currently uses .NET Framework (recommendation is to extend instead of migrating).
- Your app uses third-party libraries or NuGet packages not available for .NET.
- Your app uses .NET Framework technologies that aren't available for .NET.
- Your app uses a platform that doesn't support .NET.

## When to choose .NET

The following sections give a more detailed explanation of the previously stated reasons for picking .NET over .NET Framework.

### Cross-platform needs

If your web or service application needs to run on multiple platforms, for example, Windows, Linux, and macOS, use .NET.

.NET supports the previously mentioned operating systems as your development workstation. Visual Studio provides an Integrated Development Environment (IDE) for Windows and macOS. You can also use Visual Studio Code, which runs on macOS, Linux, and Windows. Visual Studio Code supports .NET, including IntelliSense and debugging. Most third-party editors, such as Sublime, Emacs, and VI, work with .NET. These third-party editors get editor IntelliSense using [Omnisharp](#). You can also avoid any code editor and directly use the [.NET CLI](#), available for all supported platforms.

### Microservices architecture

A microservices architecture allows a mix of technologies across a service boundary. This technology mix enables a gradual embrace of .NET for new microservices that work with other microservices or services. For example, you can mix microservices or services developed with .NET Framework, Java, Ruby, or other monolithic technologies.

There are many infrastructure platforms available. [Azure Service Fabric](#) is designed for large and complex microservice systems. [Azure App Service](#) is a good choice for stateless microservices. Microservices alternatives based on Docker fit any kind of microservices approach, as explained in the [Containers](#) section. All these platforms support .NET and make them ideal for hosting your microservices.

For more information about microservices architecture, see [.NET Microservices. Architecture for Containerized .NET Applications](#).

## Containers

Containers are commonly used in conjunction with a microservices architecture. Containers can also be used to containerize web apps or services that follow any architectural pattern. .NET Framework can be used on Windows containers, but the modularity and lightweight nature of .NET makes it a better choice for containers. When creating and deploying a container, the size of its image is much smaller with .NET than with .NET Framework. Because it's cross-platform, you can deploy server apps to Linux Docker containers, for example.

Docker containers can be hosted in your own Linux or Windows infrastructure, or in a cloud service such as [Azure Kubernetes Service](#). Azure Kubernetes Service can manage, orchestrate, and scale container-based applications in the cloud.

## High-performance and scalable systems

When your system needs the best possible performance and scalability, .NET and ASP.NET Core are your best options. The high-performance server runtime for Windows Server and Linux makes ASP.NET Core a top performing web framework on [TechEmpower benchmarks](#).

Performance and scalability are especially relevant for microservices architectures, where hundreds of microservices may be running. With ASP.NET Core, systems run with a much lower number of servers/Virtual Machines (VM). The reduced servers/VMs save costs in infrastructure and hosting.

## Side by side .NET versions per application level

To install applications with dependencies on different versions of .NET, we recommend .NET. This implementation supports side-by-side installation of different versions of the .NET runtime on the same machine. This side-by-side installation allows multiple services on the same server, each of them on its own version of .NET. It also lowers risks and saves money in application upgrades and IT operations.

Side-by-side installation isn't possible with .NET Framework. It's a Windows component, and only one version can exist on a machine at a time. Each version of .NET Framework replaces the previous version. If you install a new app that targets a later version of .NET Framework, you might break existing apps that run on the machine, because the previous version was replaced.

# When to choose .NET Framework

.NET offers significant benefits for new applications and application patterns. However, .NET Framework continues to be the natural choice for many existing scenarios, and as such, .NET Framework isn't replaced by .NET for all server applications.

## Current .NET Framework applications

In most cases, you don't need to migrate your existing applications to .NET. Instead, a recommended approach is to use .NET as you extend an existing application, such as writing a new web service in ASP.NET Core.

## Third-party libraries or NuGet packages not available for .NET

.NET Standard enables sharing code across all .NET implementations, including .NET Core/5+. With .NET Standard 2.0, a compatibility mode allows .NET Standard and .NET projects to reference .NET Framework libraries. For more information, see [Support for .NET Framework libraries](#).

You need to use .NET Framework only in cases where the libraries or NuGet packages use technologies that aren't available in .NET Standard or .NET.

## **.NET Framework technologies not available for .NET**

Some .NET Framework technologies aren't available in .NET. The following list shows the most common technologies not found in .NET:

- ASP.NET Web Forms applications: ASP.NET Web Forms are only available in .NET Framework. ASP.NET Core cannot be used for ASP.NET Web Forms.
- ASP.NET Web Pages applications: ASP.NET Web Pages aren't included in ASP.NET Core.
- Workflow-related services: Windows Workflow Foundation (WF), Workflow Services (WCF + WF in a single service), and WCF Data Services (formerly known as "ADO.NET Data Services") are only available in .NET Framework.
- Language support: Visual Basic and F# are currently supported in .NET, but not for all project types. For a list of supported project templates, see [Template options for dotnet new](#).

For more information, see [.NET Framework technologies unavailable in .NET](#).

## **Platform doesn't support .NET**

Some Microsoft or third-party platforms don't support .NET. Some Azure services provide an SDK not yet available for consumption on .NET. In such cases, you can use the equivalent REST API instead of the client SDK.

## See also

- [Choose between ASP.NET and ASP.NET Core](#)
- [ASP.NET Core targeting .NET Framework](#)
- [Target frameworks](#)
- [.NET introduction](#)
- [Porting from .NET Framework to .NET 5](#)
- [Introduction to .NET and Docker](#)
- [.NET implementations](#)
- [.NET Microservices. Architecture for Containerized .NET Applications](#)

# Overview of the .NET Upgrade Assistant

9/20/2022 • 2 minutes to read • [Edit Online](#)

You might have apps that currently run on the .NET Framework that you're interested in porting to .NET 6. The .NET Upgrade Assistant tool can assist with this process. This article provides:

- An overview of the .NET Upgrade Assistant.
- How to install the .NET Upgrade Assistant.

## What is the .NET Upgrade Assistant

The .NET Upgrade Assistant is a command-line tool that can be run on different kinds of .NET Framework apps. It's designed to assist with upgrading .NET Framework apps to .NET 6. After running the tool, **in most cases the app will require additional effort to complete the migration**. The tool includes the installation of analyzers that can assist with completing the migration.

Currently the tool supports the following .NET Framework app types:

- .NET Framework Windows Forms apps
- .NET Framework WPF apps
- .NET Native UWP apps
- .NET Framework ASP.NET MVC apps
- .NET Framework console apps
- .NET Framework class libraries

The tool also supports projects that use these code languages:

- C#
- Visual Basic.NET

The .NET Upgrade Assistant is currently prerelease and is receiving frequent updates. If you discover problems using the tool, report them in the tool's [GitHub repository](#).

## Install the .NET Upgrade Assistant

This section describes how to install the .NET Upgrade Assistant. You can also follow [a step-by-step guided tutorial](#).

### Prerequisites

- Windows Operating System
- [.NET 6 SDK](#)
- [Visual Studio 2022 17.0 or later](#)

### Installation steps

The .NET Upgrade Assistant is a .NET tool that is installed globally with the following command:

```
dotnet tool install -g upgrade-assistant
```

Similarly, because the .NET Upgrade Assistant is installed as a .NET tool, it can be easily updated by running:

```
dotnet tool update -g upgrade-assistant
```

#### IMPORTANT

Installing this tool may fail if you've configured additional NuGet feed sources. Use the `--ignore-failed-sources` parameter to treat those failures as warnings instead of errors:

```
dotnet tool install -g --ignore-failed-sources upgrade-assistant
```

## See also

- [Upgrade an ASP.NET MVC App to .NET 6](#)
- [Upgrade a WPF App to .NET 6](#)
- [Upgrade a Windows Forms App to .NET 6](#)
- [.NET Upgrade Assistant GitHub Repository](#)

# Upgrade a WPF App to .NET 6 with the .NET Upgrade Assistant

9/20/2022 • 15 minutes to read • [Edit Online](#)

The [.NET Upgrade Assistant](#) is a command-line tool that can assist with upgrading .NET Framework WPF apps to .NET 6. This article provides:

- A demonstration of how to run the tool against a .NET Framework WPF app
- Troubleshooting tips

For more information on how to install the tool, see [Overview of the .NET Upgrade Assistant](#).

## Demo app

You can use the [Basic WPF Sample](#) project to test upgrading with the Upgrade Assistant.

## Analyze your app

The .NET Upgrade Assistant tool includes an analyze mode that performs a simplified dry run of upgrading your app. It may provide insights as to what changes may be required before the upgrade is started. Open a terminal and navigate to the folder where the target project or solution is located. Run the `upgrade-assistant analyze` command, passing in the name of the project or solution you're upgrading.

For example, running the analyze mode with the [Basic WPF Sample](#) app produces the following output, indicating that there aren't any changes to be made before upgrading:

```
> upgrade-assistant analyze .\WebSiteRatings.sln

[09:50:56 INF] Loaded 5 extensions
[09:50:57 INF] Using MSBuild from C:\Program Files\dotnet\sdk\6.0.101\
[09:50:57 INF] Using Visual Studio install from C:\Program Files\Microsoft Visual Studio\2022\Preview [v17]
[09:50:59 INF] Recommending executable TFM net6.0 because the project builds to a web app
[09:50:59 INF] Recommending Windows TFM net6.0-windows because the project either has Windows-specific
dependencies or builds to a WinExe
[09:50:59 INF] Marking assembly reference System.Configuration for removal based on package mapping
configuration System.Configuration
[09:50:59 INF] Adding package System.Configuration.ConfigurationManager based on package mapping
configuration System.Configuration
[09:50:59 INF] Marking assembly reference System.Web for removal based on package mapping configuration
ASP.NET
[09:50:59 INF] Adding framework reference Microsoft.AspNetCore.App based on package mapping configuration
ASP.NET
[09:50:59 INF] Marking assembly reference System.Web.Extensions for removal based on package mapping
configuration ASP.NET
[09:51:00 INF] Reference to .NET Upgrade Assistant analyzer package
(Microsoft.DotNet.UpgradeAssistant.Extensions.Default.Analyzers, version 0.3.261602) needs to be added
[09:51:01 INF] Adding Microsoft.Windows.Compatibility 6.0.0
[09:51:01 INF] Recommending Windows TFM net6.0-windows because the project either has Windows-specific
dependencies or builds to a WinExe
[09:51:01 INF] Reference to .NET Upgrade Assistant analyzer package
(Microsoft.DotNet.UpgradeAssistant.Extensions.Default.Analyzers, version 0.3.261602) needs to be added
[09:51:01 INF] Adding Microsoft.Windows.Compatibility 6.0.0
[09:51:01 INF] Running analyzers on WebSiteRatings
[09:51:02 INF] Identified 0 diagnostics in project WebSiteRatings
[09:51:02 INF] Running analyzers on StarVoteControl
[09:51:02 INF] Identified 0 diagnostics in project StarVoteControl
```

There's quite a bit of internal diagnostic information in the output, but some information is helpful. Notice that the analyze mode indicates that the upgrade will recommend that the project target the `net6.0-windows` target framework moniker ([TFM](#)). This is because the projects referenced by the solution are WPF projects, a Windows-only technology. A console application would probably get the recommendation to upgrade to TFM `net6.0` directly, unless it used some Windows-specific libraries.

If any errors or warnings are reported, take care of them before you start an upgrade.

## Run upgrade-assistant

Open a terminal and navigate to the folder where the target project or solution is located. Run the `upgrade-assistant upgrade` command, passing in the name of the project or solution you're upgrading.

When a project is provided, the upgrade process starts on that project immediately. If a solution is provided, you'll select which project you normally run, known as the [upgrade endpoint](#). Based on that project, a dependency graph is created and a suggestion as to which order you should upgrade the projects is provided.

```
upgrade-assistant upgrade .\WebSiteRatings.sln
```

The tool runs and shows you a list of the steps it will do. As each step is completed, the tool provides a set of commands allowing the user to apply or skip the next step or some other option such as:

- Get more information about the step.
- Change projects.
- Adjust logging settings.
- Stop the upgrade and quit.

Pressing Enter without choosing a number selects the first item in the list.

As each step initializes, it may provide information about what it thinks will happen if you apply the step.

## Select the Entrypoint

The first step in upgrading the [example app](#) is choosing which project in the solution serves as the entrypoint project.

### Upgrade Steps

1. [Next step] Select an entrypoint
2. Select project to upgrade

### Choose a command:

1. Apply next step (Select an entrypoint)
2. Skip next step (Select an entrypoint)
3. See more step details
4. Configure logging
5. Exit

Choose **command 1** to start that step. The results are displayed:

```
[13:28:49 INF] Applying upgrade step Select an entrypoint
Please select the project you run. We will then analyze the dependencies and identify the recommended order
to upgrade projects.
1. StarVoteControl
2. WebSiteRatings
```

There are two projects listed: The main WPF app (WebSiteRatings) and the UserControl project (StarVoteControl). Select the WebSiteRatings project for the entrypoint, which is **item 2**.

## Select a project to upgrade

After the entrypoint is determined, the next step is to choose which project to upgrade first. In this example, the tool determined that the WPF UserControl project (StarVoteControl) should be upgraded first since the main WPF app project has a dependency on the control.

```
[13:44:47 INF] Applying upgrade step Select project to upgrade
Here is the recommended order to upgrade. Select enter to follow this list, or input the project you want to
start with.
1. StarVoteControl
2. WebSiteRatings
```

The recommended path here, is to first upgrade the **StarVoteControl** project first, since the **WebSiteRatings** project depends on it. It's best to follow the recommended upgrade path.

Upgrading the **StarVoteControl** is a simple project and doesn't present any post-upgrade problems. Therefore, this article continues on as if that project has already been upgraded and the next project to upgrade is **WebSiteRatings**.

## Upgrade the project

Once a project is selected, a list of upgrade steps the tool will take is listed.

## IMPORTANT

For the purposes of this example, the **WebSiteRatings** project is described. It's assumed that the **StarVoteControl** project was successfully upgraded. The reason for demonstrating the **WebSiteRatings** project is that it contains more of the common issues you'll run into when upgrading an app.

Based on the project you're upgrading, you may or may not see every step listed in this example.

The following output describes the steps involved in upgrading the project:

```
[16:09:24 INF] Initializing upgrade step Back up project

Upgrade Steps

Entrypoint: C:\code\migration\wpf\sampleApp\WebSiteRatings\WebSiteRatings.csproj
Current Project: C:\code\migration\wpf\sampleApp\WebSiteRatings\WebSiteRatings.csproj

1. [Next step] Back up project
2. Convert project file to SDK style
3. Clean up NuGet package references
4. Update TFM
5. Update NuGet Packages
6. Add template files
7. Upgrade app config files
   a. Convert Application Settings
   b. Convert Connection Strings
   c. Disable unsupported configuration sections
8. Update source code
   a. Apply fix for UA0002: Types should be upgraded
   b. Apply fix for UA0012: 'UnsafeDeserialize()' does not exist
9. Move to next project

Choose a command:
1. Apply next step (Back up project)
2. Skip next step (Back up project)
3. See more step details
4. Select different project
5. Configure logging
6. Exit
```

## NOTE

For the rest of this article, the upgrade steps aren't explicitly shown unless there is something important to call out. The results of each step are still shown.

### Create a backup

In this example of upgrading the project, you'll apply each step. The first step, **command 1**, is backing up the project:

```
[16:10:42 INF] Applying upgrade step Back up project
Please choose a backup path
1. Use default path [C:\code\migration\wpf\sampleApp.backup]
2. Enter custom path
```

The tool chooses a default backup path named after the current folder, but with `.backup` appended to it. You can choose a custom path as an alternative to the default path. For each upgraded project, the folder of the project is copied to the backup folder. In this example, the `WebSiteRatings` folder is copied from `sampleApp\WebSiteRatings` to `sampleApp.backup\WebSiteRatings` during the backup step:

```
[16:10:44 INF] Backing up C:\code\migration\wpf\sampleApp\WebSiteRatings to  
C:\code\migration\wpf\sampleApp.backup\WebSiteRatings  
[16:10:45 INF] Project backed up to C:\code\migration\wpf\sampleApp.backup\WebSiteRatings  
[16:10:45 INF] Upgrade step Back up project applied successfully  
Please press enter to continue...
```

## Upgrade the project file

The project is upgraded from the .NET Framework project format to the .NET SDK project format.

```
[16:10:51 INF] Applying upgrade step Convert project file to SDK style  
[16:10:51 INF] Converting project file format with try-convert, version  
0.3.261602+8aa571efd8bac422c95c35df9c7b9567ad534ad0  
[16:10:51 INF] Recommending TFM net6.0-windows because of dependency on project  
C:\code\migration\wpf\sampleApp\StarVoteControl\StarVoteControl.csproj  
C:\code\migration\wpf\sampleApp\WebSiteRatings\WebSiteRatings.csproj contains an App.config file. App.config  
is replaced by appsettings.json in .NET Core. You will need to delete App.config and migrate to  
appsettings.json if it's applicable to your project.  
[16:10:52 INF] Converting project C:\code\migration\wpf\sampleApp\WebSiteRatings\WebSiteRatings.csproj to  
SDK style  
[16:10:53 INF] Project file converted successfully! The project may require additional changes to build  
successfully against the new .NET target.  
[16:10:55 INF] Upgrade step Convert project file to SDK style applied successfully  
Please press enter to continue...
```

Pay attention to the output of each step. Notice that the example output indicates that you'll have a manual step to complete after the upgrade:

App.config is replaced by appsettings.json in .NET Core. You will need to delete App.config and migrate to appsettings.json if it's applicable to your project.

As part of this upgrade step, the NuGet packages referenced by the *packages.config* are migrated to the project file.

## Clean up NuGet references

Once the project format has been updated, the next step is to clean up the NuGet package references.

In addition to the packages referenced by your app, the *packages.config* file contains references to the dependencies of those packages. For example, if you added reference to package A which depends on package B, both packages would be referenced in the *packages.config* file. In the new project system, only the reference to package A is required. This step analyzes the package references and removes those that aren't required.

```
[16:55:18 INF] Applying upgrade step Clean up NuGet package references  
[16:55:18 INF] Removing outdated assembly reference: System.Configuration  
[16:55:18 INF] Removing outdated package reference: ControlzEx, Version=4.4.0  
[16:55:18 INF] Removing outdated package reference: Microsoft.Xaml.Behaviors.Wpf, Version=1.1.19  
[16:55:18 INF] Removing outdated package reference: SQLite.Native, Version=3.12.3  
[16:55:18 INF] Adding package reference: System.Configuration.ConfigurationManager, Version=5.0.0  
[16:55:18 INF] Adding package reference: Microsoft.DotNet.UpgradeAssistant.Extensions.Default.Analyzers,  
Version=0.3.261602  
[16:55:21 WRN] No version of System.Configuration.ConfigurationManager found that supports ["net452"];  
leaving unchanged  
[16:55:21 INF] Upgrade step Clean up NuGet package references applied successfully  
Please press enter to continue...
```

Your app is still referencing .NET Framework assemblies. Some of those assemblies may be available as NuGet packages. This step analyzes those assemblies and references the appropriate NuGet package.

## Handle the TFM

The tool next changes the **TFM** from .NET Framework to the suggested SDK. In this example, it's `net6.0-windows`.

```
[16:56:36 INF] Applying upgrade step Update TFM
[16:56:36 INF] Recommending TFM net6.0-windows because of dependency on project
C:\code\migration\wpf\sampleApp\StarVoteControl\StarVoteControl.csproj
[16:56:40 INF] Updated TFM to net6.0-windows
[16:56:40 INF] Upgrade step Update TFM applied successfully
Please press enter to continue...
```

### Upgrade NuGet packages

Next, the tool updates the project's NuGet packages to the versions that support the updated TFM,

net6.0-windows .

```
[16:58:06 INF] Applying upgrade step Update NuGet Packages
[16:58:06 INF] Removing outdated package reference: Microsoft.CSharp, Version=4.7.0
[16:58:06 INF] Removing outdated package reference: System.Data.DataSetExtensions, Version=4.5.0
[16:58:06 INF] Removing outdated package reference: EntityFramework, Version=6.2.0
[16:58:06 INF] Adding package reference: EntityFramework, Version=6.4.4
[16:58:11 INF] Upgrade step Update NuGet Packages applied successfully
Please press enter to continue...
```

### Templates, config, and code files

The next few steps may be skipped automatically by the tool if the tool determines there isn't anything to do for your project.

Once the packages are updated, the next step is to update any template files. In this example, there are no template files that need to be updated or added to the project. This step is skipped and the next step is automatically started: Upgrade app config files. In this example, the step only needs to convert the connection strings:

```
[17:02:52 INF] Applying upgrade step Convert Connection Strings
[17:02:53 INF] Upgrade step Convert Connection Strings applied successfully
[17:02:53 INF] Applying upgrade step Upgrade app config files
[17:02:53 INF] Upgrade step Upgrade app config files applied successfully
```

The final step before this project's upgrade is completed, is to update any out-of-date code references. Based on the type of project you're upgrading, a list of known code fixes is displayed for this step. Some of the fixes may not apply to your project.

8. Update source code
  - a. Apply fix for UA0002: Types should be upgraded
  - b. Apply fix for UA0012: 'UnsafeDeserialize()' does not exist

In this case, none of the suggested fixes apply to the example project, and this step automatically completes immediately after the previous step was completed.

```
[17:02:58 INF] Initializing upgrade step Update source code
[17:02:58 INF] Running analyzers on WebSiteRatings
[17:02:59 INF] Identified 0 diagnostics in project WebSiteRatings
[17:02:59 INF] Initializing upgrade step Move to next project
```

### Completing the upgrade

If there are any more projects to migrate, the tool lets you select which project to upgrade next. When there are no more projects to upgrade, the tool brings you to the "Finalize upgrade" step:

## 1. [Next step] Finalize upgrade

Choose a command:

1. Apply next step (Finalize upgrade)
2. Skip next step (Finalize upgrade)
3. See more step details
4. Configure logging
5. Exit

Once the upgrade is complete, the migrated WPF project looks like the following XML:

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <TargetFramework>net6.0-windows</TargetFramework>
    <OutputType>WinExe</OutputType>
    <GenerateAssemblyInfo>false</GenerateAssemblyInfo>
    <UseWPF>true</UseWPF>
    <ImportWindowsDesktopTargets>true</ImportWindowsDesktopTargets>
  </PropertyGroup>
  <ItemGroup>
    <None Update="sqlite.db">
      <CopyToOutputDirectory>PreserveNewest</CopyToOutputDirectory>
    </None>
  </ItemGroup>
  <ItemGroup>
    <ProjectReference Include=".\\StarVoteControl\\StarVoteControl.csproj" />
  </ItemGroup>
  <ItemGroup>
    <PackageReference Include="System.Configuration.ConfigurationManager" Version="5.0.0" />
    <PackageReference Include="Microsoft.DotNet.UpgradeAssistant.Extensions.Default.Analyzers"
Version="0.3.261602">
      <PrivateAssets>all</PrivateAssets>
    </PackageReference>
    <PackageReference Include="EntityFramework" Version="6.4.4" />
  </ItemGroup>
  <ItemGroup>
    <Service Include="{508349B6-6B84-4DF5-91F0-309BEEBAD82D}" />
  </ItemGroup>
  <ItemGroup>
    <PackageReference Include="MahApps.Metro" Version="2.4.9" />
    <PackageReference Include="Microsoft.Data.Sqlite" Version="1.0.0" />
    <PackageReference Include="SQLite" Version="3.12.3" />
  </ItemGroup>
  <ItemGroup>
    <Content Include="appsettings.json" />
  </ItemGroup>
</Project>
```

Notice that the .NET Upgrade Assistant also adds analyzers to the project that assist with continuing the upgrade process, such as the `Microsoft.DotNet.UpgradeAssistant.Extensions.Default.Analyzers` NuGet package.

## After upgrading

After you upgrade your projects, you'll need to compile and test them. Most certainly you'll have more work to do in finishing the upgrade. It's possible that the .NET Framework version of your app contained library references that your project isn't actually using. You'll need to analyze each reference and determine whether or not it's required. The tool may have also added or upgraded a NuGet package reference to wrong version.

At the time this article was published, the following updates are needed to complete the upgrade of the example project:

- Upgrade the `System.Configuration.ConfigurationManager` NuGet package to version 6.0.0. The wrong

version (5.0.0) was selected by the upgrade tool:

```
<PackageReference Include="System.Configuration.ConfigurationManager" Version="6.0.0" />
```

Once that item is fixed, the example app compiles and runs. However, there are more things that could be upgraded, for example, this app is using the `Microsoft.Data.Sqlite 1.0.0` NuGet package, the last version supporting .NET Framework directly. This package has a dependency on the `SQLite` package. If `Microsoft.Data.Sqlite` is upgraded for `6.0.0`, that dependency is removed.

## Modernize: `appsettings.json`

.NET Framework uses the `App.config` file to load settings for your app, such as connection strings and logging providers. .NET now uses the `appsettings.json` file for app settings.

`App.config` files are supported in .NET through the `System.Configuration.ConfigurationManager` NuGet package, and support for `appsettings.json` is provided by the `Microsoft.Extensions.Configuration` NuGet package.

As other libraries upgrade to .NET, they'll modernize by supporting `appsettings.json` instead of `App.config`. For example, logging providers in .NET Framework that have been upgraded for .NET 6 no longer use `App.config` for settings. It's good for you to follow their direction and also move away from using `App.config`.

With the WPF example app upgraded in the preceding section, we can remove the dependency on `System.Configuration.ConfigurationManager` and move to `appsettings.json` for the connection string used by the local database.

1. Remove the `System.Configuration.ConfigurationManager` NuGet package.
2. Add both the `Microsoft.Extensions.Configuration` and `Microsoft.Extensions.Configuration.Json` NuGet packages.

There are a variety of related `Microsoft.Extensions.Configuration` related NuGet packages your app may require.

3. Delete the `App.config` file from the project.

In the example app, this file only contained a single connection string, which was migrated to the `appsettings.json` file by the upgrade tool.

4. Set the `appsettings.json` file to copy to the output directory.

Set this through Visual Studio using the **Properties** pane, or edit the project directly and add the following `ItemGroup`:

```
<ItemGroup>
  <Content Include="appsettings.json">
    <CopyToOutputDirectory>PreserveNewest</CopyToOutputDirectory>
  </Content>
</ItemGroup>
```

5. Edit the `App.xaml.cs` file, setting up a configuration object that loads the `appsettings.json` file, the added lines are highlighted:

```

using System.Windows;
using Microsoft.Extensions.Configuration;

namespace WebSiteRatings
{
    /// <summary>
    /// Interaction logic for App.xaml
    /// </summary>
    public partial class App : Application
    {
        public static IConfiguration Config { get; private set; }

        public App()
        {
            Config = new ConfigurationBuilder()
                .AddJsonFile("appsettings.json")
                .Build();
        }
    }
}

```

6. In the `.\Models\Database.cs` file, change the `OpenConnection` method to use the new `App.Config` property. This requires importing the `Microsoft.Extensions.Configuration` namespace:

```

using Microsoft.Data.Sqlite;
using System.Collections.Generic;
using Microsoft.Extensions.Configuration;

namespace WebSiteRatings.Models
{
    internal class Database
    {
        public static SqliteConnection OpenConnection() =>
            new SqliteConnection(App.Config.GetConnectionString("database"));

        // More code below...
    }
}

```

`GetConnectionString` is an extension method provided by the `Microsoft.Extensions.Configuration` namespace.

## Modernize: Web browser control

The `WebBrowser` control referenced by the project is based on Internet Explorer, which is out-of-date. WPF for .NET includes a new browser control based on Microsoft Edge. Complete the following steps to upgrade to the new `WebView2` web browser control:

1. Add reference to the `Microsoft.Web.WebView2` NuGet package.
2. In the `MainWindow.xaml` file:
  - a. Import the control to the `wpfControls` namespace in the root element:

```

<mah:MetroWindow x:Class="WebSiteRatings.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:mah="clr-namespace:MahApps.Metro.Controls;assembly=MahApps.Metro"
    xmlns:local="clr-namespace:WebSiteRatings"
    xmlns:vm="clr-namespace:WebSiteRatings.ViewModels"
    xmlns:VoteControl="clr-namespace:StarVoteControl;assembly=StarVoteControl"
    xmlns:wpfControls="clr-
    namespace:Microsoft.Web.WebView2.Wpf;assembly=Microsoft.Web.WebView2.Wpf"
    Loaded="MetroWindow_Loaded"
    mc:Ignorable="d"
    Title="My Sites" Height="650" Width="1000">

```

- b. Down where the `<Border>` element is declared, remove the `WebBrowser` control and replace it with the `wpfControls:WebView2` control:

```

<Border Grid.Row="2" Grid.Column="1" Grid.ColumnSpan="2" BorderThickness="1"
BorderBrush="Black" Margin="5">
    <wpfControls:WebView2 x:Name="browser" ScrollViewer.CanContentScroll="True" />
</Border>

```

3. Edit the `MainWindow.xaml.cs` code behind file. Update the `ListBox_SelectionChanged` method to set the `browser.Source` property to a valid `Uri`. This code previously passed in the website URL as a string, but the new `WebView2` control requires a `Uri`.

```

private void ListBox_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
    var siteCollection = (ViewModels.SiteCollection)DataContext;

    if (siteCollection.SelectedSite != null)
        browser.Source = new Uri(siteCollection.SelectedSite.Url);
    else
        browser.NavigateToString("<body></body>");
}

```

Depending on which version of Windows a user of your app is running, they may need to install the WebView2 runtime. For more information, see [Get started with WebView2 in WPF apps](#).

## Visual Basic projects

If you're using Visual Basic to code your project, the Upgrade Assistant may contain additional steps, such as migrating the `My` namespace. You should only see these steps added when your project is using these features. With the example app, the code in the `MatchingGame.Logic` uses the `My` namespace to access the registry. This project will have a step related to `My`:

7. Update Visual Basic project
  - a. Update vbproj to support "My." namespace

## Troubleshooting tips

There are several known problems that can occur when using the .NET Upgrade Assistant. In some cases, these are problems with the [try-convert tool](#) that the .NET Upgrade Assistant uses internally.

The tool's [GitHub repository](#) has more troubleshooting tips and known issues.

## See also

- [Upgrade a Windows Forms App to .NET 6](#)
- [Upgrade an ASP.NET MVC App to .NET 6](#)
- [Overview of the .NET Upgrade Assistant](#)
- [.NET Upgrade Assistant GitHub Repository](#)

# Upgrade a Windows Forms App to .NET 6 with the .NET Upgrade Assistant

9/20/2022 • 13 minutes to read • [Edit Online](#)

The [.NET Upgrade Assistant](#) is a command-line tool that can assist with upgrading .NET Framework Windows Forms apps to .NET 6. This article provides:

- A demonstration of how to run the tool against a .NET Framework Windows Forms app
- Troubleshooting tips

For more information on how to install the tool, see [Overview of the .NET Upgrade Assistant](#).

## Demo app

You can use the [Basic Windows Forms Sample](#) project to test upgrading with the Upgrade Assistant.

## Analyze your app

The .NET Upgrade Assistant tool includes an analyze mode that performs a simplified dry run of upgrading your app. It may provide insights as to what changes may be required before the upgrade is started. Open a terminal and navigate to the folder where the target project or solution is located. Run the `upgrade-assistant analyze` command, passing in the name of the project or solution you're upgrading.

For example, running the analyze mode with the [Basic Windows Forms Sample](#) app produces the following output, indicating that there aren't any changes to be made before upgrading:

```
> upgrade-assistant analyze .\MatchingGame.sln

16:18:52 INF] Loaded 5 extensions
[16:18:53 INF] Using MSBuild from C:\Program Files\dotnet\sdk\6.0.200-preview.22055.15\
[16:18:53 INF] Using Visual Studio install from C:\Program Files\Microsoft Visual Studio\2022\Preview [v17]
[16:18:55 INF] Recommending Windows TFM net6.0-windows because the project either has Windows-specific dependencies or builds to a WinExe
[16:18:56 INF] Reference to .NET Upgrade Assistant analyzer package
(Microsoft.DotNet.UpgradeAssistant.Extensions.Default.Analyzers, version 0.3.261602) needs to be added
[16:18:57 INF] Adding Microsoft.Windows.Compatibility 6.0.0
[16:18:57 INF] Reference to .NET Upgrade Assistant analyzer package
(Microsoft.DotNet.UpgradeAssistant.Extensions.Default.Analyzers, version 0.3.261602) needs to be added
[16:18:57 INF] Running analyzers on MatchingGame
[16:18:59 INF] Identified 0 diagnostics in project MatchingGame
[16:18:59 INF] Running analyzers on MatchingGame.Logic
[16:18:59 INF] Identified 0 diagnostics in project MatchingGame.Logic
[16:18:59 WRN] HighDpiMode needs to set in Main() instead of app.config or app.manifest -
Application.SetHighDpiMode(HighDpiMode.<setting>). It is recommended to use SystemAware as the HighDpiMode option for better results.
```

There's quite a bit of internal diagnostic information in the output, but some information is helpful. Notice that the analyze mode indicates that the upgrade will recommend that the project target the `net6.0-windows` target framework moniker ([TFM](#)). This is because the projects referenced by the solution are Windows Forms projects, a Windows-only technology. A console application would probably get the recommendation to upgrade to TFM `net6.0` directly, unless it used some Windows-specific libraries.

If any errors or warnings are reported, take care of them before you start an upgrade.

# Run upgrade-assistant

Open a terminal and navigate to the folder where the target project or solution is located. Run the `upgrade-assistant upgrade` command, passing in the name of the project or solution you're upgrading.

When a project is provided, the upgrade process starts on that project immediately. If a solution is provided, you'll select which project you normally run, known as the [upgrade entrypoint](#). Based on that project, a dependency graph is created and a suggestion as to which order you should upgrade the projects is provided.

```
upgrade-assistant upgrade .\MatchingGame.sln
```

The tool runs and shows you a list of the steps it will do. As each step is completed, the tool provides a set of commands allowing the user to apply or skip the next step or some other option such as:

- Get more information about the step.
- Change projects.
- Adjust logging settings.
- Stop the upgrade and quit.

Pressing Enter without choosing a number selects the first item in the list.

As each step initializes, it may provide information about what it thinks will happen if you apply the step.

## Select the Entrypoint

The first step in upgrading the [example app](#) is choosing which project in the solution serves as the entrypoint project.

Upgrade Steps

1. [Next step] Select an entrypoint
2. Select project to upgrade

Choose a command:

1. Apply next step (Select an entrypoint)
2. Skip next step (Select an entrypoint)
3. See more step details
4. Configure logging
5. Exit

Choose **command 1** to start that step. The results are displayed:

```
[16:32:05 INF] Applying upgrade step Select an entrypoint
Please select the project you run. We will then analyze the dependencies and identify the recommended order
to upgrade projects.
1. MatchingGame
2. MatchingGame.Logic
```

There are two projects listed: The main Windows Forms app (MatchingGame) and the library project (MatchingGame.Logic). Select the MatchingGame project for the entrypoint, which is **item 1**.

## Select a project to upgrade

After the entrypoint is determined, the next step is to choose which project to upgrade first. In this example, the tool determined that the library project (MatchingGame.Logic) should be upgraded first since the main Windows Forms app project depends on it.

```
[16:36:20 INF] Applying upgrade step Select project to upgrade
Here is the recommended order to upgrade. Select enter to follow this list, or input the project you want to
start with.
1. MatchingGame.Logic
2. MatchingGame
```

The recommended path here, is to first upgrade the **MatchingGame.Logic** project first, since the **MatchingGame** project depends on it. It's best to follow the recommended upgrade path.

Upgrading the **MatchingGame.Logic** is a simple project and doesn't present any post-upgrade problems. Therefore, this article continues on as if that project has already been upgraded and the next project to upgrade is **MatchingGame**.

## Upgrade the project

Once a project is selected, a list of upgrade steps the tool will take is listed.

### IMPORTANT

For the purposes of this example, the **MatchingGame** project is described. It's assumed that the **MatchingGame.Logic** project was successfully upgraded. The reason for demonstrating the **MatchingGame** project is that it contains more of the common issues you'll run into when upgrading an app.

Based on the project you're upgrading, you may or may not see every step listed in this example.

The following output describes the steps involved in upgrading the project:

```
[16:38:44 INF] Initializing upgrade step Back up project

Upgrade Steps

Entrypoint: C:\code\Work\temp\Migration\winforms\net45cs\MatchingGame\MatchingGame.csproj
Current Project: C:\code\Work\temp\Migration\winforms\net45cs\MatchingGame\MatchingGame.csproj

1. [Next step] Back up project
2. Convert project file to SDK style
3. Clean up NuGet package references
4. Update TFM
5. Update NuGet Packages
6. Add template files
7. Update Winforms Project
    a. Default Font API Alert
    b. Winforms Source Updater
8. Upgrade app config files
    a. Convert Application Settings
    b. Convert Connection Strings
    c. Disable unsupported configuration sections
9. Update source code
    a. Apply fix for UA0002: Types should be upgraded
    b. Apply fix for UA0012: 'UnsafeDeserialize()' does not exist
10. Move to next project

Choose a command:
1. Apply next step (Back up project)
2. Skip next step (Back up project)
3. See more step details
4. Select different project
5. Configure logging
6. Exit
```

## NOTE

For the rest of this article, the upgrade steps aren't explicitly shown unless there is something important to call out. The results of each step are still shown.

### Create a backup

In this example of upgrading the project, you'll apply each step. The first step, **command 1**, is backing up the project:

```
[16:43:22 INF] Applying upgrade step Back up project
Please choose a backup path
  1. Use default path [C:\code\Work\temp\Migration\winforms\net45cs.backup]
  2. Enter custom path
```

The tool chooses a default backup path named after the current folder, but with `.backup` appended to it. You can choose a custom path as an alternative to the default path. For each upgraded project, the folder of the project is copied to the backup folder. In this example, the `MatchingGame` folder is copied from `net45cs\MatchingGame` to `net45cs.backup\MatchingGame` during the backup step:

```
[16:43:37 INF] Backing up C:\code\Work\temp\Migration\winforms\net45cs\MatchingGame to
C:\code\Work\temp\Migration\winforms\net45cs.backup\MatchingGame
[16:43:37 INF] Project backed up to C:\code\Work\temp\Migration\winforms\net45cs.backup\MatchingGame
[16:43:37 INF] Upgrade step Back up project applied successfully
Please press enter to continue...
```

### Upgrade the project file

The project is upgraded from the .NET Framework project format to the .NET SDK project format.

```
[16:44:31 INF] Applying upgrade step Convert project file to SDK style
[16:44:31 INF] Converting project file format with try-convert, version
0.3.261602+8aa571efd8bac422c95c35df9c7b9567ad534ad0
[16:44:31 INF] Recommending Windows TFM net6.0-windows because the project either has Windows-specific
dependencies or builds to a WinExe
C:\code\Work\temp\Migration\winforms\net45cs\MatchingGame\MatchingGame.csproj contains an App.config file.
App.config is replaced by appsettings.json in .NET Core. You will need to delete App.config and migrate to
appsettings.json if it's applicable to your project.
[16:44:32 INF] Converting project
C:\code\Work\temp\Migration\winforms\net45cs\MatchingGame\MatchingGame.csproj to SDK style
[16:44:32 INF] Project file converted successfully! The project may require additional changes to build
successfully against the new .NET target.
[16:44:33 INF] Upgrade step Convert project file to SDK style applied successfully
Please press enter to continue...
```

Pay attention to the output of each step. Notice that the example output indicates that you'll have a manual step to complete after the upgrade:

App.config is replaced by appsettings.json in .NET Core. You will need to delete App.config and migrate to appsettings.json if it's applicable to your project.

As part of this upgrade step, the NuGet packages referenced by the `packages.config` are migrated to the project file.

### Clean up NuGet references

Once the project format has been updated, the next step is to clean up the NuGet package references.

In addition to the packages referenced by your app, the `packages.config` file contains references to the

dependencies of those packages. For example, if you added reference to package A which depends on package B, both packages would be referenced in the *packages.config* file. In the new project system, only the reference to package A is required. This step analyzes the package references and removes those that aren't required.

```
[16:46:06 INF] Applying upgrade step Clean up NuGet package references
[16:46:06 INF] Removing outdated package reference: MetroFramework.Design, Version=1.2.0.3
[16:46:06 INF] Removing outdated package reference: MetroFramework.Fonts, Version=1.2.0.3
[16:46:06 INF] Removing outdated package reference: MetroFramework.RunTime, Version=1.2.0.3
[16:46:06 INF] Adding package reference: Microsoft.DotNet.UpgradeAssistant.Extensions.Default.Analyzers,
Version=0.3.261602
[16:46:08 INF] Upgrade step Clean up NuGet package references applied successfully
Please press enter to continue...
```

Your app is still referencing .NET Framework assemblies. Some of those assemblies may be available as NuGet packages. This step analyzes those assemblies and references the appropriate NuGet package.

#### Handle the TFM

The tool next changes the **TFM** from .NET Framework to the suggested SDK. In this example, it's `net6.0-windows`.

```
[16:47:16 INF] Applying upgrade step Update TFM
[16:47:16 INF] Recommending Windows TFM net6.0-windows because the project either has Windows-specific
dependencies or builds to a WinExe
[16:47:18 INF] Updated TFM to net6.0-windows
[16:47:18 INF] Upgrade step Update TFM applied successfully
Please press enter to continue...
```

#### Upgrade NuGet packages

Next, the tool updates the project's NuGet packages to the versions that support the updated TFM,

`net6.0-windows`.

```
[16:47:47 INF] Applying upgrade step Update NuGet Packages
[16:47:47 INF] Removing outdated package reference: Microsoft.CSharp, Version=4.7.0
[16:47:47 INF] Removing outdated package reference: System.Data.DataSetExtensions, Version=4.5.0
[16:47:47 INF] Adding package reference: Microsoft.Windows.Compatibility, Version=6.0.0
[16:47:49 INF] Upgrade step Update NuGet Packages applied successfully
Please press enter to continue...
```

The next few steps may be skipped automatically by the tool if the tool determines there isn't anything to do for your project.

#### Windows Forms project specific updates

The first Windows Forms specific upgrade is to actually notify you that the default font in Windows Forms has changed. The default font has changed from **Microsoft Sans Serif, 8.25pt** (.NET Framework) to **Segoe UI, 9pt** (.NET 6 or later). For more information, see [What's new in Windows Forms: Default font](#). Because of this change, you'll want to review all of your forms and user controls. Changing the default font may affect the layout of the controls on your forms.

```
[16:48:45 INF] Applying upgrade step Default Font API Alert
[16:48:45 WRN] Default font in Windows Forms has been changed from Microsoft Sans Serif to Segoe UI, in
order to change the default font use the API - Application.SetDefaultFont(Font font). For more details see
here - https://devblogs.microsoft.com/dotnet/whats-new-in-windows-forms-in-net-6-0-preview-5/#application-
wide-default-font.
[16:48:45 INF] Upgrade step Default Font API Alert applied successfully
Please press enter to continue...
```

The next change for Windows Forms is to update the app startup logic to call the **SetHighDpiMode** method, which sets the DPI mode of the Windows Forms application. Previously this was set in the *app.config* or

*app.manifest* file. This too may affect the layout of the controls on your forms.

```
[16:49:05 INF] Applying upgrade step Winforms Source Updater
[16:49:05 WRN] HighDpiMode needs to set in Main() instead of app.config or app.manifest -
Application.SetHighDpiMode(HighDpiMode.<setting>). It is recommended to use SystemAware as the HighDpiMode
option for better results.
[16:49:05 INF] Updated Program.cs file at
C:\code\Work\temp\Migration\winforms\net45cs\MatchingGame\Program.cs with HighDPISetting set to SystemAware
[16:49:05 INF] Upgrade step Winforms Source Updater applied successfully
[16:49:05 INF] Applying upgrade step Update Winforms Project
[16:49:05 INF] Upgrade step Update Winforms Project applied successfully
Please press enter to continue...
```

### Config and code files

Next, the *app.config* file needs to be migrated. There aren't any connection strings or settings to migrate to the new *appsettings.json* file. If you examine the output of the example project, notice that the **Initializing** entry was displayed three times, each with a different step. If a step has nothing to do, it's skipped, as is the case with the "Upgrade app config files" and "Update source code" steps:

```
[07:35:56 INF] Initializing upgrade step Upgrade app config files
[07:35:56 INF] Found 0 app settings for upgrade:
[07:35:56 INF] Found 0 connection strings for upgrade:
[07:35:56 INF] Initializing upgrade step Update source code
[07:35:56 INF] Running analyzers on MatchingGame
[07:35:57 INF] Identified 0 diagnostics in project MatchingGame
[07:35:57 INF] Initializing upgrade step Move to next project
```

### Completing the upgrade

If there are any more projects to migrate, the tool lets you select which project to upgrade next. When there are no more projects to upgrade, the tool brings you to the "Finalize upgrade" step:

```
1. [Next step] Finalize upgrade
```

Choose a command:

1. Apply next step (Finalize upgrade)
2. Skip next step (Finalize upgrade)
3. See more step details
4. Configure logging
5. Exit

Once the upgrade is complete, the migrated Windows Forms project looks like the following XML:

```

<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <TargetFramework>net6.0-windows</TargetFramework>
    <OutputType>WinExe</OutputType>
    <GenerateAssemblyInfo>false</GenerateAssemblyInfo>
    <UseWindowsForms>true</UseWindowsForms>
    <ImportWindowsDesktopTargets>true</ImportWindowsDesktopTargets>
  </PropertyGroup>
  <ItemGroup>
    <ProjectReference Include="..\MatchingGame.Logic\MatchingGame.Logic.csproj" />
  </ItemGroup>
  <ItemGroup>
    <PackageReference Include="MetroFramework" Version="1.2.0.3" />
    <PackageReference Include="Microsoft.DotNet.UpgradeAssistant.Extensions.Default.Analyzers" Version="0.3.261602">
      <PrivateAssets>all</PrivateAssets>
    </PackageReference>
    <PackageReference Include="Microsoft.Windows.Compatibility" Version="6.0.0" />
  </ItemGroup>
</Project>

```

Notice that the .NET Upgrade Assistant also adds analyzers to the project that assist with continuing the upgrade process, such as the `Microsoft.DotNet.UpgradeAssistant.Extensions.Default.Analyzers` NuGet package.

## Visual Basic projects

If you're using Visual Basic to code your project, the .NET Upgrade Assistant may contain additional steps, such as migrating the `My` namespace. You should only see these steps added when your project is using these features. With the example app, the code in the `MatchingGame.Logic` uses the `My` namespace to access the registry. This project will have a step related to `My`:

7. Update Visual Basic project
  - a. Update vbproj to support "My." namespace

## After upgrading

After you upgrade your projects, you'll need to compile and test them. Most certainly you'll have more work to do in finishing the upgrade. It's possible that the .NET Framework version of your app contained library references that your project isn't actually using. You'll need to analyze each reference and determine whether or not it's required. The tool may have also added or upgraded a NuGet package reference to wrong version.

At the time this article was published, `MatchingGame.Logic` project fails to compile. This project is using the Windows Registry, which isn't provided directly by .NET 6 as it was with .NET Framework. To access the Windows Registry, add the NuGet package `Microsoft.Win32.Registry` to the project.

```
<PackageReference Include="Microsoft.Win32.Registry" Version="5.0.0" />
```

Once that item is fixed, the example app compiles and runs.

## Troubleshooting tips

There are several known problems that can occur when using the .NET Upgrade Assistant. In some cases, these are problems with the [try-convert tool](#) that the .NET Upgrade Assistant uses internally.

The tool's [GitHub repository](#) has more troubleshooting tips and known issues.

## See also

- [Upgrade a WPF App to .NET 6](#)
- [Upgrade an ASP.NET MVC App to .NET 6](#)
- [Overview of the .NET Upgrade Assistant](#)
- [.NET Upgrade Assistant GitHub Repository](#)

# Migrate UWP apps to Windows App SDK with the .NET Upgrade Assistant

9/20/2022 • 20 minutes to read • [Edit Online](#)

The [.NET Upgrade Assistant](#) is a command-line tool that can assist with upgrading .NET Framework UWP apps to .NET 6. Your project will be migrated to the Windows App SDK, and use Windows UI (WinUI) 3. This article provides:

- What to expect
- Things to know before starting
- A demonstration of how to run the tool against a UWP app
- Troubleshooting tips

For more information on how to install the tool, see [Overview of the .NET Upgrade Assistant](#).

## What to expect

This tool migrates your app by:

- Backing up your project.
- Converts your project to the latest SDK format and cleans up your NuGet package references.
- Targets .NET 6 and Windows App SDK.
- Upgrades from WinUI 2 to WinUI 3.
- Adds new template files such as *App.Xaml*, *MainWindow.Xaml* and publish profiles.
- Update namespaces and adds **MainPage** navigation.
- Attempts to detect and fix APIs that have changed, and marks APIs that are no longer supported, with `//TODO` code comments.
- Supports WinRT components and non-standard project structure.
- Produces a post upgrade report.

We aim to provide migration guidance in form of warning messages within the tool and TODO comments within your project as the tool tries to migrate the project. In this way, you'll always be in control of your migration. And for the APIs where complete automation isn't possible, plan is to add `//TODO` comments for the developers to know where the work will be needed. A typical `//TODO` comment will also include a link to our existing migration documentation. Check the Task list within the Visual Studio to see all the action items as TODO comments.

### NOTE

After a successful run of the tool, you may choose to do the following if needed: move your code from *App.xaml.old.cs* to *App.xaml.cs* and move *AssemblyInfo.cs* from backup

## Things to know before starting

This tool currently supports C#, and in most cases the app will require more effort to complete the migration. The goal of the tool is to convert your project and code, so that it can compile. Some features require you to investigate and fix, and have `//TODO` code comments. For more information about what to consider before migrating, see [What is supported when migrating from UWP to WinUI 3](#).

Additionally, you may choose to wait for the next version of .NET Upgrade Assistant tool before you start migrating your app, because of the current limitations of the tool:

- Custom views aren't supported.

For example, you won't receive a warning or a fix for a `CustomDialog` that extends `MessageDialog` and calls an API incorrectly.

- Multi window apps might not convert correctly.

This release is currently in preview, and is receiving frequent updates. If you discover problems using the tool, report them in the tool's [GitHub repository](#). Use the **UWP** area tag so that all UWP related issues can be redirected to us.

#### TIP

You can also refer to the [UWP migration guide](#), which tells you more about the changes from UWP APIs to the new Windows App SDK-supported APIs and capabilities.

## A demonstration of how to run the tool against a UWP app

You can use the [PhotoLab UWP Sample app](#) project to test upgrading with the Upgrade Assistant.

#### NOTE

You may also want to see the manual migration of PhotoLab sample app as a case study documented [here](#).

For the demo, we have a UWP sample app called "PhotoLab", which is an app for viewing and editing image files, demonstrating XAML layout, data binding, and UI customization features. The app will require more effort to complete the migration. This should be familiar if you've used .NET Upgrade Assistant in the past to migrate a WPF or WinForms app from .NET Framework to .NET 6. Now, run it against the PhotoLab app, which is .NET Native UWP project, and follow the steps one-by-one.

## Analyze your app

#### NOTE

At the time of publishing this document, Analyze command is not working as it should. We are working to resolve the same. You may use the upgrade command without skipping the backup step.

The .NET Upgrade Assistant tool includes an analyze mode that performs a simplified dry run of upgrading your app. It may provide insights as to what changes may be required before the upgrade is started. Open a terminal and navigate to the folder where the target project or solution is located. Run the `upgrade-assistant analyze` command, passing in the name of the project or solution you're upgrading.

For example, running the analyze mode with the [PhotoLab UWP Sample app](#) produces the following output, indicating that there aren't any changes to be made before upgrading:

```
> upgrade-assistant analyze .\PhotoLab.sln

[23:08:38 INF] Loaded 7 extensions
[23:08:53 INF] Using MSBuild from C:\Program Files\dotnet\sdk\6.0.200\
[23:08:53 INF] Using Visual Studio install from C:\Program Files\Microsoft Visual Studio\2022\Preview [v17]
[23:09:04 INF] Writing output to .\AnalysisReport.sarif
[23:09:04 INF] Skip minimum dependency check because Windows App SDK cannot work with targets lower than
already recommended TFM.
[23:09:04 INF] Recommending Windows TFM net6.0-windows because the project either has Windows-specific
dependencies or builds to a WinExe
[23:09:04 INF] Marking package Microsoft.NETCore.UniversalWindowsPlatform for removal based on package
mapping configuration UWP
[23:09:04 INF] Adding package Microsoft.WindowsAppSDK based on package mapping configuration UWP
[23:09:04 INF] Adding package CommunityToolkit.WinUI.UI.Animations based on package mapping configuration
UWP
[23:09:04 INF] Adding package Microsoft.Graphics.Win2D based on package mapping configuration UWP
[23:09:04 INF] Marking package Microsoft.Toolkit.Uwp.UI.Animations for removal based on package mapping
configuration UWP
[23:09:04 INF] Marking package Microsoft.UI.Xaml for removal based on package mapping configuration UWP
[23:09:06 WRN] No version of Microsoft.Toolkit.Uwp.UI.Animations found that supports ["net6.0-windows"];
leaving unchanged
[23:09:07 INF] Package Microsoft.UI.Xaml, Version=2.4.2 does not support the target(s) net6.0-windows but a
newer version (2.7.1) does.
[23:09:09 INF] Package Microsoft.WindowsAppSDK, Version=1.0.0 does not support the target(s) net6.0-windows
but a newer version (1.0.3) does.
[23:09:10 WRN] No version of CommunityToolkit.WinUI.UI.Animations found that supports ["net6.0-windows"];
leaving unchanged
[23:09:11 INF] Reference to .NET Upgrade Assistant analyzer package
(Microsoft.DotNet.UpgradeAssistant.Extensions.Default.Analyzers, version 0.3.326103) needs to be added
[23:09:13 INF] Adding Microsoft.Windows.Compatibility 6.0.0 helps with speeding up the upgrade process for
Windows-based APIs
[23:09:14 WRN] Unable to find a supported WinUI nuget package for Microsoft.Toolkit.Uwp.UI.Animations.
Skipping this package.
[23:09:16 INF] Running analyzers on PhotoLab
[23:09:25 INF] Identified 7 diagnostics in project PhotoLab
[23:09:25 INF] Diagnostic UA307 with the message Detect UWP back button generated
[23:09:25 INF] Diagnostic UA307 with the message Detect UWP back button generated
[23:09:25 INF] Diagnostic UA309 with the message Detect content dialog api generated
[23:09:25 INF] Diagnostic UA307 with the message Detect UWP back button generated
[23:09:25 INF] Diagnostic UA307 with the message Detect UWP back button generated
[23:09:25 INF] Diagnostic UA309 with the message Detect content dialog api generated
[23:09:25 INF] Diagnostic UA310 with the message Tries to detect the creation of known classes that
implement IInitializeWithWindow generated
[23:09:25 INF] Analysis Complete, the report is available at .\AnalysisReport.sarif
```

There's quite a bit of internal diagnostic information in the output, but some information is helpful. Notice that the analyze mode indicates that the upgrade will recommend that the project target the `net6.0-windows` target framework moniker ([TFM](#)). A console application would probably get the recommendation to upgrade to TFM `net6.0` directly, unless it used some Windows-specific libraries.

If any errors or warnings are reported, take care of them before you start an upgrade.

## Run upgrade-assistant

Open a terminal and navigate to the folder where the target project or solution is located. Run the `upgrade-assistant upgrade` command, passing in the name of the project or solution you're upgrading.

When a project is provided, the upgrade process starts on that project immediately. If a solution is provided, you'll select which project you normally run, known as the upgrade entrypoint. Based on that project, a dependency graph is created and a suggestion as to which order you should upgrade the projects is provided.

```
upgrade-assistant upgrade .\PhotoLab.sln
```

The tool runs and shows you a list of the steps it will do. As each step is completed, the tool provides a set of commands allowing the user to apply or skip the next step or some other option such as:

- Get more information about the step.
- Change projects.
- Adjust logging settings.
- Stop the upgrade and quit.

Pressing Enter without choosing a number selects the first item in the list.

As each step initializes, it may provide information about what it thinks will happen if you apply the step.

### Select the entrypoint and project to upgrade

The first step in upgrading the [PhotoLab UWP Sample app](#) is choosing which project in the solution serves as the entrypoint project. You may notice that there's only one entrypoint, this is because there's only one project in this solution.

```
[23:29:49 INF] Loaded 7 extensions
[23:29:52 INF] Using MSBuild from C:\Program Files\dotnet\sdk\6.0.200\
[23:29:52 INF] Using Visual Studio install from C:\Program Files\Microsoft Visual Studio\2022\Preview [v17]
[23:30:01 INF] Initializing upgrade step Select an entrypoint
[23:30:01 INF] Setting entrypoint to only project in solution: .\source\repos\Windows-appsample-photo-
lab\PhotoLab\PhotoLab.csproj
[23:30:01 INF] Initializing upgrade step Select project to upgrade
[23:30:04 INF] Initializing upgrade step Back up project
Upgrade Steps
```

After the entrypoint is determined, the next step is to choose which project to upgrade first. However, in this example, the tool determined that there's only one project within the solution, and it should begin upgrading that project.

### Upgrade the project

Once a project is selected, a list of upgrade steps the tool will take is listed.

#### IMPORTANT

Based on the project you're upgrading, you may or may not see every step listed in this example.

The following output describes the steps involved in upgrading the project:

```
[23:30:04 INF] Initializing upgrade step Back up project
Upgrade Steps

Entrypoint: .\source\repos\Windows-appsample-photo-lab\PhotoLab\PhotoLab.csproj
Current Project: .\source\repos\Windows-appsample-photo-lab\PhotoLab\PhotoLab.csproj

1. [Next step] Back up project
2. Convert project file to SDK style
3. Clean up NuGet package references
   a. Duplicate reference analyzer
   b. Package map reference analyzer
   c. Target compatibility reference analyzer
   d. Upgrade assistant reference analyzer
      e. Windows Compatibility Pack Analyzer
```

- e. Windows Compatibility Pack Analyzer
  - f. MyDotAnalyzer reference analyzer
  - g. Newtonsoft.Json reference analyzer
  - h. Windows App SDK package analysis
  - i. Transitive reference analyzer
4. Update TFM
5. Update NuGet Packages
- a. Duplicate reference analyzer
  - b. Package map reference analyzer
  - c. Target compatibility reference analyzer
  - d. Upgrade assistant reference analyzer
  - e. Windows Compatibility Pack Analyzer
  - f. MyDotAnalyzer reference analyzer
  - g. Newtonsoft.Json reference analyzer
  - h. Windows App SDK package analysis
  - i. Transitive reference analyzer
6. Add template files
7. Update Windows Desktop Project
- a. Update WinUI namespaces
  - b. Update WinUI Project Properties
  - c. Update package.appxmanifest
  - d. Remove unnecessary files
  - e. Update animations xaml
  - f. Insert back button in XAML
8. Update source code
- a. Apply fix for UA0002: Types should be upgraded
  - b. Apply fix for UA0012: 'UnsafeDeserialize()' does not exist
  - c. Apply fix for UA0014: .NET MAUI projects should not reference Xamarin.Forms namespaces
  - d. Apply fix for UA0015: .NET MAUI projects should not reference Xamarin.Essentials namespaces
  - e. Apply fix for [UA306_A1, UA306_A2, UA306_A3, UA306_A4, UA306_B, UA306_C, UA306_D, UA306_E, UA306_F, UA306_G, UA306_H, UA306_I]: Replace usage of Windows.UI.Core.CoreDispatcher, Replace usage of Window.Current.Dispatcher, Replace usage of App.Window.Dispatcher, Replace usage of Window.Dispatcher, Replace usage of Windows.Media.Capture.CameraCaptureUI, Replace usage of Microsoft.UI.Xaml.Controls.InkCanvas, Replace usage of Microsoft.UI.Xaml.Controls.Maps.MapControl, Replace usage of Microsoft.UI.Xaml.Controls.MediaElement, Replace usage of Windows.Graphics.Printing.PrintManager, Replace usage of Windows.Security.Authentication.Web.WebAuthenticationBroker, Replace usage of Windows.UI.Xaml.Media.AcrylicBrush.BackgroundSource, Replace usage of Windows.UI.Shell.TaskbarManager
  - f. Apply fix for UA307: Custom back button implementation is needed
  - g. Apply fix for UA309: ContentDialog API needs to set XamlRoot
  - h. Apply fix for UA310: Classes that implement IInitializeWithWindow need to be initialized with WindowHandle
  - i. Apply fix for UA311: Classes that implement IDataTransferManager should use IDataTransferManagerInterop.ShowShareUIForWindow
  - j. Apply fix for UA312: Interop APIs should use the window handle
  - k. Apply fix for [UA313, UA314]: MRT to MRT core migration, MRT to MRT core migration
  - l. Apply fix for [UA315_A, UA315_C, UA315_B]: Windows App SDK apps should use Microsoft.UI.Windowing.AppWindow, Windows App SDK apps should use Microsoft.UI.Windowing.AppWindow
9. Move to next project

Choose a command:

1. Apply next step (Back up project)
2. Skip next step (Back up project)
3. See more step details
4. Configure logging
5. Exit

### **Upgrade the project file**

The project is upgraded from the .NET Framework project format to the .NET SDK project format.

```
[02:14:51 INF] Applying upgrade step Convert project file to SDK style
[02:14:52 INF] Converting project file format with try-convert, version 0.9.0-dev
[02:14:54 INF] Skip minimum dependency check because Windows App SDK cannot work with targets lower than
already recommended TFM.
[02:14:54 INF] Recommending Windows TFM net6.0-windows because the project either has Windows-specific
dependencies or builds to a WinExe
[02:14:56 INF] Converting project .\source\repos\Windows-appsample-photo-lab\PhotoLab\PhotoLab.csproj to SDK
style
[02:14:58 INF] Project file converted successfully! The project may require additional changes to build
successfully against the new .NET target.
[02:15:01 INF] Upgrade step Convert project file to SDK style applied successfully
```

Pay attention to the output of each step. The tool will indicate a message and you may need to make changes manually from this step onwards.

#### Clean up NuGet references

Once the project format has been updated, the next step is to clean up the NuGet package references.

In addition to the packages referenced by your app, the *packages.config* file contains references to the dependencies of those packages. For example, if you added reference to package A which depends on package B, both packages would be referenced in the *packages.config* file. In the new project system, only the reference to package A is required. This step analyzes the package references and removes those that aren't required.

```
[02:18:32 INF] Initializing upgrade step Clean up NuGet package references
[02:18:32 INF] Initializing upgrade step Duplicate reference analyzer
[02:18:32 INF] No package updates needed
[02:18:32 INF] Initializing upgrade step Package map reference analyzer
[02:18:32 INF] Marking package Microsoft.NETCore.UniversalWindowsPlatform for removal based on package
mapping configuration UWP
[02:18:32 INF] Adding package Microsoft.WindowsAppSDK based on package mapping configuration UWP
[02:18:32 INF] Adding package CommunityToolkit.WinUI.UI.Animations based on package mapping configuration
UWP
[02:18:32 INF] Adding package Microsoft.Graphics.Win2D based on package mapping configuration UWP
[02:18:32 INF] Marking package Microsoft.Toolkit.Uwp.UI.Animations for removal based on package mapping
configuration UWP
[02:18:32 INF] Marking package Microsoft.UI.Xaml for removal based on package mapping configuration UWP

[02:19:04 INF] Applying upgrade step Remove package 'Microsoft.NETCore.UniversalWindowsPlatform'
[02:19:04 INF] Removing outdated package reference: Microsoft.NETCore.UniversalWindowsPlatform,
Version=5.3.3
[02:19:04 INF] Upgrade step Remove package 'Microsoft.NETCore.UniversalWindowsPlatform' applied successfully

[02:20:34 INF] Applying upgrade step Remove package 'Microsoft.Toolkit.Uwp.UI.Animations'
[02:20:34 INF] Removing outdated package reference: Microsoft.Toolkit.Uwp.UI.Animations, Version=1.5.1
[02:20:34 INF] Upgrade step Remove package 'Microsoft.Toolkit.Uwp.UI.Animations' applied successfully

[02:21:38 INF] Removing outdated package reference: Microsoft.UI.Xaml, Version=2.4.2
[02:21:38 INF] Upgrade step Remove package 'Microsoft.UI.Xaml' applied successfully

[02:22:13 INF] Adding package reference: Microsoft.WindowsAppSDK, Version=1.0.0
[02:22:13 INF] Upgrade step Add package 'Microsoft.WindowsAppSDK' applied successfully

[02:22:13 INF] Adding package reference: Microsoft.WindowsAppSDK, Version=1.0.0
[02:22:13 INF] Upgrade step Add package 'Microsoft.WindowsAppSDK' applied successfully
Please press enter to continue...

[02:23:04 INF] Adding package reference: CommunityToolkit.WinUI.UI.Animations, Version=7.1.2
[02:23:04 INF] Upgrade step Add package 'CommunityToolkit.WinUI.UI.Animations' applied successfully

[02:23:42 INF] Adding package reference: Microsoft.Graphics.Win2D, Version=1.0.0.30
[02:23:42 INF] Upgrade step Add package 'Microsoft.Graphics.Win2D' applied successfully

[02:23:42 INF] Applying upgrade step Package map reference analyzer
[02:23:42 INF] Upgrade step Package map reference analyzer applied successfully
```

```
[02:23:42 INF] Upgrade step Package map reference analyzer applied successfully

[02:24:22 INF] Initializing upgrade step Target compatibility reference analyzer
[02:24:22 INF] No package updates needed
[02:24:22 INF] Initializing upgrade step Upgrade assistant reference analyzer
[02:24:23 INF] Reference to .NET Upgrade Assistant analyzer package
(Microsoft.DotNet.UpgradeAssistant.Extensions.Default.Analyzers, version 0.3.326103) needs to be added
[02:24:23 INF] Initializing upgrade step Add package
'Microsoft.DotNet.UpgradeAssistant.Extensions.Default.Analyzers'

[02:28:38 INF] Applying upgrade step Add package
'Microsoft.DotNet.UpgradeAssistant.Extensions.Default.Analyzers'
[02:28:38 INF] Adding package reference: Microsoft.DotNet.UpgradeAssistant.Extensions.Default.Analyzers,
Version=0.3.326103
[02:28:38 INF] Upgrade step Add package 'Microsoft.DotNet.UpgradeAssistant.Extensions.Default.Analyzers'
applied successfully
[02:28:38 INF] Applying upgrade step Upgrade assistant reference analyzer
[02:28:38 INF] Upgrade step Upgrade assistant reference analyzer applied successfully
[02:28:52 INF] Applying upgrade step Add package 'Microsoft.Windows.Compatibility'
[02:28:52 INF] Adding package reference: Microsoft.Windows.Compatibility, Version=6.0.0
[02:28:52 INF] Upgrade step Add package 'Microsoft.Windows.Compatibility' applied successfully
[02:28:52 INF] Applying upgrade step Windows Compatibility Pack Analyzer
[02:28:52 INF] Upgrade step Windows Compatibility Pack Analyzer applied successfully
[02:29:06 INF] Initializing upgrade step MyDotAnalyzer reference analyzer
[02:29:06 INF] No package updates needed
[02:29:06 INF] Initializing upgrade step Newtonsoft.Json reference analyzer
[02:29:06 INF] No package updates needed
[02:29:06 INF] Initializing upgrade step Windows App SDK package analysis
[02:29:06 INF] No package updates needed
[02:29:06 INF] Initializing upgrade step Transitive reference analyzer
[02:29:06 INF] No package updates needed
[02:29:06 INF] Applying upgrade step Clean up NuGet package references
[02:29:06 INF] Upgrade step Clean up NuGet package references applied successfully
```

Your app is still referencing .NET Framework assemblies. Some of those assemblies may be available as NuGet packages. This step analyzes those assemblies and references the appropriate NuGet package.

In this example, a user may also see app specific reference such as `Microsoft.Toolkit.Uwp.UI.Animations`, `CommunityToolkit.WinUI.UI.Animations` and `Microsoft.Graphics.Win2D`.

#### Handle the TFM

The tool next changes the [TFM](#) from .NET Framework to the suggested SDK. In this example, it's `net6.0-windows`.

```
[02:29:06 INF] Initializing upgrade step Update TFM
[02:29:06 INF] Skip minimum dependency check because Windows App SDK cannot work with targets lower than
already recommended TFM.
[02:29:06 INF] Recommending Windows TFM net6.0-windows10.0.19041 because the project either has Windows-
specific dependencies or builds to a WinExe
```

#### Upgrade NuGet packages

Next, the tool updates the project's NuGet packages to the versions that support the updated TFM, `net6.0-windows`.

```
[02:29:06 INF] Initializing upgrade step Update NuGet Packages
[02:29:06 INF] Initializing upgrade step Duplicate reference analyzer
[02:29:06 INF] No package updates needed
[02:29:06 INF] Initializing upgrade step Package map reference analyzer
[02:29:06 INF] No package updates needed
[02:29:06 INF] Initializing upgrade step Target compatibility reference analyzer
[02:29:06 INF] No package updates needed
[02:29:06 INF] Initializing upgrade step Upgrade assistant reference analyzer
[02:29:06 INF] No package updates needed
[02:29:06 INF] Initializing upgrade step Windows Compatibility Pack Analyzer
[02:29:07 INF] No package updates needed
[02:29:07 INF] Initializing upgrade step MyDotAnalyzer reference analyzer
[02:29:07 INF] No package updates needed
[02:29:07 INF] Initializing upgrade step Newtonsoft.Json reference analyzer
[02:29:07 INF] No package updates needed
[02:29:07 INF] Initializing upgrade step Windows App SDK package analysis
[02:29:07 INF] No package updates needed
[02:29:07 INF] Initializing upgrade step Transitive reference analyzer
[02:29:07 INF] No package updates needed
[02:29:07 INF] Applying upgrade step Update NuGet Packages
[02:29:07 INF] Upgrade step Update NuGet Packages applied successfully
```

#### Templates, config, and code files

The next few steps may be skipped automatically by the tool if the tool determines there isn't anything to do for your project.

Once the packages are updated, the next step is to update any template files. In this example, the tool automatically adds necessary publish profiles, *App.xaml.cs*, *MainWindow.xaml.cs*, *MainWindow.xaml* etc.

```
[02:32:44 INF] Applying upgrade step Add template files
[02:32:44 INF] Added template file app.manifest
[02:32:44 INF] Added template file Properties\launchSettings.json
[02:32:44 INF] Added template file Properties\PublishProfiles\win10-arm64.pubxml
[02:32:44 INF] Added template file Properties\PublishProfiles\win10-x64.pubxml
[02:32:44 INF] Added template file Properties\PublishProfiles\win10-x86.pubxml
[02:32:44 INF] File already exists, moving App.xaml.cs to App.xaml.old.cs
[02:32:44 INF] Added template file App.xaml.cs
[02:32:44 INF] Added template file MainWindow.xaml.cs
[02:32:44 INF] Added template file MainWindow.xaml
[02:32:44 INF] Added template file UWPToWinAppSDKUpgradeHelpers.cs
[02:32:44 INF] 9 template items added
[02:32:44 INF] Upgrade step Add template files applied successfully
```

#### UWP specific changes

The next step for the tool is to update the UWP app to the new Windows Desktop Project.

#### IMPORTANT

You may choose to skip the step for back button insertion as per your wish. Inserting back button may cause the UI to behave differently. If this happens, remove the stack panel that is inserted as a parent of the back button and reposition the back button where it seems .

```
[02:36:53 INF] Applying upgrade step Update WinUI namespaces
[02:36:53 INF] Upgrade step Update WinUI namespaces applied successfully

[02:38:45 INF] Applying upgrade step Update WinUI Project Properties
[02:38:46 INF] Upgrade step Update WinUI Project Properties applied successfully

[02:39:11 INF] Applying upgrade step Update package.appxmanifest
[02:39:11 INF] Upgrade step Update package.appxmanifest applied successfully

[02:39:11 INF] Applying upgrade step Update package.appxmanifest
[02:39:11 INF] Upgrade step Update package.appxmanifest applied successfully

[02:39:37 INF] Applying upgrade step Remove unnecessary files
[02:39:37 INF] Deleting .\source\repos\Windows-appsample-photo-lab\PhotoLab\Properties\AssemblyInfo.cs as it
is not required for Windows App SDK projects.
[02:39:37 INF] Upgrade step Remove unnecessary files applied successfully

[02:40:22 INF] Applying upgrade step Update animations xaml
[02:40:22 INF] Upgrade step Update animations xaml applied successfully

[02:40:42 INF] Applying upgrade step Insert back button in XAML
[02:40:42 INF] Upgrade step Insert back button in XAML applied successfully
[02:40:42 INF] Applying upgrade step Update Windows Desktop Project
[02:40:42 INF] Upgrade step Update Windows Desktop Project applied successfully
```

### Updating the source code

In this step, the tool will try to migrate your code and perform source specific code changes.

Code migration for the PhotoLab sample app includes:

- Changes to Content Dialog and File Save picker APIs.
- Xaml update for Animations package.
- Showing warning messages and adding TODO comments in *DetailPage.xaml* and *DetailPage.xaml.cs* and *MainPage.xaml.cs* for custom back button.
- Implementing the back button functionality and adding a TODO comment to customize XAML button.
- A documentation link can be accessed from the CLI tool to study more about for back button implementation.

```
[02:41:34 INF] Applying upgrade step Apply fix for UA307: Custom back button implementation is needed
[02:41:34 WRN] .\source\repos\Windows-appsample-photo-lab\PhotoLab\ MainPage.xaml.cs
    TODO UA307 Default back button in the title bar does not exist in WinUI3 apps.
    The tool has generated a custom back button "UAGeneratedBackButton" in the XAML file.
    Feel free to edit its position, behavior and use the custom back button instead.
    Read: https://aka.ms/UA-back-button
[02:41:34 INF] Diagnostic UA307 fixed in .\source\repos\Windows-appsample-photo-
lab\PhotoLab\ MainPage.xaml.cs
[02:41:34 WRN] .\source\repos\Windows-appsample-photo-lab\PhotoLab\ DetailPage.xaml.cs
    TODO UA307 Default back button in the title bar does not exist in WinUI3 apps.
    The tool has generated a custom back button "UAGeneratedBackButton" in the XAML file.
    Feel free to edit its position, behavior and use the custom back button instead.
    Read: https://aka.ms/UA-back-button
[02:41:34 INF] Diagnostic UA307 fixed in .\source\repos\Windows-appsample-photo-
lab\PhotoLab\ DetailPage.xaml.cs
[02:41:34 INF] Running analyzers on PhotoLab
[02:41:37 INF] Identified 4 diagnostics in project PhotoLab
[02:41:37 WRN] .\source\repos\Windows-appsample-photo-lab\PhotoLab\ DetailPage.xaml.cs
    TODO UA307 Default back button in the title bar does not exist in WinUI3 apps.
    The tool has generated a custom back button "UAGeneratedBackButton" in the XAML file.
    Feel free to edit its position, behavior and use the custom back button instead.
    Read: https://aka.ms/UA-back-button
[02:41:37 INF] Diagnostic UA307 fixed in .\source\repos\Windows-appsample-photo-
lab\PhotoLab\ DetailPage.xaml.cs
[02:41:37 INF] Running analyzers on PhotoLab
[02:41:39 INF] Identified 3 diagnostics in project PhotoLab
[02:41:39 INF] Upgrade step Apply fix for UA307: Custom back button implementation is needed applied
successfully

[02:45:20 INF] Applying upgrade step Apply fix for UA309: ContentDialog API needs to set XamlRoot
[02:45:20 INF] Diagnostic UA309 fixed in .\source\repos\Windows-appsample-photo-
lab\PhotoLab\ DetailPage.xaml.cs
[02:45:20 INF] Diagnostic UA309 fixed in .\source\repos\Windows-appsample-photo-
lab\PhotoLab\ MainPage.xaml.cs
[02:45:20 INF] Running analyzers on PhotoLab
[02:45:23 INF] Identified 1 diagnostics in project PhotoLab
[02:45:23 INF] Upgrade step Apply fix for UA309: ContentDialog API needs to set XamlRoot applied
successfully

[02:45:52 INF] Applying upgrade step Apply fix for UA310: Classes that implement IInitializeWithWindow need
to be initialized with Window Handle
[02:45:52 INF] Diagnostic UA310 fixed in ..\source\repos\Windows-appsample-photo-
lab\PhotoLab\ DetailPage.xaml.cs
[02:45:52 INF] Running analyzers on PhotoLab
[02:45:54 INF] Identified 0 diagnostics in project PhotoLab
[02:45:54 INF] Applying upgrade step Update source code
[02:46:00 INF] Upgrade step Update source code applied successfully
[02:46:00 INF] Upgrade step Apply fix for UA310: Classes that implement IInitializeWithWindow need to be
initialized with Window Handle applied successfully
```

### Completing the upgrade

If there are any more projects to migrate, the tool lets you select which project to upgrade next. When there are no more projects to upgrade, the tool brings you to the "Finalize upgrade" step:

```

1. [Next step] Finalize upgrade

Choose a command:
1. Apply next step (Finalize upgrade)
2. Skip next step (Finalize upgrade)
3. See more step details
4. Configure logging
5. Exit
>
[02:47:13 INF] Applying upgrade step Finalize upgrade
[02:47:13 INF] Upgrade step Finalize upgrade applied successfully

```

Ideally, after successfully running the tool, the user should be able to F5 and run their new WinUI3 desktop project of PhotoLab app. Once the upgrade is complete, the migrated UWP project looks like the following XML:

```

<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <TargetFramework>net6.0-windows10.0.19041.0</TargetFramework>
    <Platform Condition=" '$(Platform)' == 'x86' >x86</Platform>
    <OutputType>WinExe</OutputType>
    <DefaultLanguage>en-US</DefaultLanguage>
    <TargetPlatformMinVersion>10.0.17763.0</TargetPlatformMinVersion>
    <RuntimeIdentifiers>win10-x86;win10-x64;win10-arm64</RuntimeIdentifiers>
    <UseWinUI>true</UseWinUI>
    <ApplicationManifest>app.manifest</ApplicationManifest>
    <EnablePreviewMsixTooling>true</EnablePreviewMsixTooling>
    <Platforms>x86;x64;arm64</Platforms>
    <PublishProfile>win10-$(Platform).pubxml</PublishProfile>
  </PropertyGroup>
  <ItemGroup>
    <AppxManifest Include="Package.appxmanifest">
      <SubType>Designer</SubType>
    </AppxManifest>
  </ItemGroup>
  <ItemGroup>
    <PackageReference Include="Microsoft.WindowsAppSDK" Version="1.0.0" />
    <PackageReference Include="CommunityToolkit.WinUI.UI.Animations" Version="7.1.2" />
    <PackageReference Include="Microsoft.Graphics.Win2D" Version="1.0.0.30" />
    <PackageReference Include="Microsoft.DotNet.UpgradeAssistant.Extensions.Default.Analyzers" Version="0.3.326103" />
      <PrivateAssets>all</PrivateAssets>
    </PackageReference>
    <PackageReference Include="Microsoft.Windows.Compatibility" Version="6.0.0" />
  </ItemGroup>
  <ItemGroup>
    <Compile Remove="App.xaml.old.cs" />
  </ItemGroup>
  <ItemGroup>
    <None Include="App.xaml.old.cs" />
  </ItemGroup>
</Project>

```

Notice that the .NET Upgrade Assistant also adds analyzers to the project that assist with continuing the upgrade process, such as the `Microsoft.DotNet.UpgradeAssistant.Extensions.Default.Analyzers` NuGet package.

Also notice that it's using Windows App SDK, WinUI3, and .NET 6. And now, you can take advantage of all new features that modern apps have to offer and grow your app with the platform.

## After upgrading

After you upgrade your projects, you'll need to compile and test them. Most certainly you'll have more work to do in finishing the upgrade. All TODO comments and action items can be seen on the Task List inside the Visual

Studio. To open the Task List, press **View > TaskList**. It's possible that the .NET Framework version of your app contained library references that your project isn't actually using. You'll need to analyze each reference and determine whether or not it's required. The tool may have also added or upgraded a NuGet package reference to wrong version.

## Troubleshooting tips

There are several known problems that can occur when using the .NET Upgrade Assistant. In some cases, these problems are with the [try-convert tool](#) that the .NET Upgrade Assistant uses internally.

The tool's [GitHub repository](#) has more troubleshooting tips and known issues.

# Upgrade an ASP.NET MVC app to .NET 6 with the .NET Upgrade Assistant

9/20/2022 • 4 minutes to read • [Edit Online](#)

The [.NET Upgrade Assistant](#) is a command-line tool that can assist with upgrading .NET Framework ASP.NET MVC apps to .NET 6. This article provides:

- A demonstration of how to run the tool against a .NET Framework ASP.NET MVC app
- Troubleshooting tips

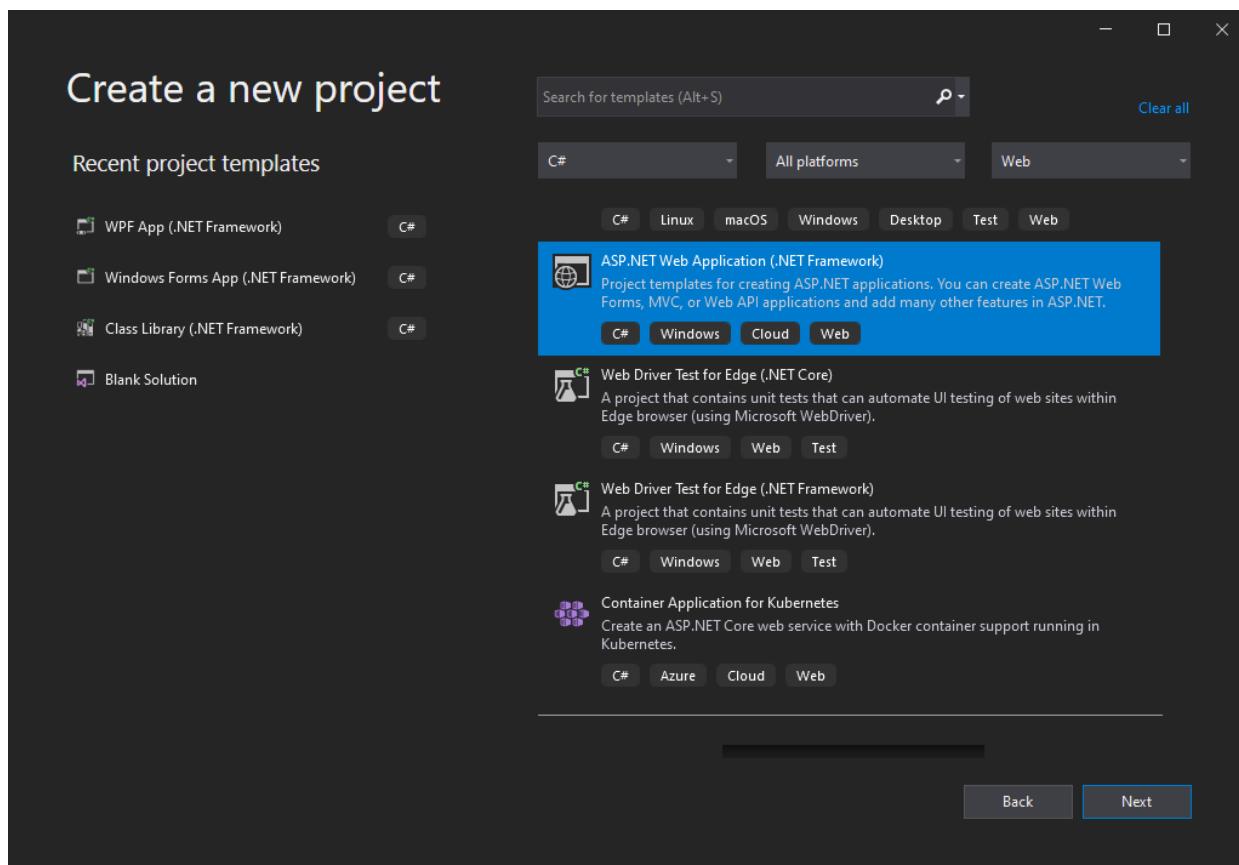
## Upgrade .NET Framework ASP.NET MVC apps

This section demonstrates running the .NET Upgrade Assistant against a newly created ASP.NET MVC app targeting .NET Framework 4.6.1. For more information on how to install the tool, see [Overview of the .NET Upgrade Assistant](#).

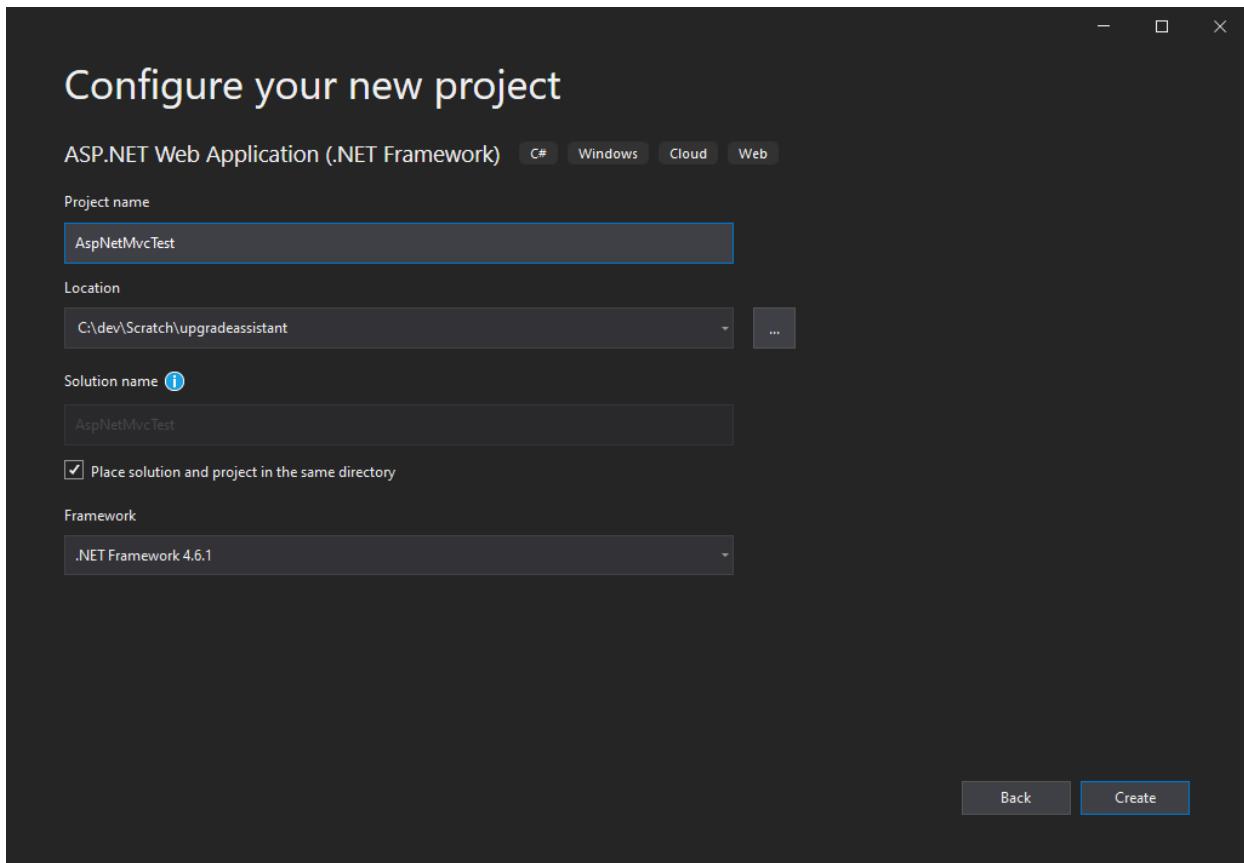
### Initial demo setup

If you're running the .NET Upgrade Assistant against your own .NET Framework app, you can skip this step. If you just want to try it out to see how it works, this step shows you how to set up a sample ASP.NET MVC and Web API (.NET Framework) project to use.

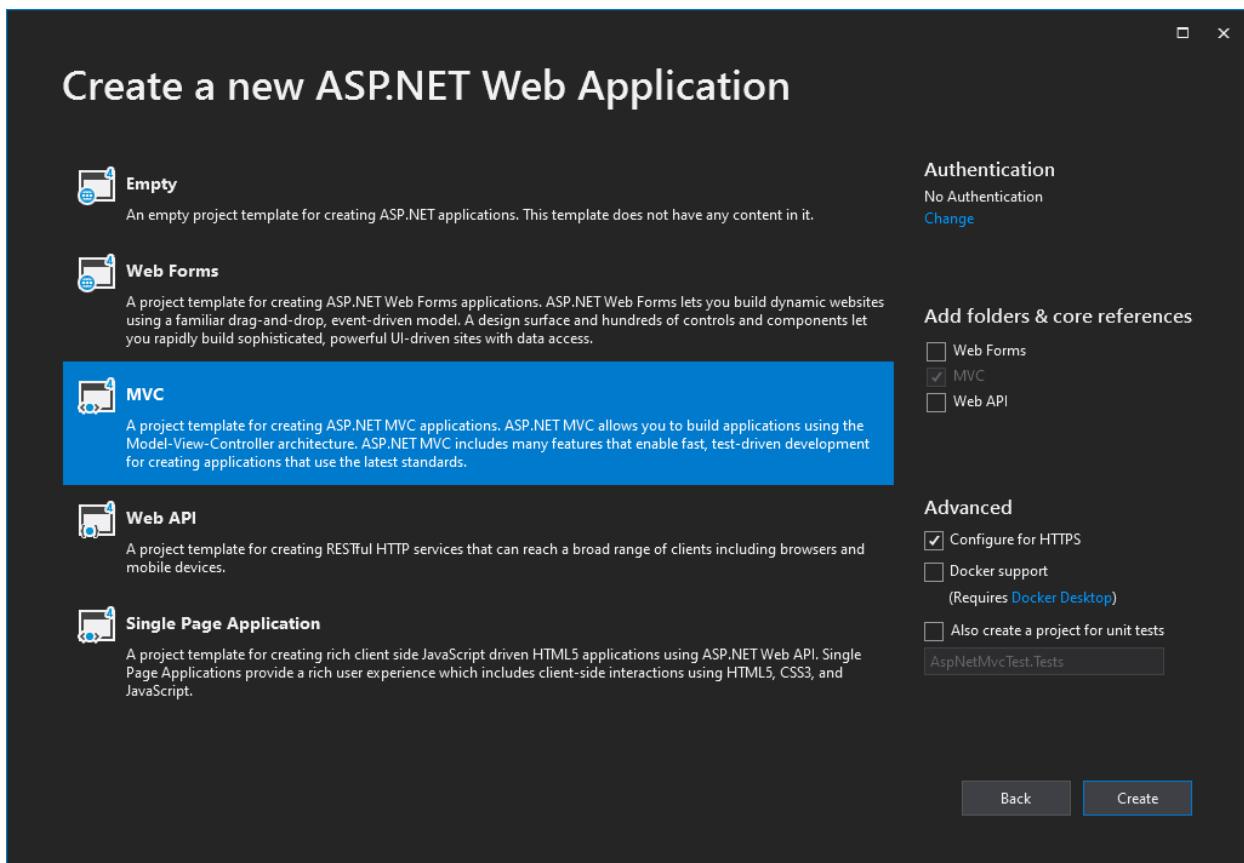
Using Visual Studio, create a new ASP.NET Web Application project using .NET Framework.



Name the project **AspNetMvcTest**. Configure the project to use **.NET Framework 4.6.1**.



In the next dialog, choose **MVC** application, then select **Create**.



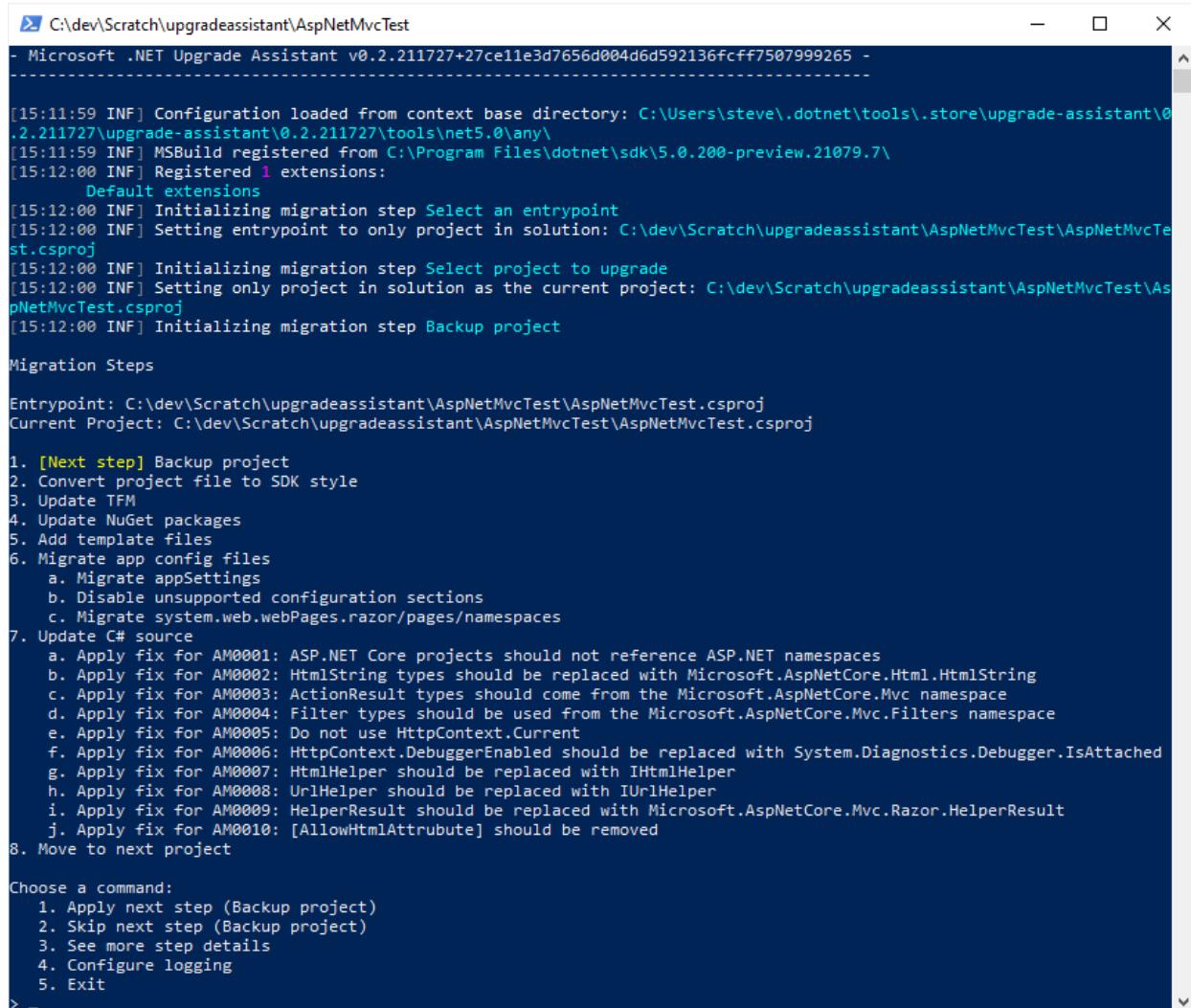
Review the created project and its files, especially its project file(s).

### Run upgrade-assistant

Open a terminal and navigate to the folder where the target project or solution is located. Run the `upgrade-assistant` command, passing in the name of the project you're targeting (you can run the command from anywhere, as long as the path to the project file is valid).

```
upgrade-assistant upgrade .\AspNetMvcTest.csproj
```

The tool runs and shows you a list of the steps it will do.



```
C:\dev\Scratch\upgradeassistant\AspNetMvcTest
- Microsoft .NET Upgrade Assistant v0.2.211727+27ce11e3d7656d004d6d592136fcff7507999265 -
[15:11:59 INF] Configuration loaded from context base directory: C:\Users\steve\.dotnet\tools\.store\upgrade-assistant\0.2.211727\upgrade-assistant\0.2.211727\tools\net5.0\any\
[15:11:59 INF] MSBuild registered from C:\Program Files\dotnet\sdk\5.0.200-preview.21079.7\
[15:12:00 INF] Registered 1 extensions:
  Default extensions
[15:12:00 INF] Initializing migration step Select an entrypoint
[15:12:00 INF] Setting entrypoint to only project in solution: C:\dev\Scratch\upgradeassistant\AspNetMvcTest\AspNetMvcTest.csproj
[15:12:00 INF] Initializing migration step Select project to upgrade
[15:12:00 INF] Setting only project in solution as the current project: C:\dev\Scratch\upgradeassistant\AspNetMvcTest\AspNetMvcTest.csproj
[15:12:00 INF] Initializing migration step Backup project

Migration Steps

EntryPoint: C:\dev\Scratch\upgradeassistant\AspNetMvcTest\AspNetMvcTest.csproj
Current Project: C:\dev\Scratch\upgradeassistant\AspNetMvcTest\AspNetMvcTest.csproj

1. [Next step] Backup project
2. Convert project file to SDK style
3. Update TFM
4. Update NuGet packages
5. Add template files
6. Migrate app config files
  a. Migrate appSettings
  b. Disable unsupported configuration sections
  c. Migrate system.web.webPages.razor/pages/namespaces
7. Update C# source
  a. Apply fix for AM0001: ASP.NET Core projects should not reference ASP.NET namespaces
  b. Apply fix for AM0002: HtmlString types should be replaced with Microsoft.AspNetCore.Html.HtmlString
  c. Apply fix for AM0003: ActionResult types should come from the Microsoft.AspNetCore.Mvc namespace
  d. Apply fix for AM0004: Filter types should be used from the Microsoft.AspNetCore.Mvc.Filters namespace
  e. Apply fix for AM0005: Do not use HttpContext.Current
  f. Apply fix for AM0006: HttpContext.DebuggerEnabled should be replaced with System.Diagnostics.Debugger.IsAttached
  g. Apply fix for AM0007: HtmlHelper should be replaced with IHtmlHelper
  h. Apply fix for AM0008: UrlHelper should be replaced with IUrlHelper
  i. Apply fix for AM0009: HelperResult should be replaced with Microsoft.AspNetCore.Mvc.Razor.HelperResult
  j. Apply fix for AM0010: [AllowHtmlAttribute] should be removed
8. Move to next project

Choose a command:
  1. Apply next step (Backup project)
  2. Skip next step (Backup project)
  3. See more step details
  4. Configure logging
  5. Exit
>
```

As each step is completed, the tool provides a set of commands allowing the user to apply or skip the next step, see more details, configure logging, or exit the process. If the tool detects that a step will perform no actions, it automatically skips that step and continues to the next step until it reaches one that has actions to do. Pressing Enter will start the next step if no other selection is made.

In this example, the apply step is chosen each time. The first step is to back up the project.

```
C:\dev\Scratch\upgradeassistant\AspNetMvcTest
pNetMvcTest.csproj
[15:12:00 INF] Initializing migration step Backup project

Migration Steps

Entrypoint: C:\dev\Scratch\upgradeassistant\AspNetMvcTest\AspNetMvcTest.csproj
Current Project: C:\dev\Scratch\upgradeassistant\AspNetMvcTest\AspNetMvcTest.csproj

1. [Next step] Backup project
2. Convert project file to SDK style
3. Update TFM
4. Update NuGet packages
5. Add template files
6. Migrate app config files
  a. Migrate appSettings
  b. Disable unsupported configuration sections
  c. Migrate system.web.webPages.razor/pages/namespaces
7. Update C# source
  a. Apply fix for AM0001: ASP.NET Core projects should not reference ASP.NET namespaces
  b. Apply fix for AM0002: HtmlString types should be replaced with Microsoft.AspNetCore.Html.HtmlString
  c. Apply fix for AM0003: ActionResult types should come from the Microsoft.AspNetCore.Mvc namespace
  d. Apply fix for AM0004: Filter types should be used from the Microsoft.AspNetCore.Mvc.Filters namespace
  e. Apply fix for AM0005: Do not use HttpContext.Current
  f. Apply fix for AM0006: HttpContext.DebuggerEnabled should be replaced with System.Diagnostics.Debugger.IsAttached
  g. Apply fix for AM0007: HtmlHelper should be replaced with IHtmlHelper
  h. Apply fix for AM0008: UrlHelper should be replaced with IUrlHelper
  i. Apply fix for AM0009: HelperResult should be replaced with Microsoft.AspNetCore.Mvc.Razor.HelperResult
  j. Apply fix for AM0010: [AllowHtmlAttribute] should be removed
8. Move to next project

Choose a command:
  1. Apply next step (Backup project)
  2. Skip next step (Backup project)
  3. See more step details
  4. Configure logging
  5. Exit
> 1
[15:13:58 INF] Applying migration step Backup project
Please choose a backup path
  1. Use default path [C:\dev\Scratch\upgradeassistant\AspNetMvcTest.backup]
  2. Enter custom path
> 1
[15:14:04 INF] Backing up C:\dev\Scratch\upgradeassistant\AspNetMvcTest to C:\dev\Scratch\upgradeassistant\AspNetMvcTest.backup
[15:14:04 WRN] Could not copy file C:\dev\Scratch\upgradeassistant\AspNetMvcTest\log.txt due to 'The process cannot access the file 'C:\dev\Scratch\upgradeassistant\AspNetMvcTest\log.txt' because it is being used by another process.'
[15:14:05 INF] Project backed up to C:\dev\Scratch\upgradeassistant\AspNetMvcTest.backup
[15:14:05 INF] Migration step Backup project applied successfully
Please press enter to continue...
```

The tool prompts for a custom path for the backup, or to use the default, which will place the project backup in the same folder with a `.backup` extension. The next step the tool does is to convert the project file to SDK style.

```
C:\dev\Scratch\upgradeassistant\AspNetMvcTest
Migration Steps

Entrypoint: C:\dev\Scratch\upgradeassistant\AspNetMvcTest\AspNetMvcTest.csproj
Current Project: C:\dev\Scratch\upgradeassistant\AspNetMvcTest\AspNetMvcTest.csproj

1. [Complete] Backup project
2. [Next step] Convert project file to SDK style
3. Update TFM
4. Update NuGet packages
5. Add template files
6. Migrate app config files
   a. Migrate appSettings
   b. Disable unsupported configuration sections
   c. Migrate system.web.webPages.razor/pages/namespaces
7. Update C# source
   a. Apply fix for AM0001: ASP.NET Core projects should not reference ASP.NET namespaces
   b. Apply fix for AM0002: HtmlString types should be replaced with Microsoft.AspNetCore.Html.HtmlString
   c. Apply fix for AM0003: ActionResult types should come from the Microsoft.AspNetCore.Mvc namespace
   d. Apply fix for AM0004: Filter types should be used from the Microsoft.AspNetCore.Mvc.Filters namespace
   e. Apply fix for AM0005: Do not use HttpContext.Current
   f. Apply fix for AM0006: HttpContext.DebuggerEnabled should be replaced with System.Diagnostics.Debugger.IsAttached
   g. Apply fix for AM0007: HtmlHelper should be replaced with IHtmlHelper
   h. Apply fix for AM0008: UrlHelper should be replaced with IUrlHelper
   i. Apply fix for AM0009: HelperResult should be replaced with Microsoft.AspNetCore.Mvc.Razor.HelperResult
   j. Apply fix for AM0010: [AllowHtmlAttribute] should be removed
8. Move to next project

Choose a command:
 1. Apply next step (Convert project file to SDK style)
 2. Skip next step (Convert project file to SDK style)
 3. See more step details
 4. Configure logging
 5. Exit
>
[12:32:41 INF] Applying migration step Convert project file to SDK style
[12:32:41 INF] Converting project file format with try-convert
[12:32:41 INF] [try-convert] C:\dev\Scratch\upgradeassistant\AspNetMvcTest\AspNetMvcTest.csproj contains a reference to System.Web, which is not supported on .NET Core. You may have significant work ahead of you to fully port this project.
[12:32:41 INF] [try-convert] 'C:\dev\Scratch\upgradeassistant\AspNetMvcTest\AspNetMvcTest.csproj' is a legacy web project and/or reference System.Web. Legacy Web projects and System.Web are unsupported on .NET Core. You will need to rewrite your application or find a way to not depend on System.Web to convert this project.
[12:32:44 INF] [try-convert] Conversion complete!
[12:32:44 INF] Project file converted successfully! The project may require additional changes to build successfully against the new .NET target.
[12:32:45 INF] [dotnet-restore] Determining projects to restore...
[12:32:45 INF] [dotnet-restore] Restored C:\dev\Scratch\upgradeassistant\AspNetMvcTest\AspNetMvcTest.csproj (in 142 ms).
[12:32:46 INF] Migration step Convert project file to SDK style applied successfully
Please press enter to continue...
```

Once the project format has been updated, the next step is to update the TFM of the project.

```
C:\dev\Scratch\upgradeassistant\AspNetMvcTest

Migration Steps

Entrypoint: C:\dev\Scratch\upgradeassistant\AspNetMvcTest\AspNetMvcTest.csproj
Current Project: C:\dev\Scratch\upgradeassistant\AspNetMvcTest\AspNetMvcTest.csproj

1. [Complete] Backup project
2. [Complete] Convert project file to SDK style
3. [Next step] Update TFM
4. Update NuGet packages
5. Add template files
6. Migrate app config files
   a. Migrate appSettings
   b. Disable unsupported configuration sections
   c. Migrate system.web.webPages.razor/pages/namespaces
7. Update C# source
   a. Apply fix for AM0001: ASP.NET Core projects should not reference ASP.NET namespaces
   b. Apply fix for AM0002: HtmlString types should be replaced with Microsoft.AspNetCore.Html.HtmlString
   c. Apply fix for AM0003: ActionResult types should come from the Microsoft.AspNetCore.Mvc namespace
   d. Apply fix for AM0004: Filter types should be used from the Microsoft.AspNetCore.Mvc.Filters namespace
   e. Apply fix for AM0005: Do not use HttpContext.Current
   f. Apply fix for AM0006: HttpContext.DebuggerEnabled should be replaced with System.Diagnostics.Debugger.IsAttached
   g. Apply fix for AM0007: HtmlHelper should be replaced with IHtmlHelper
   h. Apply fix for AM0008: UrlHelper should be replaced with IUrlHelper
   i. Apply fix for AM0009: HelperResult should be replaced with Microsoft.AspNetCore.Mvc.Razor.HelperResult
   j. Apply fix for AM0010: [AllowHtmlAttribute] should be removed
8. Move to next project

Choose a command:
 1. Apply next step (Update TFM)
 2. Skip next step (Update TFM)
 3. See more step details
 4. Configure logging
 5. Exit
>
[12:39:00 INF] Applying migration step Update TFM
[12:39:02 INF] [dotnet-restore] Determining projects to restore...
[12:39:03 INF] [dotnet-restore] C:\dev\Scratch\upgradeassistant\AspNetMvcTest\AspNetMvcTest.csproj : warning NU1701: Package 'Antlr 3.5.0.2' was restored using '.NETFramework,Version=v4.6.1, .NETFramework,Version=v4.6.2, .NETFramework,Version=v4.7, .NETFramework,Version=v4.7.1, .NETFramework,Version=v4.7.2, .NETFramework,Version=v4.8' instead of the project target framework 'net5.0'. This package may not be fully compatible with your project.
[12:39:03 INF] [dotnet-restore] C:\dev\Scratch\upgradeassistant\AspNetMvcTest\AspNetMvcTest.csproj : warning NU1701: Package 'WebGrease 1.6.0' was restored using '.NETFramework,Version=v4.6.1, .NETFramework,Version=v4.6.2, .NETFramework,Version=v4.7, .NETFramework,Version=v4.7.1, .NETFramework,Version=v4.7.2, .NETFramework,Version=v4.8' instead of the project target framework 'net5.0'. This package may not be fully compatible with your project.
[12:39:03 INF] [dotnet-restore] Restored C:\dev\Scratch\upgradeassistant\AspNetMvcTest\AspNetMvcTest.csproj (in 144 ms).
[12:39:03 INF] Migration step Update TFM applied successfully
Please press enter to continue...
```

Next, the tool updates the project's NuGet packages. Several packages need updates, and a new analyzer package is added.

```
C:\dev\Scratch\upgradeassistant\AspNetMvcTest
[12:39:32 INF] Packages to be added:
Antlr4, Version=4.6.6
Microsoft.DotNet.UpgradeAssistant.Extensions.Default.Analyzers, Version=0.2.211942

Migration Steps

Entrypoint: C:\dev\Scratch\upgradeassistant\AspNetMvcTest\AspNetMvcTest.csproj
Current Project: C:\dev\Scratch\upgradeassistant\AspNetMvcTest\AspNetMvcTest.csproj

1. [Complete] Backup project
2. [Complete] Convert project file to SDK style
3. [Complete] Update TFM
4. [Next step] Update NuGet packages
5. Add template files
6. Migrate app config files
   a. Migrate appSettings
   b. Disable unsupported configuration sections
   c. Migrate system.web.webPages.razor/pages/namespaces
7. Update C# source
   a. Apply fix for AM0001: ASP.NET Core projects should not reference ASP.NET namespaces
   b. Apply fix for AM0002: HtmlString types should be replaced with Microsoft.AspNetCore.Html.HtmlString
   c. Apply fix for AM0003: ActionResult types should come from the Microsoft.AspNetCore.Mvc namespace
   d. Apply fix for AM0004: Filter types should be used from the Microsoft.AspNetCore.Mvc.Filters namespace
   e. Apply fix for AM0005: Do not use HttpContext.Current
   f. Apply fix for AM0006: HttpContext.DebuggerEnabled should be replaced with System.Diagnostics.Debugger.IsAttached
   g. Apply fix for AM0007: HtmlHelper should be replaced with IHtmlHelper
   h. Apply fix for AM0008: UrlHelper should be replaced with IUrlHelper
   i. Apply fix for AM0009: HelperResult should be replaced with Microsoft.AspNetCore.Mvc.Razor.HelperResult
   j. Apply fix for AM0010: [AllowHtmlAttribute] should be removed
8. Move to next project

Choose a command:
 1. Apply next step (Update NuGet packages)
 2. Skip next step (Update NuGet packages)
 3. See more step details
 4. Configure logging
 5. Exit
>
[12:39:56 INF] Applying migration step Update NuGet packages
[12:39:56 INF] Removing outdated package reference: Antlr, Version=3.5.0.2
[12:39:56 INF] Removing outdated package reference: WebGrease, Version=1.6.0
[12:39:56 INF] Adding package reference: Antlr4, Version=4.6.6
[12:39:56 INF] Adding package reference: Microsoft.DotNet.UpgradeAssistant.Extensions.Default.Analyzers, Version=0.2.211942
[12:39:57 INF] [dotnet-restore] Determining projects to restore...
[12:39:58 INF] [dotnet-restore] Restored C:\dev\Scratch\upgradeassistant\AspNetMvcTest\AspNetMvcTest.csproj (in 395 ms).
[12:39:59 INF] Migration step Update NuGet packages applied successfully
Please press enter to continue...
```

Once the packages are updated, the next step is to add template files, if any. The tool notes there are four expected template items that must be added, and then adds them. The following is a list of the template files:

- Program.cs
- Startup.cs
- appsettings.json
- appsettings.Development.json

These files are used by ASP.NET Core for [app startup](#) and [configuration](#).

```
C:\dev\Scratch\upgradeassistant\AspNetMvcTest
[12:40:26 INF] Initializing migration step Add template files
[12:40:26 INF] 4 expected template items needed

Migration Steps

Entrypoint: C:\dev\Scratch\upgradeassistant\AspNetMvcTest\AspNetMvcTest.csproj
Current Project: C:\dev\Scratch\upgradeassistant\AspNetMvcTest\AspNetMvcTest.csproj

1. [Complete] Backup project
2. [Complete] Convert project file to SDK style
3. [Complete] Update TFM
4. [Complete] Update NuGet packages
5. [Next step] Add template files
6. Migrate app config files
   a. Migrate appSettings
   b. Disable unsupported configuration sections
   c. Migrate system.web.webPages.razor/pages/namespaces
7. Update C# source
   a. Apply fix for AM0001: ASP.NET Core projects should not reference ASP.NET namespaces
   b. Apply fix for AM0002: HtmlString types should be replaced with Microsoft.AspNetCore.Html.HtmlString
   c. Apply fix for AM0003: ActionResult types should come from the Microsoft.AspNetCore.Mvc namespace
   d. Apply fix for AM0004: Filter types should be used from the Microsoft.AspNetCore.Mvc.Filters namespace
   e. Apply fix for AM0005: Do not use HttpContext.Current
   f. Apply fix for AM0006: HttpContext.DebuggerEnabled should be replaced with System.Diagnostics.Debugger.IsAttached
   g. Apply fix for AM0007: HtmlHelper should be replaced with IHtmlHelper
   h. Apply fix for AM0008: UrlHelper should be replaced with IUrlHelper
   i. Apply fix for AM0009: HelperResult should be replaced with Microsoft.AspNetCore.Mvc.Razor.HelperResult
   j. Apply fix for AM0010: [AllowHtmlAttribute] should be removed
8. Move to next project

Choose a command:
 1. Apply next step (Add template files)
 2. Skip next step (Add template files)
 3. See more step details
 4. Configure logging
 5. Exit
>
[12:40:30 INF] Applying migration step Add template files
[12:40:30 INF] Added template file Program.cs from Default extension
[12:40:30 INF] Added template file Startup.cs from Default extension
[12:40:30 INF] Added template file appsettings.json from Default extension
[12:40:30 INF] Added template file appsettings.Development.json from Default extension
[12:40:31 INF] 4 template items added
[12:40:31 INF] Migration step Add template files applied successfully
Please press enter to continue...
```

Next, the tool migrates config files. The tool identifies app settings and disables unsupported configuration sections, then migrates the `appSettings` configuration values.

```
C:\dev\Scratch\upgradeassistant\AspNetMvcTest
[12:40:49 INF] Initializing migration step Migrate app config files
[12:40:50 INF] Found 4 app settings for migration: webpages:Version, webpages:Enabled, ClientValidationEnabled, UnobtrusiveJavaScriptEnabled
[12:40:50 INF] 1 web page namespace imports need migrated: AspNetMvcTest

Migration Steps

Entry point: C:\dev\Scratch\upgradeassistant\AspNetMvcTest\AspNetMvcTest.csproj
Current Project: C:\dev\Scratch\upgradeassistant\AspNetMvcTest\AspNetMvcTest.csproj

1. [Complete] Backup project
2. [Complete] Convert project file to SDK style
3. [Complete] Update TFM
4. [Complete] Update NuGet packages
5. [Complete] Add template files
6. Migrate app config files
   a. [Next step] Migrate appSettings
   b. [Complete] Disable unsupported configuration sections
   c. Migrate system.web.webPages.razor/pages/namespaces
7. Update C# source
   a. Apply fix for AM0001: ASP.NET Core projects should not reference ASP.NET namespaces
   b. Apply fix for AM0002: HtmlString types should be replaced with Microsoft.AspNetCore.Html.HtmlString
   c. Apply fix for AM0003: ActionResult types should come from the Microsoft.AspNetCore.Mvc namespace
   d. Apply fix for AM0004: Filter types should be used from the Microsoft.AspNetCore.Mvc.Filters namespace
   e. Apply fix for AM0005: Do not use HttpContext.Current
   f. Apply fix for AM0006: HttpContext.DebuggerEnabled should be replaced with System.Diagnostics.Debugger.IsAttached
   g. Apply fix for AM0007: HtmlHelper should be replaced with IHtmlHelper
   h. Apply fix for AM0008: UrlHelper should be replaced with IUrlHelper
   i. Apply fix for AM0009: HelperResult should be replaced with Microsoft.AspNetCore.Mvc.Razor.HelperResult
   j. Apply fix for AM0010: [AllowHtmlAttribute] should be removed
8. Move to next project

Choose a command:
 1. Apply next step (Migrate appSettings)
 2. Skip next step (Migrate appSettings)
 3. See more step details
 4. Configure logging
 5. Exit
>
[12:40:52 INF] Applying migration step Migrate appSettings
[12:40:52 INF] Migration step Migrate appSettings applied successfully
Please press enter to continue...
```

The tool completes the migration of config files by migrating `system.web.webPages.razor/pages/namespaces`.

```
C:\dev\Scratch\upgradeassistant\AspNetMvcTest

Migration Steps

Entrypoint: C:\dev\Scratch\upgradeassistant\AspNetMvcTest\AspNetMvcTest.csproj
Current Project: C:\dev\Scratch\upgradeassistant\AspNetMvcTest\AspNetMvcTest.csproj

1. [Complete] Backup project
2. [Complete] Convert project file to SDK style
3. [Complete] Update TFM
4. [Complete] Update NuGet packages
5. [Complete] Add template files
6. Migrate app config files
   a. [Complete] Migrate appSettings
   b. [Complete] Disable unsupported configuration sections
   c. [Next step] Migrate system.web.webPages/razor/pages/namespaces
7. Update C# source
   a. Apply fix for AM0001: ASP.NET Core projects should not reference ASP.NET namespaces
   b. Apply fix for AM0002: HtmlString types should be replaced with Microsoft.AspNetCore.Html.HtmlString
   c. Apply fix for AM0003: ActionResult types should come from the Microsoft.AspNetCore.Mvc namespace
   d. Apply fix for AM0004: Filter types should be used from the Microsoft.AspNetCore.Mvc.Filters namespace
   e. Apply fix for AM0005: Do not use HttpContext.Current
   f. Apply fix for AM0006: HttpContext.DebuggerEnabled should be replaced with System.Diagnostics.Debugger.IsAttached
   g. Apply fix for AM0007: HtmlHelper should be replaced with IHtmlHelper
   h. Apply fix for AM0008: UrlHelper should be replaced with IUrlHelper
   i. Apply fix for AM0009: HelperResult should be replaced with Microsoft.AspNetCore.Mvc.Razor.HelperResult
   j. Apply fix for AM0010: [AllowHtmlAttribute] should be removed
8. Move to next project

Choose a command:
 1. Apply next step (Migrate system.web.webPages/razor/pages/namespaces)
 2. Skip next step (Migrate system.web.webPages/razor/pages/namespaces)
 3. See more step details
 4. Configure logging
 5. Exit
>
[12:41:25 INF] Applying migration step Migrate system.web.webPages/razor/pages/namespaces
[12:41:25 INF] View imports written to C:\dev\Scratch\upgradeassistant\AspNetMvcTest\Views\_ViewImports.cshtml
[12:41:25 INF] Migration step Migrate system.web.webPages/razor/pages/namespaces applied successfully
[12:41:25 INF] Applying migration step Migrate app config files
[12:41:25 INF] Migration step Migrate app config files applied successfully
Please press enter to continue...
```

The tool applies known fixes to migrate C# references to their new counterparts.

```
C:\dev\Scratch\upgradeassistant\AspNetMvcTest

1. [Complete] Backup project
2. [Complete] Convert project file to SDK style
3. [Complete] Update TFM
4. [Complete] Update NuGet packages
5. [Complete] Add template files
6. [Complete] Migrate app config files
  a. [Complete] Migrate appSettings
  b. [Complete] Disable unsupported configuration sections
  c. [Complete] Migrate system.web.webPages.razor/pages/namespaces
7. Update C# source
  a. [Next step] Apply fix for AM0001: ASP.NET Core projects should not reference ASP.NET namespaces
  b. [Complete] Apply fix for AM0002: HtmlString types should be replaced with Microsoft.AspNetCore.Html.HtmlString
  c. Apply fix for AM0003: ActionResult types should come from the Microsoft.AspNetCore.Mvc namespace
  d. [Complete] Apply fix for AM0004: Filter types should be used from the Microsoft.AspNetCore.Mvc.Filters namespace
  e. [Complete] Apply fix for AM0005: Do not use HttpContext.Current
  f. [Complete] Apply fix for AM0006: HttpContext.DebuggerEnabled should be replaced with System.Diagnostics.Debugger.IsAttached
  g. [Complete] Apply fix for AM0007: HtmlHelper should be replaced with IHtmlHelper
  h. [Complete] Apply fix for AM0008: UrlHelper should be replaced with IUrlHelper
  i. [Complete] Apply fix for AM0009: HelperResult should be replaced with Microsoft.AspNetCore.Mvc.Razor.HelperResult
  j. [Complete] Apply fix for AM0010: [AllowHtmlAttribute] should be removed
8. Move to next project

Choose a command:
  1. Apply next step (Apply fix for AM0001: ASP.NET Core projects should not reference ASP.NET namespaces)
  2. Skip next step (Apply fix for AM0001: ASP.NET Core projects should not reference ASP.NET namespaces)
  3. See more step details
  4. Configure logging
  5. Exit
>
[12:42:14 INF] Applying migration step Apply fix for AM0001: ASP.NET Core projects should not reference ASP.NET namespaces
[12:42:15 INF] Diagnostic AM0001 fixed in C:\dev\Scratch\upgradeassistant\AspNetMvcTest\App_Start\BundleConfig.cs
[12:42:15 INF] Diagnostic AM0001 fixed in C:\dev\Scratch\upgradeassistant\AspNetMvcTest\App_Start\FilterConfig.cs
[12:42:15 INF] Diagnostic AM0001 fixed in C:\dev\Scratch\upgradeassistant\AspNetMvcTest\App_Start\RouteConfig.cs
[12:42:15 INF] Diagnostic AM0001 fixed in C:\dev\Scratch\upgradeassistant\AspNetMvcTest\Controllers\HomeController.cs
[12:42:15 INF] Diagnostic AM0001 fixed in C:\dev\Scratch\upgradeassistant\AspNetMvcTest\Global.asax.cs
[12:42:15 INF] Diagnostic AM0001 fixed in C:\dev\Scratch\upgradeassistant\AspNetMvcTest\App_Start\BundleConfig.cs
[12:42:15 INF] Diagnostic AM0001 fixed in C:\dev\Scratch\upgradeassistant\AspNetMvcTest\App_Start\FilterConfig.cs
[12:42:15 INF] Diagnostic AM0001 fixed in C:\dev\Scratch\upgradeassistant\AspNetMvcTest\App_Start\RouteConfig.cs
[12:42:15 INF] Diagnostic AM0001 fixed in C:\dev\Scratch\upgradeassistant\AspNetMvcTest\Controllers\HomeController.cs
[12:42:15 INF] Diagnostic AM0001 fixed in C:\dev\Scratch\upgradeassistant\AspNetMvcTest\Global.asax.cs
[12:42:15 INF] Diagnostic AM0001 fixed in C:\dev\Scratch\upgradeassistant\AspNetMvcTest\App_Start\RouteConfig.cs
[12:42:15 INF] Diagnostic AM0001 fixed in C:\dev\Scratch\upgradeassistant\AspNetMvcTest\Global.asax.cs
[12:42:15 INF] Diagnostic AM0001 fixed in C:\dev\Scratch\upgradeassistant\AspNetMvcTest\Global.asax.cs
[12:42:15 INF] Migration step Apply fix for AM0001: ASP.NET Core projects should not reference ASP.NET namespaces applied successfully
Please press enter to continue...
```

Since this is the last project, the next step, "Move to next project", prompts to complete the process of migrating the entire solution.

```
C:\dev\Scratch\upgradeassistant\AspNetMvcTest

[12:42:55 INF] Initializing migration step Complete Solution

Migration Steps

Entrypoint: C:\dev\Scratch\upgradeassistant\AspNetMvcTest\AspNetMvcTest.csproj

1. [Next step] Complete Solution

Choose a command:
  1. Apply next step (Complete Solution)
  2. Skip next step (Complete Solution)
  3. See more step details
  4. Configure logging
  5. Exit
>
[12:42:57 INF] Applying migration step Complete Solution
[12:42:57 INF] Migration step Complete Solution applied successfully
Please press enter to continue...
```

Once this process has completed, open the project file and review it. Look for static files like these:

```
<ItemGroup>
  <Content Include="fonts\glyphicon-halflings-regular.woff2" />
  <Content Include="fonts\glyphicon-halflings-regular.woff" />
  <Content Include="fonts\glyphicon-halflings-regular.ttf" />
  <Content Include="fonts\glyphicon-halflings-regular.eot" />
  <Content Include="Content\bootstrap.min.css.map" />
  <Content Include="Content\bootstrap.css.map" />
  <Content Include="Content\bootstrap-theme.min.css.map" />
  <Content Include="Content\bootstrap-theme.css.map" />
  <Content Include="Scripts\jquery-3.4.1.slim.min.map" />
  <Content Include="Scripts\jquery-3.4.1.min.map" />
</ItemGroup>
```

Static files that should be served by the web server should be moved to an appropriate folder within a root level folder named `wwwroot`. See [Static files in ASP.NET Core](#) for details. Once the files have been moved, the `<Content>` elements in the project file corresponding to these files can be deleted. In fact, all `<Content>` elements and their containing groups can be removed. Also, any `<PackageReference>` to a client-side library like `bootstrap` or `jquery` should be removed.

By default, the project will be converted as a class library. Change the first line's `Sdk` attribute to `Microsoft.NET.Sdk.Web` and set the `<TargetFramework>` to `net5.0`. Compile the project. At this point, the number of errors should be fairly small. When porting a new ASP.NET 4.6.1 MVC project, the remaining errors refer to files in the `App_Start` folder:

- `BundleConfig.cs`
- `FilterConfig.cs`
- `RouteConfig.cs`

These files, and the entire `App_Start` folder, can be deleted. Likewise, the `Global.asax` and `Global.asax.cs` files can be removed.

At this point the only errors that remain are related to bundling. There are [several ways to configure bundling and minification in ASP.NET Core](#). Choose whatever makes the most sense for your project.

## Troubleshooting tips

There are several known problems that can occur when using the .NET Upgrade Assistant. In some cases, these are problems with the [try-convert tool](#) that the .NET Upgrade Assistant uses internally.

The tool's [GitHub repository](#) has more troubleshooting tips and known issues.

## See also

- [Upgrade a WPF App to .NET 6](#)
- [Upgrade a Windows Forms App to .NET 6](#)
- [Overview of the .NET Upgrade Assistant](#)
- [.NET Upgrade Assistant GitHub Repository](#)

# Upgrade Assistant telemetry

9/20/2022 • 2 minutes to read • [Edit Online](#)

The [Upgrade Assistant](#) includes a telemetry feature that collects usage data. The telemetry data is used to help understand how to make improvements to the tool.

## How to opt out

The Upgrade Assistant telemetry feature is enabled by default. To opt out of the telemetry feature, set the

`DOTNET_UPGRADEASSISTANT_TELEMETRY_OPTOUT` environment variable to `1` or `true`.

- [Console](#)
- [PowerShell](#)
- [Bash](#)

Create and assign persisted environment variable, given the value.

```
:: Assigns the env var to the value
setx DOTNET_UPGRADEASSISTANT_TELEMETRY_OPTOUT="1"
```

In a new instance of the **Command Prompt**, read the environment variable.

```
:: Prints the env var value
echo %DOTNET_UPGRADEASSISTANT_TELEMETRY_OPTOUT%
```

## Disclosure

The Upgrade Assistant displays text similar to the following when you first run the tool. Text may vary slightly depending on the version of the tool you're running. This "first run" experience is how Microsoft notifies you about data collection.

```
Telemetry
-----
The .NET tools collect usage data in order to help us improve your experience.
It is collected by Microsoft and shared with the community. You can opt-out of
telemetry by setting the DOTNET_UPGRADEASSISTANT_TELEMETRY_OPTOUT environment
variable to '1' or 'true' using your favorite shell.
```

To suppress the "first run" experience text, set the `DOTNET_UPGRADEASSISTANT_SKIP_FIRST_TIME_EXPERIENCE` environment variable to `1` or `true`.

## Data points

The telemetry feature doesn't:

- Collect samples of code

The data is sent securely to Microsoft servers and held under restricted access.

Protecting your privacy is important to us. If you suspect the telemetry feature is collecting sensitive data or the

data is being insecurely or inappropriately handled, take one of the following actions:

- File an issue in the [dotnet/upgrade-assistant](#) repository.
- Send an email to [dotnet@microsoft.com](mailto:dotnet@microsoft.com) for investigation.

The telemetry feature collects the following data.

UPGRADE ASSISTANT VERSIONS	DATA
>=0.2.231802	Timestamp of invocation.
>=0.2.231802	Three-octet IP address used to determine the geographical location.
>=0.2.231802	Operating system and version.
>=0.2.231802	Runtime ID (RID) the tool is running on.
>=0.2.231802	Whether the tool is running in a container.
>=0.2.231802	Hashed Media Access Control (MAC) address: a cryptographically (SHA256) hashed and unique ID for a machine.
>=0.2.231802	Kernel version.
>=0.2.231802	Upgrade Assistant version.
>=0.2.231802	The command and argument names invoked. Actual argument values aren't collected.
>=0.2.231802	MSBuild version used.
>=0.2.231802	Hashed solution id (or hashed path if no id is available).
>=0.2.231802	Hashed project id (or hashed path if no id is available) for each project.
>=0.2.231802	Hashed project id (or hashed path if no id is available) for each entrypoint.
>=0.2.231802	For each step, the time to initialize and apply the step.
>=0.2.231802	For each step, the decision selected (for example, <code>apply</code> ).

## Additional resources

- [.NET SDK telemetry](#)
- [.NET CLI telemetry data](#)

# Breaking changes may occur when porting code

9/20/2022 • 2 minutes to read • [Edit Online](#)

Changes that affect compatibility, otherwise known as breaking changes, will occur between versions of .NET. Changes are impactful when porting from .NET Framework to .NET because of certain technologies not being available. Also, you can come across breaking changes simply because .NET is a cross-platform technology and .NET Framework isn't.

Microsoft strives to maintain a high level of compatibility between .NET versions, so while breaking changes do occur, they're carefully considered.

Before upgrading major versions, check the breaking changes documentation for changes that might affect you.

## Categories of breaking changes

There are several types of breaking changes...

- modifications to the public contract
- behavioral changes
- Platform support
- Internal implementation changes
- Code changes

For more information about what is allowed or disallowed, see [Changes that affect compatibility](#)

## Types of compatibility

Compatibility refers to the ability to compile or run code on a .NET implementation other than the one with which the code was originally developed.

There are six different ways a change can affect compatibility...

- Behavioral changes
- Binary compatibility
- Source compatibility
- Design-time compatibility
- Backwards compatibility
- Forward compatibility

For more information, see [How code changes can affect compatibility](#)

## Find breaking changes

Changes that affect compatibility are documented and should be reviewed before porting from .NET Framework to .NET or when upgrading to a newer version of .NET.

For more information, see [Breaking changes reference overview](#)

# Analyze your dependencies to port code from .NET Framework to .NET

9/20/2022 • 5 minutes to read • [Edit Online](#)

To identify the unsupported third-party dependencies in your project, you must first understand your dependencies. External dependencies are the NuGet packages or `.dll` files you reference in your project, but that you don't build yourself.

Porting your code to .NET Standard 2.0 or below ensures that it can be used with both .NET Framework and .NET. However, if you don't need to use the library with .NET Framework, consider targeting the latest version of .NET.

## Migrate your NuGet packages to `PackageReference`

.NET can't use the `packages.config` file for NuGet references. Both .NET and .NET Framework can use `PackageReference` to specify package dependencies. If you're using `packages.config` to specify your packages in your project, convert it to the `PackageReference` format.

To learn how to migrate, see the [Migrate from packages.config to PackageReference](#) article.

## Upgrade your NuGet packages

After you migrate your project to the `PackageReference` format, verify if your packages are compatible with .NET.

First, upgrade your packages to the latest version that you can. This can be done with the NuGet Package Manager UI in Visual Studio. It's likely that newer versions of your package dependencies are already compatible with .NET Core.

## Analyze your package dependencies

If you haven't already verified that your converted and upgraded package dependencies work on .NET Core, there are a few ways that you can achieve that:

### Analyze NuGet packages using nuget.org

You can see the Target Framework Monikers (TFMs) that each package supports on [nuget.org](#) under the `Dependencies` section of the package page.

Although using the site is an easier method to verify the compatibility, `Dependencies` information isn't available on the site for all packages.

### Analyze NuGet packages using NuGet Package Explorer

A NuGet package is itself a set of folders that contain platform-specific assemblies. Check if there's a folder that contains a compatible assembly inside the package.

The easiest way to inspect NuGet package folders is to use the [NuGet Package Explorer](#) tool. After installing it, use the following steps to see the folder names:

1. Open the NuGet Package Explorer.
2. Click **Open package from online feed**.
3. Search for the name of the package.
4. Select the package name from the search results and click **open**.

5. Expand the `/lib` folder on the right-hand side and look at folder names.

Look for a folder with names using one of the following patterns: `netstandardX.Y`, `netX.Y`, or `netcoreappX.Y`.

These values are the [Target Framework Monikers \(TFMs\)](#) that map to versions of [.NET Standard](#), .NET, and .NET Core, which are all compatible with .NET.

#### IMPORTANT

When looking at the TFMs that a package supports, note that a TFM other than `netstandard*` targets a specific implementation of .NET, such as .NET 5, .NET Core, or .NET Framework. Starting with .NET 5, the `net*` TFM (without an operating system designation) effectively replaces `netstandard*` as a [portable target](#). For example, `net5.0` targets the .NET 5 API surface and is cross-platform friendly, but `net5.0-windows` targets the .NET 5 API surface as implemented on the Windows operating system.

## .NET Framework compatibility mode

After analyzing the NuGet packages, you might find that they only target the .NET Framework.

Starting with .NET Standard 2.0, the .NET Framework compatibility mode was introduced. This compatibility mode allows .NET Standard and .NET Core projects to reference .NET Framework libraries. Referencing .NET Framework libraries doesn't work for all projects, such as if the library uses Windows Presentation Foundation (WPF) APIs, but it does unblock many porting scenarios.

When you reference NuGet packages that target .NET Framework in your project, such as

`Huitian.PowerCollections`, you get a package fallback warning ([NU1701](#)) similar to the following example:

NU1701: Package 'Huitian.PowerCollections 1.0.0' was restored using '.NETFramework,Version=v4.6.1' instead of the project target framework '.NETStandard,Version=v2.0'. This package may not be fully compatible with your project.

That warning is displayed when you add the package and every time you build to make sure you test that package with your project. If your project works as expected, you can suppress that warning by editing the package properties in Visual Studio or by manually editing the project file in your favorite code editor.

To suppress the warning by editing the project file, find the `PackageReference` entry for the package you want to suppress the warning for and add the `NoWarn` attribute. The `NoWarn` attribute accepts a comma-separated list of all the warning IDs. The following example shows how to suppress the `NU1701` warning for the `Huitian.PowerCollections` package by editing your project file manually:

```
<ItemGroup>
  <PackageReference Include="Huitian.PowerCollections" Version="1.0.0" NoWarn="NU1701" />
</ItemGroup>
```

For more information on how to suppress compiler warnings in Visual Studio, see [Suppressing warnings for NuGet packages](#).

## If NuGet packages won't run on .NET

There are a few things you can do if a NuGet package you depend on doesn't run on .NET Core:

- If the project is open source and hosted somewhere like GitHub, you can engage the developers directly.
- You can contact the author directly on [nuget.org](#). Search for the package and click **Contact Owners** on the left-hand side of the package's page.
- You can search for another package that runs on .NET Core that accomplishes the same task as the package you were using.

- You can attempt to write the code the package was doing yourself.
- You could eliminate the dependency on the package by changing the functionality of your app, at least until a compatible version of the package becomes available.

Remember that open-source project maintainers and NuGet package publishers are often volunteers. They contribute because they care about a given domain, do it for free, and often have a different daytime job. Be mindful of that when contacting them to ask for .NET Core support.

If you can't resolve your issue with any of these options, you may have to port to .NET Core at a later date.

The .NET Team would like to know which libraries are the most important to support with .NET Core. You can send an email to [dotnet@microsoft.com](mailto:dotnet@microsoft.com) about the libraries you'd like to use.

## Analyze non-NuGet dependencies

You may have a dependency that isn't a NuGet package, such as a DLL in the file system. The only way to determine the portability of that dependency is to run the [.NET Portability Analyzer](#) tool. The tool analyzes assemblies that target the .NET Framework and identifies APIs that aren't portable to other .NET platforms such as .NET Core. You can run the tool as a console application or as a [Visual Studio extension](#).

## Next steps

- [Overview of porting from .NET Framework to .NET](#)

# Use the Windows Compatibility Pack to port code to .NET 5+

9/20/2022 • 2 minutes to read • [Edit Online](#)

Some of the most common issues found when porting existing code from .NET Framework to .NET are dependencies on APIs and technologies that are only found in .NET Framework. The *Windows Compatibility Pack* provides many of these technologies, so it's much easier to build .NET applications and .NET Standard libraries.

The compatibility pack is a logical [extension of .NET Standard 2.0](#) that significantly increases the API set. Existing code compiles with almost no modifications. To keep its promise of "the set of APIs that all .NET implementations provide", .NET Standard doesn't include technologies that can't work across all platforms, such as registry, Windows Management Instrumentation (WMI), or reflection emit APIs. The Windows Compatibility Pack sits on top of .NET Standard and provides access to these Windows-only technologies. It's especially useful for customers that want to move to .NET but plan to stay on Windows, at least as a first step. In that scenario, you can use Windows-only technologies removes the migration hurdle.

## Package contents

The Windows Compatibility Pack is provided via the [Microsoft.Windows.Compatibility NuGet package](#) and can be referenced from projects that target .NET or .NET Standard.

It provides about 20,000 APIs, including Windows-only and cross-platform APIs from the following technology areas:

- Code Pages
- CodeDom
- Configuration
- Directory Services
- Drawing
- ODBC
- Permissions
- Ports
- Windows Access Control Lists (ACL)
- Windows Communication Foundation (WCF)
- Windows Cryptography
- Windows EventLog
- Windows Management Instrumentation (WMI)
- Windows Performance Counters
- Windows Registry
- Windows Runtime Caching
- Windows Services

For more information, see the [specification of the compatibility pack](#).

## Get started

1. Before porting, make sure to take a look at the [Porting process](#).

- When porting existing code to .NET or .NET Standard, install the [Microsoft.Windows.Compatibility NuGet package](#).

If you want to stay on Windows, you're all set.

- If you want to run the .NET application or .NET Standard library on Linux or macOS, use the [Platform compatibility analyzer](#) to find usage of APIs that won't work cross-platform.

- Either remove the usages of those APIs, replace them with cross-platform alternatives, or guard them using a platform check, like:

```
private static string GetLoggingPath()
{
    // Verify the code is running on Windows.
    if (RuntimeInformation.IsOSPlatform(OSPlatform.Windows))
    {
        using (var key = Registry.CurrentUser.OpenSubKey(@"Software\Fabrikam\AssetManagement"))
        {
            if (key?.GetValue("LoggingDirectoryPath") is string configuredPath)
                return configuredPath;
        }
    }

    // This is either not running on Windows or no logging path was configured,
    // so just use the path for non-roaming user-specific data files.
    var appDataPath = Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData);
    return Path.Combine(appDataPath, "Fabrikam", "AssetManagement", "Logging");
}
```

For a demo, check out the [Channel 9 video of the Windows Compatibility Pack](#).

## See also

- [Overview of porting from .NET Framework to .NET](#)
- [ASP.NET to ASP.NET Core migration](#)
- [Migrate .NET Framework WPF apps to .NET](#)
- [Migrate .NET Framework Windows Forms apps to .NET](#)

# .NET Framework technologies unavailable on .NET Core and .NET 5+

9/20/2022 • 3 minutes to read • [Edit Online](#)

Several technologies available to .NET Framework libraries aren't available for use with .NET 5+ (and .NET Core), such as app domains, remoting, and code access security (CAS). If your libraries rely on one or more of the technologies listed on this page, consider the alternative approaches mentioned.

For more information on API compatibility, see [Breaking changes in .NET](#).

## Application domains

Application domains (AppDomains) isolate apps from one another. AppDomains require runtime support and are resource-expensive. Creating more app domains isn't supported, and there are no plans to add this capability in the future. For code isolation, use separate processes or containers as an alternative. To dynamically load assemblies, use the [AssemblyLoadContext](#) class.

To make code migration from .NET Framework easier, .NET 5+ exposes some of the [AppDomain](#) API surface. Some of the APIs function normally (for example, [AppDomain.UnhandledException](#)), some members do nothing (for example, [SetCachePath](#)), and some of them throw [PlatformNotSupportedException](#) (for example, [CreateDomain](#)). Check the types you use against the [System.AppDomain](#) reference source in the [dotnet/runtime GitHub repository](#). Make sure to select the branch that matches your implemented version.

## Remoting

.NET Remoting isn't supported on .NET 5+ (and .NET Core). .NET remoting was identified as a problematic architecture. It's used for communicating across application domains, which are no longer supported. Also, remoting requires runtime support, which is expensive to maintain.

For simple communication across processes, consider inter-process communication (IPC) mechanisms as an alternative to remoting, such as the [System.IO.Pipes](#) class or the [MemoryMappedFile](#) class. For more complex scenarios, the open-source [StreamJsonRpc](#) project provides a cross-platform .NET Standard remoting framework that works on top of existing stream or pipe connections.

Across machines, use a network-based solution as an alternative. Preferably, use a low-overhead plain text protocol, such as HTTP. The [Kestrel web server](#), which is the web server used by ASP.NET Core, is an option here. Also, consider using [System.Net.Sockets](#) for network-based, cross-machine scenarios. StreamJsonRpc, mentioned earlier, can be used for JSON or binary (via MessagePack) communication over web sockets.

For more messaging options, see [.NET Open Source Developer Projects: Messaging](#).

## Code access security (CAS)

Sandboxing, which relies on the runtime or the framework to constrain which resources a managed application or library uses or runs, [isn't supported on .NET Framework](#) and therefore is also not supported on .NET Core and .NET 5+. CAS is no longer treated as a security boundary, because there are too many cases in .NET Framework and the runtime where an elevation of privileges occurs. Also, CAS makes the implementation more complicated and often has correctness-performance implications for applications that don't intend to use it.

Use security boundaries provided by the operating system, such as virtualization, containers, or user accounts, for running processes with the minimum set of privileges.

## Security transparency

Similar to CAS, security transparency separates sandboxed code from security critical code in a declarative fashion but is [no longer supported as a security boundary](#). This feature is heavily used by Silverlight.

To run processes with the least set of privileges, use security boundaries provided by the operating system, such as virtualization, containers, or user accounts.

## System.EnterpriseServices

[System.EnterpriseServices](#) (COM+) isn't supported by .NET Core and .NET 5+.

## Workflow Foundation

Windows Workflow Foundation (WF) is not supported in .NET 5+ (including .NET Core). For an alternative, see [CoreWF](#).

### TIP

Windows Communication Foundation (WCF) server can be used in .NET 5+ by using the [CoreWCF NuGet packages](#). For more information, see [CoreWCF 1.0 has been Released](#).

## Saving assemblies generated by reflection

.NET 5+ (including .NET Core) does not support saving assemblies that are generated by the [System.Reflection.Emit](#) APIs. The [AssemblyBuilder.Save](#) method is not available in .NET 5+ (including .NET Core). In addition, the following fields of the [AssemblyBuilderAccess](#) enumeration aren't available:

- [ReflectionOnly](#)
- [RunAndSave](#)
- [Save](#)

As an alternative, consider the [ILPack library](#).

For more information, see [dotnet/runtime issue 15704](#).

## Loading multi-module assemblies

Assemblies that consist of multiple modules (`outputType=Module` in MSBuild) are not supported in .NET 5+ (including .NET Core).

As an alternative, consider merging the individual modules into a single assembly file.

## XSLT script blocks

XSLT [script blocks](#) are supported only in .NET Framework. They are not supported on .NET Core or .NET 5 or later.

## See also

- [Overview of porting from .NET Framework to .NET](#)

# Find unsupported APIs in your code

9/20/2022 • 2 minutes to read • [Edit Online](#)

APIs in your .NET Framework code may not be supported in .NET for many reasons. These reasons range from the simple to fix, such as a namespace change, to the more challenging to fix, such as an entire technology not being supported. The first step is to determine which of your APIs are no longer supported and then identify the proper fix.

## .NET Portability Analyzer

The .NET Portability Analyzer is a tool that analyzes assemblies and provides a detailed report on .NET APIs that are missing for the applications or libraries to be portable on your specified targeted .NET platforms.

To use the .NET Portability Analyzer in Visual Studio, install the [extension from the marketplace](#).

For more information, see [The .NET Portability Analyzer](#).

## Upgrade assistant

The [.NET Upgrade Assistant](#) is a command-line tool that can be run on different kinds of .NET Framework apps. It's designed to assist with upgrading .NET Framework apps to .NET 5. After running the tool, in most cases, the app will **require more effort to complete the migration**. The tool includes the installation of analyzers that can assist with completing the migration. This tool works on the following types of .NET Framework applications:

- Windows Forms
- WPF
- ASP.NET MVC
- Console
- Class libraries

This tool uses the [.NET Portability Analyzer](#) among other tools, and guides the migration process. For more information about the tool, see [Overview of the .NET Upgrade Assistant](#).

# Prerequisites to porting code

9/20/2022 • 2 minutes to read • [Edit Online](#)

Make the needed changes to build and run a .NET application before beginning the work to port your code. These changes can be done while still building and running a .NET Framework application.

## Upgrade to required tooling

Upgrade to a version of MSBuild/Visual Studio that supports the version of .NET you will be targeting. See [Versioning relationship between the .NET SDK, MSBuild and VS](#) for more info.

## Update .NET Framework target version

We recommend that you target your .NET Framework app to version 4.7.2 or higher. This ensures the availability of the latest API alternatives for cases where .NET Standard doesn't support existing APIs.

For each of the projects you wish to port, do the following in Visual Studio:

1. Right-click on the project and select **Properties**.
2. In the **Target Framework** dropdown, select **.NET Framework 4.7.2**.
3. Recompile the project.

Because your projects now target .NET Framework 4.7.2, use that version of the .NET Framework as your base for porting code.

## Change to PackageReference format

Convert all references to the [PackageReference](#) format.

## Convert to SDK style project format

Convert your projects to the [SDK-style format](#).

## Update dependencies

Update dependencies to their latest version available, and to .NET Standard version where possible.

## Next steps

- [Create a porting plan](#)

# Create a porting plan

9/20/2022 • 5 minutes to read • [Edit Online](#)

Before you jump straight into the code, take the time to go through the recommended pre-migration steps. This article gives you insight into the kinds of issues you may come across, and helps you decide on an approach that makes the most sense.

## Port your code

Make sure that you follow the [prerequisites to porting code](#) before you continue any further. Be ready to decide on the best approach for you and begin porting code.

### Deal primarily with the compiler

This approach works well for small projects or projects that don't use many .NET Framework APIs. The approach is simple:

1. Optionally, run **ApiPort** on your project. If you run **ApiPort**, gain knowledge from the report on issues you'll need to address.
2. Copy all of your code over into a new .NET project.
3. While referring to the portability report (if generated), solve compiler errors until the project fully compiles.

Although it's unstructured, this code-focused approach often resolves issues quickly. A project that contains only data models might be an ideal candidate for this approach.

### Stay on the .NET Framework until portability issues are resolved

This approach might be the best if you prefer to have code that compiles during the entire process. The approach is as follows:

1. Run **ApiPort** on a project.
2. Address issues by using different APIs that are portable.
3. Take note of any areas where you're prevented from using a direct alternative.
4. Repeat the prior steps for all projects you're porting until you're confident each is ready to be copied over into a new .NET project.
5. Copy the code into a new .NET project.
6. Work out any issues where you noted that a direct alternative doesn't exist.

This careful approach is more structured than simply working out compiler errors, but it's still relatively code-focused and has the benefit of always having code that compiles. The way you resolve certain issues that couldn't be addressed by just using another API varies greatly. You may find that you need to develop a more comprehensive plan for certain projects, which is covered in the next approach.

### Develop a comprehensive plan of attack

This approach might be best for larger and more complex projects, where restructuring code or completely rewriting certain areas of code might be necessary to support .NET. The approach is as follows:

1. Run **ApiPort** on a project.
2. Understand where each non-portable type is used and how that affects overall portability.
  - Understand the nature of those types. Are they small in number but used frequently? Are they large in number but used infrequently? Is their use concentrated, or is it spread throughout your code?

- Is it easy to isolate code that isn't portable so that you can deal with it more effectively?
  - Do you need to refactor your code?
  - For those types that aren't portable, are there alternative APIs that accomplish the same task? For example, if you're using the [WebClient](#) class, use the [HttpClient](#) class instead.
  - Are there different portable APIs available to accomplish a task, even if it's not a drop-in replacement? For example, if you're using [XmlSchema](#) to parse XML but don't require XML schema discovery, you could use [System.Xml.Linq](#) APIs and implement parsing yourself instead of relying on an API.
3. If you have assemblies that are difficult to port, is it worth leaving them on .NET Framework for now? Here are some things to consider:
- You may have some functionality in your library that's incompatible with .NET because it relies too heavily on .NET Framework or Windows-specific functionality. Is it worth leaving that functionality behind for now and releasing a temporary .NET version of your library with fewer features until resources are available to port the features?
  - Would a refactor help?
4. Is it reasonable to write your own implementation of an unavailable .NET Framework API?

You could consider copying, modifying, and using code from the [.NET Framework reference source](#). The reference source code is licensed under the [MIT License](#), so you have significant freedom to use the source as a basis for your own code. Just be sure to properly attribute Microsoft in your code.

5. Repeat this process as needed for different projects.

The analysis phase could take some time depending on the size of your codebase. Spending time in this phase to thoroughly understand the scope of changes needed and to develop a plan usually saves you time in the end, particularly if you have a complex codebase.

Your plan could involve making significant changes to your codebase while still targeting .NET Framework 4.7.2. This is a more structured version of the previous approach. How you go about executing your plan is dependent on your codebase.

### Mixed approach

It's likely that you'll mix the above approaches on a per-project basis. Do what makes the most sense to you and for your codebase.

## Port your tests

The best way to make sure everything works when you've ported your code is to test your code as you port it to .NET. To do this, you'll need to use a testing framework that builds and runs tests for .NET. Currently, you have three options:

- [xUnit](#)
  - [Getting Started](#)
  - [Tool to convert an MSTest project to xUnit](#)
- [NUnit](#)
  - [Getting Started](#)
  - [Blog post about migrating from MSTest to NUnit](#)
- [MSTest](#)

## Recommended approach

Ultimately, the porting effort depends heavily on how your .NET Framework code is structured. A good way to port your code is to begin with the *base* of your library, which is the foundational components of your code. This might be data models or some other foundational classes and methods that everything else uses directly or

indirectly.

1. Port the test project that tests the layer of your library that you're currently porting.
2. Copy over the base of your library into a new .NET project and select the version of .NET Standard you wish to support.
3. Make any changes needed to get the code to compile. Much of this may require adding NuGet package dependencies to your *csproj* file.
4. Run the tests and make any needed adjustments.
5. Pick the next layer of code to port over and repeat the prior steps.

If you start with the base of your library and move outward from the base and test each layer as needed, porting is a systematic process where problems are isolated to one layer of code at a time.

## Next steps

- [Overview of the .NET Upgrade Assistant](#)
- [Organize your project to support both .NET Framework and .NET Core](#)

# Organize your project to support both .NET Framework and .NET

9/20/2022 • 2 minutes to read • [Edit Online](#)

You can create a solution that compiles for both .NET Framework and .NET side by side. This article covers several project-organization options to help you achieve this goal. Here are some typical scenarios to consider when you're deciding how to set up your project layout with .NET. The list may not cover everything you want.

- [Combine existing projects and .NET projects into a single project](#)

## Benefits:

- Simplifies your build process by compiling a single project rather than multiple projects that each target a different .NET Framework version or platform.
- Simplifies source file management for multi-targeted projects because you must manage a single project file. When adding or removing source files, the alternatives require you to manually sync these files with your other projects.
- Easily generate a NuGet package for consumption.
- Allows you to write code for a specific .NET Framework version by using compiler directives.

## Drawback:

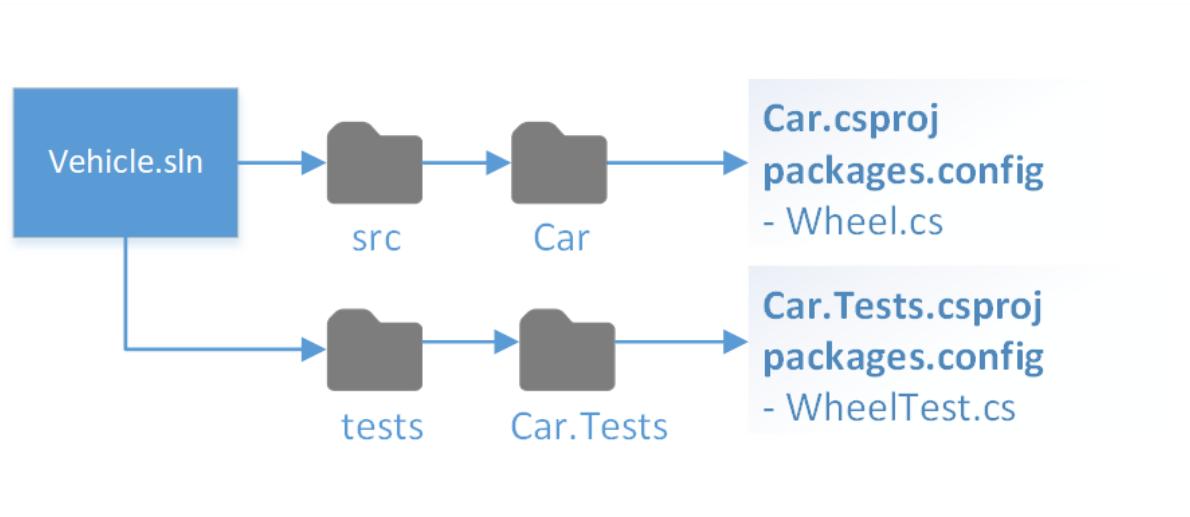
- Requires developers to use Visual Studio 2019 or a later version to open existing projects. To support older versions of Visual Studio, [keeping your project files in different folders](#) is a better option.

- [Keep all projects separate](#)

## Benefits:

- Supports development on existing projects for developers and contributors who may not have Visual Studio 2019 or a later version.
- Lowers the possibility of creating new bugs in existing projects because no code churn is required in those projects.

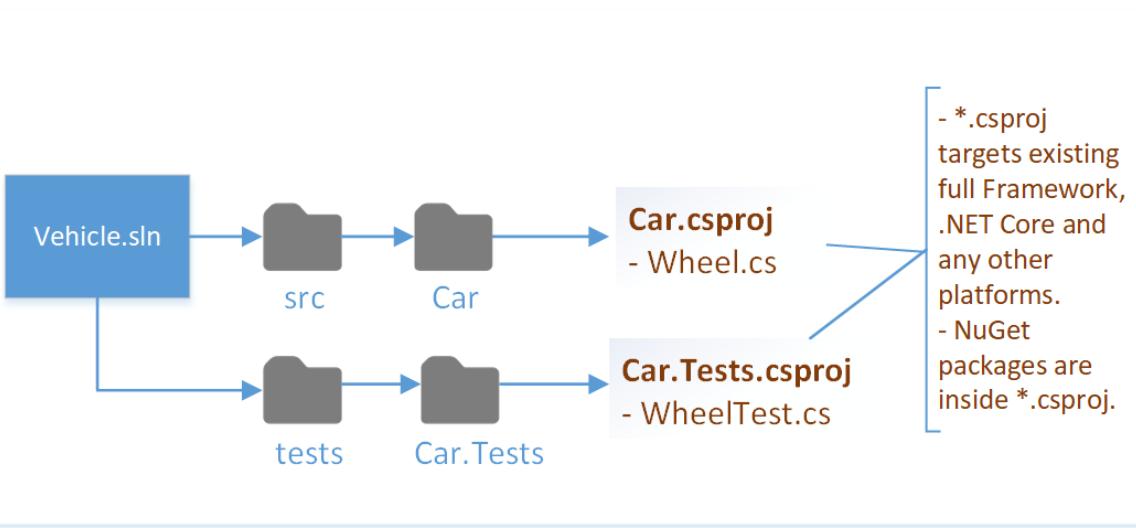
Consider [this example GitHub repository](#). The figure below shows how this repository is laid out:



The following sections describe several ways to add support for .NET based on the example repository.

## Replace existing projects with a multi-targeted .NET project

Reorganize the repository so that any existing `*.csproj` files are removed and a single `*.csproj` file is created that targets multiple frameworks. This is a great option, because a single project can compile for different frameworks. It also has the power to handle different compilation options and dependencies per targeted framework.



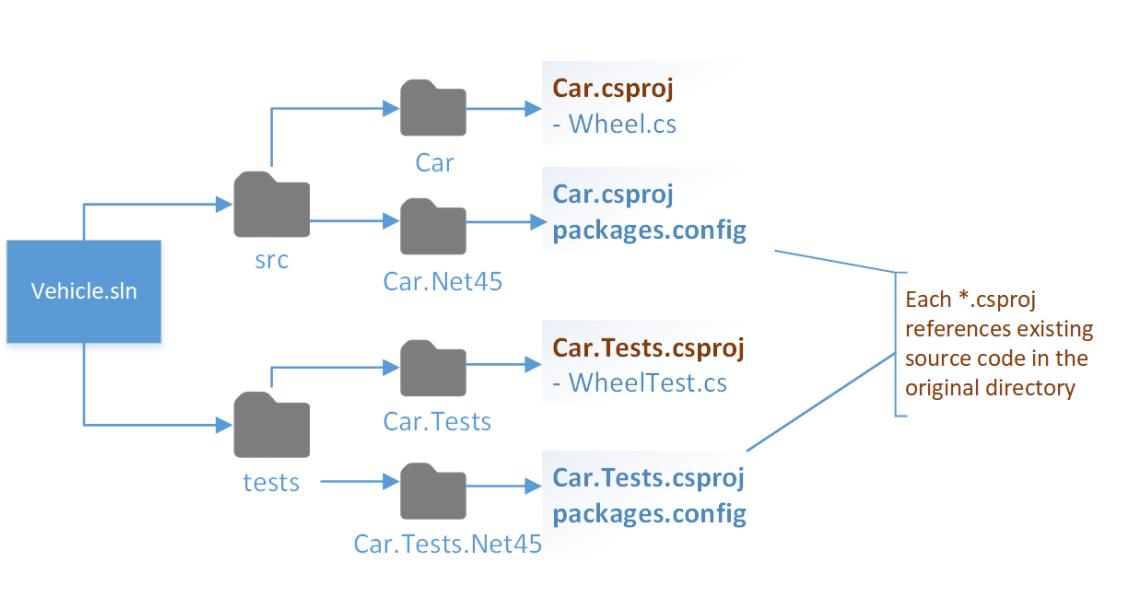
For example code, see [GitHub](#).

Changes to note are:

- Replacement of `packages.config` and `*.csproj` with a new .NET `*.csproj`. NuGet packages are specified with `<PackageReference>` `ItemGroup`.

## Keep existing projects and create a .NET project

If there are existing projects that target older frameworks, you may want to leave these projects untouched and use a .NET project to target future frameworks.



For example code, see [GitHub](#).

The .NET and existing projects are kept in separate folders. Keeping projects in separate folders avoids forcing you to have Visual Studio 2019 or later versions. You can create a separate solution that only opens the old

projects.

## See also

- [.NET porting documentation](#)

# How to port a C++/CLI project to .NET Core or .NET 5

9/20/2022 • 4 minutes to read • [Edit Online](#)

Beginning with .NET Core 3.1 and Visual Studio 2019, [C++/CLI projects](#) can target .NET Core. This support makes it possible to port Windows desktop applications with C++/CLI interop layers to .NET Core/.NET 5+. This article describes how to port C++/CLI projects from .NET Framework to .NET Core 3.1.

## C++/CLI .NET Core limitations

There are some important limitations to porting C++/CLI projects to .NET Core compared to other languages:

- C++/CLI support for .NET Core is Windows only.
- C++/CLI projects can't target .NET Standard, only .NET Core (or .NET Framework).
- C++/CLI projects don't support the new SDK-style project file format. Instead, even when targeting .NET Core, C++/CLI projects use the existing vcxproj file format.
- C++/CLI projects can't multitarget multiple .NET platforms. If you need to build a C++/CLI project for both .NET Framework and .NET Core, use separate project files.
- .NET Core doesn't support `-clr:pure` or `-clr:safe` compilation, only the new `-clr:netcore` option (which is equivalent to `-clr` for .NET Framework).

## Port a C++/CLI project

To port a C++/CLI project to .NET Core, make the following changes to the `.vcxproj` file. These migration steps differ from the steps needed for other project types because C++/CLI projects don't use SDK-style project files.

1. Replace `<CLRSupport>true</CLRSupport>` properties with `<CLRSupport>NetCore</CLRSupport>`. This property is often in configuration-specific property groups, so you may need to replace it in multiple places.
2. Replace `<TargetFrameworkVersion>` properties with `<TargetFramework>netcoreapp3.1</TargetFramework>`.
3. Remove any .NET Framework references (like `<Reference Include="System" />`). .NET Core SDK assemblies are automatically referenced when using `<CLRSupport>NetCore</CLRSupport>`.
4. Update API usage in `.cpp` files, as necessary, to remove APIs unavailable to .NET Core. Because C++/CLI projects tend to be fairly thin interop layers, there are often not many changes needed. You can use the [.NET Portability Analyzer](#) to identify unsupported .NET APIs used by C++/CLI binaries just as with purely managed binaries.

### WPF and Windows Forms usage

.NET Core C++/CLI projects can use Windows Forms and WPF APIs. To use these Windows desktop APIs, you need to add explicit framework references to the UI libraries. SDK-style projects that use Windows desktop APIs reference the necessary framework libraries automatically by using the `Microsoft.NET.Sdk.WindowsDesktop` SDK. Because C++/CLI projects don't use the SDK-style project format, they need to add explicit framework references when targeting .NET Core.

To use Windows Forms APIs, add this reference to the `.vcxproj` file:

```
<!-- Reference all of Windows Forms -->
<FrameworkReference Include="Microsoft.WindowsDesktop.App.WindowsForms" />
```

To use WPF APIs, add this reference to the `.vcxproj` file:

```
<!-- Reference all of WPF -->
<FrameworkReference Include="Microsoft.WindowsDesktop.App.WPF" />
```

To use both Windows Forms and WPF APIs, add this reference to the `.vcxproj` file:

```
<!-- Reference the entirety of the Windows desktop framework:
     Windows Forms, WPF, and the types that provide integration between them -->
<FrameworkReference Include="Microsoft.WindowsDesktop.App" />
```

Currently, it's not possible to add these references using Visual Studio's reference manager. Instead, update the project file by editing it manually. In Visual Studio, you'll need to unload the project first. You can also use another editor like Visual Studio Code.

## Build without MSBuild

It's also possible to build C++/CLI projects without using MSBuild. Follow these steps to build a C++/CLI project for .NET Core directly with `cl.exe` and `link.exe`.

1. When compiling, pass `-clr:netcore` to `cl.exe`.
2. Reference necessary .NET Core reference assemblies.
3. When linking, provide the .NET Core app host directory as a `LibPath` (so that `ijwhost.lib` can be found).
4. Copy `ijwhost.dll` (from the .NET Core app host directory) to the project's output directory.
5. Make sure a `runtimesconfig.json` file exists for the first component of the application that will run managed code. If the application has a managed entry point, a `runtime.config` file will be created and copied automatically. If the application has a native entry point, though, you need to create a `runtimesconfig.json` file for the first C++/CLI library to use the .NET Core runtime.

## Known issues

There are a few known issues to look out for when working with C++/CLI projects that target .NET Core 3.1 or .NET 5+:

- A WPF framework reference in .NET Core C++/CLI projects currently causes some extraneous warnings about being unable to import symbols. These warnings can be safely ignored and should be fixed soon.
- If the application has a native entry point, the C++/CLI library that first executes managed code needs a `runtimesconfig.json` file. This config file is used when the .NET Core runtime starts. C++/CLI projects don't create `runtimesconfig.json` files automatically at build time yet, so the file must be generated manually. If a C++/CLI library is called from a managed entry point, then the C++/CLI library doesn't need a `runtimesconfig.json` file (since the entry point assembly will have one that is used when starting the runtime). A simple sample `runtimesconfig.json` file is shown below. For more information, see the [spec on GitHub](#).

```
{
  "runtimesOptions": {
    "tfm": "netcoreapp3.1",
    "framework": {
      "name": "Microsoft.NETCore.App",
      "version": "3.1.0"
    }
  }
}
```

- On Windows 7, loading a .NET Core C++/CLI assembly when the entry application is native may exhibit failing behavior. This failing behavior is due to the Windows 7 loader not respecting non-`mscoree.dll` entry points for C++/CLI assemblies. The recommended course of action is to convert the entry application to managed code. Scenarios involving Thread Local Storage (TLS) are specifically unsupported in all cases on Windows 7.
- C++/CLI assemblies may be loaded multiple times, each time into a new `AssemblyLoadContext`. If the first time that managed code in a C++/CLI assembly is executed is from a native caller, the assembly is loaded into a separate `AssemblyLoadContext`. If the first time that managed code is executed is from a managed caller, the assembly is loaded into the same `AssemblyLoadContext` as the caller (usually the default). To always load your C++/CLI assembly into the default `AssemblyLoadContext`, add an "initialize" style call from your entry-point assembly to your C++/CLI assembly. For more information, see this [dotnet/runtime issue](#).

This limitation is removed starting in .NET 7. C++/CLI assemblies that target .NET 7 or a later version are always loaded into the default `AssemblyLoadContext`.